Under consideration for publication in Formal Aspects of Computing

# Exploring Tabular Verification and Refinement

Emil Sekerinski

Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada

**Abstract.** Tabular representations have been proposed for structuring complex mathematical expressions as they appear in the specification of programs. We argue that tables not only help in writing and checking complex expressions, but also in their formal manipulation. More specifically, we explore the use of tabular predicates and tabular relations in program verification and refinement.

**Keywords:** tabular expressions, program specification, program correctness, program development

## 1. Introduction

The use of tabular expressions in the specification of programs is motivated by the observation that, by the very nature of digital computers, the input/output behaviour of programs exhibits many "discontinuities". As discontinuities are less common in analogue systems, tabular expressions are meant to describe discontinuities in a more convenient way than traditional mathematics [Par92]. The present work continues this line by exploring how the tabular representation of expressions helps in coping with discontinuities in the process of verifying and refining programs.

The expressions we are dealing with are predicates—as they allow an abstract specification of the input/output behaviour—and relations—as they model nondeterministic programs. Figure 1 gives an example of a tabular predicate that is used to define a relation. The example is that of an elevator whose state is represented by integer variable *floor* for the current floor, variable *reqs*, a set of integers, for the floors to which requests exist, and variable *mode* with values *up*, *down*, *waiting* for the current direction of the elevator. We use the convention that unprimed variables refer to initial values and primed variables refer to final values. The table specifies the operation $ButtonPressed\,(f)$ of requesting the elevator at floor $f$. The table shows how the conditions that express the discontinuities are arranged along two axes, the top and the left header. Thus the table specifies six cases.

The structure of tables offers a natural way of checking their soundness: analysing the *coverage* of each header helps to determine if the specification is *complete* and analysing the *disjointness* of each header helps

*Correspondence and offprint requests to*: Emil Sekerinski, Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, Ontario L8S 4L7, Canada. e-mail: emil@mcmaster.ca

| | $mode = waiting$ | $mode \neq waiting$ |
|---|---|---|
| $f > floor$ | $reqs' = \{f\} \wedge$<br>$mode' = up$ | $reqs' = reqs \cup \{f\} \wedge$<br>$mode' = mode$ |
| $f = floor$ | $reqs' = \{\} \wedge$<br>$mode' = waiting$ | $reqs' = reqs \wedge$<br>$mode' = mode$ |
| $f < floor$ | $reqs' = \{f\} \wedge$<br>$mode' = down$ | $reqs' = reqs \cup \{f\} \wedge$<br>$mode' = mode$ |

**Fig. 1.** A tabular predicate used for defining the relation $ButtonPressed\,(f)$

to determine if the specification is *consistent* in the sense that there is no unwanted nondeterminism [HJL96]. In the example, both headers cover *true* as the disjunction of all predicates of one header is *true*, and both headers are disjoint, as the conjunction of any two header predicates of one header is *false*.

Compared to using conditional expressions for case splits, nested structures are avoided, making the whole expression easier to parse and making it less likely that cases are overlooked. Compared to (finite) state diagrams for expressing state transitions like in the example, it is again less likely that cases are overlooked. In a sense, combinations of conditions that appear in the headers and the corresponding entry in the body describe little scenarios. Compared to notations that describe scenarios purely as a sequence of events, tables are more precise because they explicitly refer to a state.

Wilder and Tucker give example of the use of tables in everyday life, in computing theory, and in software development and discuss readability issues [WT95]. They trace back early use of tables to the Babylonian period and argue for the potential of tables.

The use of tables for software development has been demonstrated by a number of projects and tools [Abr97, HL96, HJL96, Hen80, LMFM00, LHHR94]. Parnas proposes ten kinds of tables [Par92], which Janicki and Khedri further unify and generalise [Jan95, JK01]. Zucker gives transformation rules between two kinds of tables proposed by Parnas, normal function tables and inverted function tables [Zuc96]. By comparison, here we restrict ourselves to tables representing predicates and relations. Tables have also been added to the PVS theorem prover [ORS97]; PVS treats tables like nested conditionals.

The present work explores how tabular predicates and tabular relations help in formal manipulations that occur in the process of verifying and refining specifications. We argue that

1. the structure of tables leads to a natural way of decomposing their manipulation, and
2. tabular manipulation rules are easier to memorise and apply than their textual counterparts.

We derive a number of theorems supporting these claims. It turns out that many of the theorems have an intuitive interpretation, but the side conditions are less obvious. The technical contribution of this work is to derive those less obvious side conditions.

In comparison to other treatments of tables, we give meaning to all tables, irrespective of the coverage and disjointness of their headers. If a theorem requires certain coverage or the disjointness of headers, we state so explicitly. We find this more insightful and mathematically pleasing than always requiring complete coverage and disjointness; beside that, these properties are not necessarily preserved by transformations, but we do like to give a meaning to the outcome of each transformation we consider.

We start by introducing the notation for tabular predicates together with some basic transformations in Section 2 and study boolean operations on tabular predicates in Section 3. We continue by introducing tabular relations together with basic transformations and operations in Section 4. One main contribution are the verification rules in Section 5. This is followed by a discussion of refinement in Section 6 and the other main contribution, the refinement rules in Section 7. We conclude with a summary and an outlook of open issues.

## 2. Tabular Predicates

We assume that every expression $e$ has a unique type $T$, written $e : T$. For a function $f$ of type $T \to U$ the application to argument $e$ of type $T$ is written as $f\ e$. Predicates are expressions of type *Bool*, with values *true* and *false*. On predicates we use the operators $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication), $\Leftarrow$ (consequence). We also use generalised conjunction $\wedge i \in I \bullet p_i$ and generalised disjunction

$\vee i \in I \bullet p_i$. For predicates $\equiv$ (equivalence) has the same meaning as $=$ except that $\equiv$ has a lower precedence than all other operators on predicates but $=$ has a higher precedence. Distinguishing $\equiv$ from $=$ allows us to omit parentheses in expressions like $Id\ x\ y \equiv x = y$ and allows us to state that $\equiv$ is always associative.

Tabular predicates are predicates written as a disjunction of conjunctions. A tabular predicate with one *header* consisting of predicates $p$, $q$, $r$ and a *body* consisting of predicates $s$, $t$, $u$ is defined by:

$$\begin{array}{c|c|c} p & q & r \\ \hline s & t & u \end{array} \quad \equiv \quad (p \wedge s) \vee (q \wedge t) \vee (r \wedge u)$$

In general, let $I$ be a finite and non-empty set of indices and let $pv$ be an indexed collection of predicates that we call a *vector*, with elements $pv_i$ for $i \in I$. Tables with a single header are *one-dimensional*. With $pv$ and $qv$ vectors over the same index set, we introduce a shorthand for a table with header $pv$, and body $qv$, defined by generalising above example:

$$\begin{array}{c} pv \\ \hline qv \end{array} \quad \equiv \quad \vee\, i \bullet pv_i \wedge qv_i$$

On vectors $pv$ and $qv$ over the same index set $\neg pv$, $pv \wedge qv$, $pv \vee qv$, $pv \Rightarrow qv$, $pv \Leftarrow qv$, and $pv \equiv qv$ are all defined by the pointwise extension of the corresponding operators on *Bool*, e.g. $(\neg pv)_i \equiv \neg pv_i$ and $(pv \wedge qv)_i \equiv pv_i \wedge qv_i$. On occasion we identify a predicate $p$ with a vector with all elements being $p$. This also allows us to write expressions like $p \wedge pv$, with the meaning of $(p \wedge pv)_i \equiv p \wedge pv_i$, and similarly for other boolean operators.

In general, an $n$-dimensional table has $n$ headers; here we restrict ourselves to one- and two-dimensional tables. Let $I$ and $J$ be index sets, let $pv$ be an $I$-indexed vector, let $qv$ be a $J$-indexed vector, and let $rm$ be a doubly indexed collection of predicates that we call a *matrix*, with elements $rm_{i,j}$ for $i \in I$ and $j \in J$. We introduce a shorthand for a two-dimensional tabular predicate with headers $pv$, $qv$ and body $rm$:

$$\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \quad \equiv \quad \vee i, j \bullet pv_i \wedge qv_j \wedge rm_{i,j}$$

We also use a shorthand with multiple vectors in one header, with the special case of one vector being a single predicate:

$$\begin{array}{c|c|c} & qv & rv \\ \hline pv & qm & rm \end{array} \quad \equiv \quad \vee i \bullet (\vee j \bullet pv_i \wedge qv_j \wedge qm_{i,j}) \vee (\vee k \bullet pv_i \wedge rv_k \wedge rm_{i,k})$$

On matrices $rm$ and $sm$ over the same index sets $\neg rm$, $rm \wedge sm$, $rm \vee sm$, $rm \Rightarrow sm$, $rm \Leftarrow sm$, and $rm \equiv sm$ are all defined by the pointwise extension of the corresponding operators on *Bool*, e.g. $(\neg rm)_{i,j} \equiv \neg rm_{i,j}$ and $(rm \wedge sm)_{i,j} \equiv rm_{i,j} \wedge sm_{i,j}$. On occasion we identify a predicate $p$ with a matrix with all elements being $p$. This also allows us to write expressions like $p \wedge pm$, with the meaning of $(p \wedge pm)_{i,j} \equiv p \wedge pm_{i,j}$, and similarly for other boolean operators. We will also identify a vector $pv$ with a matrix with all columns being $pv$. This allows us to write expressions like $pv \wedge pm$, with the meaning of $(pv \wedge pm)_{i,j} \equiv pv_i \wedge pm_{i,j}$, and similarly for other boolean operators.

For a matrix $rm$, we define its transposition $rm^T$ to swap rows and columns, formally $rm^T_{j,i} = rm_{i,j}$. Transposing the body of a two-dimensional table and swapping its headers leads to an equivalent table:

**Theorem 2.1 (Transposing).**

$$\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \quad \equiv \quad \begin{array}{c|c} & pv \\ \hline qv & rm^T \end{array}$$

Some theorems that follow involve only one header. By applying transposition, a dual theorem for the other header follows immediately. In these cases we give only one theorem and leave out the dual one. A direct consequence of the commutativity of disjunction is that we can swap rows and, by duality, columns arbitrarily:

**Theorem 2.2 (Swapping Rows and Columns).**

$$\begin{array}{c|c|c} & qv & rv \\ \hline pv & qm & rm \end{array} \quad \equiv \quad \begin{array}{c|c|c} & rv & qv \\ \hline pv & rm & qm \end{array}$$

A table can be split into a disjunction of two tables by separating some rows or some columns. This gives a way of reducing the size of tables if they become too big:

**Theorem 2.3 (Splitting and Joining Tables).**

$$\begin{array}{c|c|c} & qv & rv \\ \hline pv & qm & rm \end{array} \equiv \begin{array}{c|c} & qv \\ \hline pv & qm \end{array} \vee \begin{array}{c|c} & rv \\ \hline pv & rm \end{array}$$

We may want to extend a table by further rows or columns in a way that preserves the meaning of the table, perhaps in order to make some cases explicit. Dually, we may want to contract a table by deleting rows or columns if they are redundant. The following theorem gives the condition for doing so. In its formulation we use the fact that $\equiv$ is associative:

**Theorem 2.4 (Extending and Contracting).**

$$\begin{array}{c|c|c} & qv & rv \\ \hline pv & qm & rm \end{array} \equiv \begin{array}{c|c} & qv \\ \hline pv & qm \end{array} \equiv \begin{array}{c|c} & rv \\ \hline pv & rm \end{array} \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & qm \end{array}$$

*Proof.* Let us write the theorem as $a \equiv b \equiv c \Rightarrow b$. By applying Theorem 2.3, this is equivalent to $b \vee c \equiv b \equiv c \Rightarrow b$, which holds as $b \vee c \equiv b$ is indeed equivalent to $c \Rightarrow b$. □

A two-dimensional table with multiple columns but a single row can be flattened into a one-dimensional table by conjoining the header of the row to each body element. In the general case with multiple rows, we can conjoin each row header to all elements of that row and then take the disjunctions of the columns. Given a matrix $rm$, we write $rm_i$ for the vector forming the $i$-th row. Thus $\vee i \bullet rm_i$ is a vector that is obtained by disjoining the columns of $rm$. In the following theorem we identify vector $rv$ with a matrix with one row:

**Theorem 2.5 (Lifting and Flattening).**

(a) $$\begin{array}{c|c} & qv \\ \hline p & rv \end{array} \equiv \begin{array}{c} qv \\ \hline p \wedge rv \end{array}$$

(b) $$\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \equiv \begin{array}{c} qv \\ \hline \vee i \bullet pv_i \wedge rm_i \end{array}$$

A one-dimensional table can be trivially lifted to a two-dimensional table by adding the row header *true*, i.e. taking $p = true$ in part (a) of above theorem. The above way of flattening a two-dimensional table always preserves one header. Alternatively, we can flatten a table with headers $pv$ and $qv$ into a table with one header consisting of all combinations of $pv_i \wedge qv_j$, as done by Zucker [Zuc96].

The possibility of lifting and flattening tables means that each theorem about a one-dimensional table can be applied to a two-dimensional table and vice versa. We continue to give theorems only for two-dimensional tables: after all, two-dimensional tables have a richer visual structure, which was our starting point. Corresponding theorems for one-dimensional tables arise as simple special cases.

A vector $pv$ is *disjoint* if all its elements are mutually exclusive, $\neg(pv_i \wedge pv_k)$ for all $i$ and $k$ with $i \neq k$. Two vectors $pv$ and $qv$ are *jointly disjoint* if $\neg(pv_i \wedge qv_j \wedge pv_k \wedge qv_l)$ for all $i$, $j$, $k$, $l$ with either $i \neq k$ or $j \neq l$. If $pv$ and $qv$ are jointly disjoint, then the conjunction of any two elements of $pv$ ($qv$) does not need to be false in isolation but only if conjoined with an element of $qv$ ($pv$). Vector $pv$ *covers (at least)* $c$ if one of its elements is true if $c$ is true, $c \Rightarrow \vee i \bullet pv_i$, and *covers exactly* $c$ if $c \equiv \vee i \bullet pv_i$. Vector $pv$ is *total* if it covers *true*. Vector $pv$ *partitions* $c$ if it is disjoint and covers exactly $c$. Finally, a table is in *canonical form* if all its headers are jointly disjoint and each header is total. The table in Fig. 1 is in canonical form.

A table with one of its headers having overlapping elements can be successively transformed into an equivalent table with a disjoint header by replacing pairs of rows or columns with overlapping header elements. Suppose $s, t$ are possibly overlapping predicates:

**Theorem 2.6 (Removing Header Overlap).**

$$\begin{array}{c|c|c|c} & qv & s & t \\ \hline pv & rm & sv & tv \end{array} \equiv \begin{array}{c|c|c|c|c} & qv & s \wedge \neg t & \neg s \wedge t & s \wedge t \\ \hline pv & rm & sv & tv & sv \vee tv \end{array}$$

Repeatedly applying the above rule yields a table with a disjoint upper header; as there is choice in applying the rule, the resulting tabular structure is not uniquely determined. The following rule allows transforming any header into a total one while preserving the meaning of the table:

| | $mode = waiting$ | $mode \neq waiting$ |
|---|---|---|
| $f = floor$ | $reqs' = \{\}$ | $reqs' = reqs$ |
| $f \neq floor$ | $reqs' = \{f\}$ | $reqs' = reqs \cup \{f\}$ |

| | $mode = up$ | $mode = waiting$ | $mode = down$ |
|---|---|---|---|
| $f > floor$ | $mode' = up$ | $mode' = up$ | $mode' = down$ |
| $f = floor$ | $mode' = up$ | $mode' = waiting$ | $mode' = down$ |
| $f < floor$ | $mode' = up$ | $mode' = down$ | $mode' = down$ |

**Fig. 2.** Predicates *requestsReq* (upper table) and *modeReq* (lower table) expressing the requirements how the variables *reqs* and *mode*, respectively, have to change when the button for calling the elevator at floor $f$ is pressed.

**Theorem 2.7 (Making Header Total).**

| | $qv$ |
|---|---|
| $pv$ | $rm$ |

$\equiv$

| | $qv$ | $\neg \vee j \bullet qv_j$ |
|---|---|---|
| $pv$ | $rm$ | $false$ |

Hence, with above two rules any header can be transformed into an equivalent one that partitions *true*. While this gives us the option of transforming a table into an equivalent canonical one, we do not require each well-defined table to be in canonical form. By comparison, Parnas [Par92] and Zucker [Zuc96] consider such tables not to be *proper* (for that reason they use a slightly different terminology) and PVS [ORS97] generates proof obligations to ensure properness.

We consider further means of transforming tables that turn out to be useful intermediate steps when combining larger tables. We can replace table elements while preserving the meaning of the table:

**Theorem 2.8 (Replacing Table Elements).**

(a)

| | $q$ | $qv$ |
|---|---|---|
| $p$ | $r$ | $\cdots$ |
| $pv$ | $\cdots$ | $\cdots$ |

$\equiv$

| | $q$ | $qv$ |
|---|---|---|
| $p$ | $r'$ | $\cdots$ |
| $pv$ | $\cdots$ | $\cdots$ |

$\Leftarrow \ (p \wedge q \Rightarrow (r \equiv r'))$

(b)

| | $q$ | $qv$ |
|---|---|---|
| $pv$ | $rv$ | $\cdots$ |

$\equiv$

| | $q'$ | $qv$ |
|---|---|---|
| $pv$ | $rv$ | $\cdots$ |

$\Leftarrow \ (\wedge i \bullet pv_i \wedge rv_i \Rightarrow (q \equiv q'))$

Replacing a header element is not necessarily going to preserve disjointness and coverage of the header, but may be used to achieve disjointness or a specific coverage. If tables are used to specify operations, we may replace table elements to move all conditions on the inputs to the headers and all conditions on the outputs to the body. Next we give a theorem that allows to split a row or column and to join two rows or columns:

**Theorem 2.9 (Splitting and Joining Rows and Columns).**

| | $q \vee r$ | $\cdots$ |
|---|---|---|
| $pv$ | $sv$ | $\cdots$ |

$\equiv$

| | $q$ | $r$ | $\cdots$ |
|---|---|---|---|
| $pv$ | $sv$ | $sv$ | $\cdots$ |

When using tables for specification, we *separate concerns* by using a number of tables describing related issues. For further analysing and for combining these tables we typically need to rewrite them in order to make their headers identical.

**Example 2.1 (Harmonising Tables).** Consider harmonising the predicates *requestsReq* and *modeReq* in Fig. 2. They specify separately the requirements on the transitions of variables *reqs* and *mode*. For conjoining these requirements we need—as shown later—to make their headers identical. We assume that *up*, *down*, *waiting* are the only values of *mode*. For harmonising the left headers we decide to split the header element $f \neq floor$ of *requestsReq* into $f > floor$ and $f < floor$ according to Theorem 2.9. For harmonising the upper headers we decide to join the $mode = up$ and $mode = down$ columns of *modeReq* according to Theorem 2.9, after first replacing $mode' = up$ and $mode' = down$ with $mode' = mode$ according to Theorem 2.8 (a). The result is shown in Fig. 3.

We now describe a general procedure for harmonising tables. Suppose we have two tables given, in each of which we have singled out one header element that we want to harmonise. Let these be called $q$ and $t$, i.e. let

| | $mode = waiting$ | $mode \neq waiting$ |
|---|---|---|
| $f > floor$ | $reqs' = \{f\}$ | $reqs' = reqs \cup \{f\}$ |
| $f = floor$ | $reqs' = \{\}$ | $reqs' = reqs$ |
| $f < floor$ | $reqs' = \{f\}$ | $reqs' = reqs \cup \{f\}$ |

| | $mode = waiting$ | $mode \neq waiting$ |
|---|---|---|
| $f > floor$ | $mode' = up$ | $mode' = mode$ |
| $f = floor$ | $mode' = waiting$ | $mode' = mode$ |
| $f < floor$ | $mode' = down$ | $mode' = mode$ |

**Fig. 3.** The predicates *requestsReq* and *modeReq* of Fig. 2 harmonised to have identical headers.

the two tables be of the form:

| | $q$ | $qv$ |
|---|---|---|
| $pv$ | $rv$ | $rm$ |

and

| | $t$ | $tv$ |
|---|---|---|
| $sv$ | $uv$ | $um$ |

If $q \equiv t$, then these two header elements are already identical and we can continue harmonising the remaining header elements. Suppose now that $q \Rightarrow t$. Then with Theorem 2.9 we transform the right table such that we get:

| | $q$ | $qv$ |
|---|---|---|
| $pv$ | $rv$ | $rm$ |

and

| | $q$ | $\neg q \wedge t$ | $tv$ |
|---|---|---|---|
| $sv$ | $uv$ | $uv$ | $um$ |

Thus we arrive at two tables that have one header element in common and can continue harmonising the remaining header elements. For example, we can apply this theorem to the tables in Fig. 2 by observing that $f > floor \Rightarrow f \neq floor$ and hence split $f \neq floor$ into $f > floor$ and $\neg(f > floor) \wedge f \neq floor$, arriving at the same table as in Fig. 3. Now let $q$ and $t$ be arbitrary predicates. By applying Theorem 2.9 to both tables we get:

| | $q \wedge t$ | $q \wedge \neg t$ | $qv$ |
|---|---|---|---|
| $pv$ | $rv$ | $rv$ | $rm$ |

and

| | $q \wedge t$ | $\neg q \wedge t$ | $tv$ |
|---|---|---|---|
| $sv$ | $uv$ | $uv$ | $um$ |

Repeated application of this step results in two tables with their header elements either pairwise equal or pairwise disjoint. If the header elements are disjoint, i.e. $\neg(q \wedge t)$ holds, then we trivially harmonise those by adding the header element of the other table and making the new body elements false, i.e. we arrive at:

| | $q$ | $t$ | $qv$ |
|---|---|---|---|
| $pv$ | $rv$ | $false$ | $rm$ |

and

| | $q$ | $t$ | $tv$ |
|---|---|---|---|
| $sv$ | $false$ | $uv$ | $um$ |

In the next section we give theorems about operation on tables that require tables to have identical headers.

## 3. Operations on Tabular Predicates

We give some basic theorems about common boolean operators applied to tables. The negation of a table can be shown to hold by establishing that each body element is false. A negation in front of a table can be distributed into its body under certain conditions:

**Theorem 3.1 (Table Negation).**

(a) $\neg$ (table $pv$ / $rm$ | $qv$) $\equiv$ $\wedge\, i, j \bullet pv_i \wedge qv_j \Rightarrow \neg rm_{i,j}$

(b) $\neg$ (table $pv$ / $rm$ | $qv$) $\Rightarrow$ table $pv$ / $\neg rm$ | $qv$ — if $pv$, $qv$ are total

(c) $\neg$ (table $pv$ / $rm$ | $qv$) $\Leftarrow$ table $pv$ / $\neg rm$ | $qv$ — if $pv$, $qv$ are jointly disjoint

(d) $\neg$ (table $pv$ / $rm$ | $qv$) $\equiv$ table $pv$ / $\neg rm$ | $qv$ — if $pv$, $qv$ are total and jointly disjoint

*Proof.* Part (a) follows immediately from the definitions and logic. For (b) we get by applying the definitions:

$$\begin{array}{ll} & \neg(\vee i,j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \Rightarrow (\vee i,j \bullet pv_i \wedge qv_j \wedge \neg rm_{i,j}) \\ \equiv & (\vee i,j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \vee (\vee i,j \bullet pv_i \wedge qv_j \wedge \neg rm_{i,j}) \\ \equiv & \vee i,j \bullet pv_i \wedge qv_j \end{array}$$

Hence the proof shows that totality of $pv$ and $qv$ is both necessary and sufficient. For (c) we get by applying the definitions:

$$\begin{array}{ll} & \neg(\vee i,j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \Leftarrow (\vee i,j \bullet pv_i \wedge qv_j \wedge \neg rm_{i,j}) \\ \equiv & \neg((\wedge i,j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \wedge (\wedge i,j \bullet pv_i \wedge qv_j \wedge \neg rm_{i,j})) \\ \equiv & \neg(\wedge i,j,k,l \bullet pv_i \wedge qv_j \wedge rm_{i,j} \wedge pv_k \wedge qv_l \wedge \neg rm_{k,l}) \end{array}$$

We proceed by analysing two cases: if $i,j = k,l$ then $rm_{i,j} \wedge \neg rm_{k,l}$ is false and therefore the whole predicate true. Now if $i,j \neq k,l$ then $pv_i \wedge qv_j \wedge pv_k \wedge qv_l$ is false due to the disjointness assumption, hence the whole predicate is true as well, which concludes the proof of (c). Finally, (d) follows from both (b) and (c). □

Two tables with equal headers can be conjoined by taking the conjunction of the corresponding body elements under certain conditions:

**Theorem 3.2 (Table Conjunction).**

(a)
$$\begin{array}{|c|c|}\hline & qv \\ \hline pv & rm \\ \hline \end{array} \wedge \begin{array}{|c|c|}\hline & qv \\ \hline pv & sm \\ \hline \end{array} \Leftarrow \begin{array}{|c|c|}\hline & qv \\ \hline pv & rm \wedge sm \\ \hline \end{array}$$

(b)
$$\begin{array}{|c|c|}\hline & qv \\ \hline pv & rm \\ \hline \end{array} \wedge \begin{array}{|c|c|}\hline & qv \\ \hline pv & sm \\ \hline \end{array} \equiv \begin{array}{|c|c|}\hline & qv \\ \hline pv & rm \wedge sm \\ \hline \end{array} \qquad \text{if } pv,\ qv \text{ are jointly disjoint}$$

In the proof we make use of following lemma:

**Lemma 3.1.**

(a) $(\vee k \bullet p_k) \wedge (\vee k \bullet q_k) \Leftarrow \vee k \bullet p_k \wedge q_k$

(b) $(\vee k \bullet p_k) \wedge (\vee k \bullet q_k) \equiv \vee k \bullet p_k \wedge q_k \qquad \text{if } k \neq l \Rightarrow \neg(p_k \wedge q_l)$

*Proof of Theorem 3.2* We introduce a new index $k$ for the pair $i,j$ and rewrite (a) as $(\vee k \bullet pq_k \wedge rm_k) \wedge (\vee k \bullet pq_k \wedge sm_k) \Leftarrow \vee k \bullet pq_k \wedge rm_k \wedge sm_k$ with $pq_{i,j} \equiv pv_i \wedge qv_j$. This allows us to apply Lemma 3.1 (a) in the first step of following proof:

$$\begin{array}{ll} & (\vee k \bullet pq_k \wedge rm_k) \wedge (\vee k \bullet pq_k \wedge sm_k) \\ \Leftarrow & \vee k \bullet pq_k \wedge rm_k \wedge pq_k \wedge sm_k \\ \equiv & \vee k \bullet pq_k \wedge rm_k \wedge sm_k \end{array}$$

We prove (b) in a similar way by introducing $k$ for the pair $i,j$ and rewriting it as $(\vee k \bullet pq_k \wedge rm_k) \wedge (\vee k \bullet pq_k \wedge sm_k) \equiv \vee k \bullet pq_k \wedge rm_k \wedge sm_k$ with $pq_{i,j} \equiv pv_i \wedge qv_j$. As $pv, qv$ are jointly disjoint we have $k \neq l \Rightarrow \neg(pq_k \wedge pq_l)$, allowing us to apply Lemma 3.1 (b) in the first step of following proof:

$$\begin{array}{ll} & (\vee k \bullet pq_k \wedge rm_k) \wedge (\vee k \bullet pq_k \wedge sm_k) \\ \equiv & \vee k \bullet pq_k \wedge rm_k \wedge pq_k \wedge sm_k \\ \equiv & \vee k \bullet pq_k \wedge rm_k \wedge sm_k \end{array}$$

□

Two tables with equal headers can be disjoined by taking the disjunction of the corresponding body elements:

**Theorem 3.3 (Table Disjunction).**

$$\begin{array}{|c|c|}\hline & qv \\ \hline pv & rm \\ \hline \end{array} \vee \begin{array}{|c|c|}\hline & qv \\ \hline pv & sm \\ \hline \end{array} \equiv \begin{array}{|c|c|}\hline & qv \\ \hline pv & rm \vee sm \\ \hline \end{array}$$

We give two theorems about implications $s \Rightarrow t$. In the first one either $s$ or $t$ is a tabular predicate and the other one a simple predicate. This can be used for showing that a tabular specification $s$ has a certain property $t$ by $s \Rightarrow t$. Alternatively, given some specification $s$ we can show that it has property $t$ expressed in tabular form. In the second theorem both $s$ and $t$ are tabular predicates.

**Theorem 3.4 (Predicate-Table Implication).**

(a) $\left(\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \Rightarrow s\right) \equiv (\wedge i,j \bullet pv_i \wedge qv_j \wedge rm_{i,j} \Rightarrow s)$

(b) $\left(s \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & rm \end{array}\right) \Rightarrow (\wedge i,j \bullet s \wedge pv_i \wedge qv_j \Rightarrow rm_{i,j})$ if $pv, qv$ are jointly disjoint

(c) $\left(s \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & rm \end{array}\right) \Leftarrow (\wedge i,j \bullet s \wedge pv_i \wedge qv_j \Rightarrow rm_{i,j})$ if $pv$ covers $s$ and $qv$ covers $s$

(d) $\left(s \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & rm \end{array}\right) \equiv (\wedge i,j \bullet s \wedge pv_i \wedge qv_j \Rightarrow rm_{i,j})$ if $pv$ covers $s$ and $qv$ covers $s$ and $pv$, $qv$ are jointly disjoint

(e) $\left(s \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & rm \end{array}\right) \equiv \begin{array}{c|c} & s \Rightarrow qv \\ \hline s \Rightarrow pv & s \Rightarrow rm \end{array}$

(f) $\left(s \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & rm \end{array}\right) \equiv \begin{array}{c|c} & qv \\ \hline pv & s \Rightarrow rm \end{array}$ if $pv$, $qv$ are total and jointly disjoint

Parts (a) to (d) allow the implication to be established by considering each case that the tables describes in turn, thus decomposing a possibly large proof into a number of smaller ones. In (b) to (c) where the table is on the right hand side of the implication, we need side conditions about coverage or disjointness. By contrast, (e) does not have side conditions and preserves the structure of the table. However, in this case the implication distributes into the table body as well as into the headers. In (f) the implication distributes into the table body only, but now under the side condition of totality and disjointness.

**Lemma 3.2.**

(a) $(\wedge k \bullet p_k \Rightarrow q_k) \Rightarrow (\vee k \bullet p_k \wedge q_k)$ if $\vee k \bullet p_k$

(b) $(\wedge k \bullet p_k \Rightarrow q_k) \Leftarrow (\vee k \bullet p_k \wedge q_k)$ if $k \neq l \Rightarrow \neg(p_k \wedge p_l)$

(c) $(\wedge k \bullet p_k \Rightarrow q_k) \equiv (\vee k \bullet p_k \wedge q_k)$ if $\vee k \bullet p_k$ and $k \neq l \Rightarrow \neg(p_k \wedge p_l)$

*Proof of Theorem 3.4* The proof of (a) is straightforward. For the proof of (b) we make use of Lemma 3.2 (b) with $k$ replaced by the pair $i,j$, as $pv, qv$ are jointly disjoint:

$$\begin{array}{lcl} L.H.S. & \equiv & s \Rightarrow (\vee i,j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \\ & \Rightarrow & s \Rightarrow (\wedge i,j \bullet pv_i \wedge qv_j \Rightarrow rm_{i,j}) \\ & \equiv & \wedge i,j \bullet s \Rightarrow (pv_i \wedge qv_j \Rightarrow rm_{i,j}) \\ & \equiv & R.H.S. \end{array}$$

For the proof of (c) we make use of Lemma 3.2 (a) with $k$ replaced by the pair $i,j$, as $s \Rightarrow \vee i \bullet pv_i$ and $s \Rightarrow \vee j \bullet qv_j$:

$$\begin{array}{lcl} L.H.S. & \equiv & s \Rightarrow (\vee i,j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \\ & \Leftarrow & s \Rightarrow (\wedge i,j \bullet pv_i \wedge qv_j \Rightarrow rm_{i,j}) \\ & \equiv & \wedge i,j \bullet s \Rightarrow (pv_i \wedge qv_j \Rightarrow rm_{i,j}) \\ & \equiv & R.H.S. \end{array}$$

Part (d) follows from both (b) and (c). For (e) we have:

$$\begin{array}{lcl} L.H.S. & \equiv & s \Rightarrow (\vee i,j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \\ & \equiv & \vee i,j \bullet s \Rightarrow pv_i \wedge qv_j \wedge rm_{i,j} \\ & \equiv & \vee i,j \bullet (s \Rightarrow pv_i) \wedge (s \Rightarrow qv_j) \wedge (s \Rightarrow rm_{i,j}) \\ & \equiv & R.H.S. \end{array}$$

Part (f) follows from (d) and Lemma 3.2 (c). □

We continue with a theorem that combines two tables with identical headers.

**Theorem 3.5 (Table Implication).**

(a) $\left(\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & sm \end{array}\right) \Leftarrow \ \wedge i, j \bullet pv_i \wedge qv_j \wedge rm_{i,j} \Rightarrow sm_{i,j}$

(b) $\left(\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & sm \end{array}\right) \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & rm \Rightarrow sm \end{array}$ if $pv$, $qv$ are total

(c) $\left(\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & sm \end{array}\right) \Leftarrow \begin{array}{c|c} & qv \\ \hline pv & rm \Rightarrow sm \end{array}$ if $pv$, $qv$ are jointly disjoint

(d) $\left(\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \Rightarrow \begin{array}{c|c} & qv \\ \hline pv & sm \end{array}\right) \equiv \begin{array}{c|c} & qv \\ \hline pv & rm \Rightarrow sm \end{array}$ if $pv$, $qv$ are total and jointly disjoint

Part (a) allows the implication to be established by considering each case that the tables describes in turn, thus decomposing a possibly large proof into a number of smaller ones. Parts (b) to (c) preserve the structure of the table, but require either totality or disjointness; we remind that totality can always be satisfied by first applying Theorem 2.7.

*Proof.* For (a) we have:

$$\begin{array}{rcl} L.H.S. & \equiv & (\vee i, j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \Rightarrow (\vee i, j \bullet pv_i \wedge qv_j \wedge sm_{i,j}) \\ & \equiv & \wedge i, j \bullet pv_i \wedge qv_j \wedge rm_{i,j} \Rightarrow (\vee i, j \bullet pv_i \wedge qv_j \wedge sm_{i,j}) \\ & \Leftarrow & \wedge i, j \bullet pv_i \wedge qv_j \wedge rm_{i,j} \Rightarrow pv_i \wedge qv_j \wedge sm_{i,j} \\ & \equiv & R.H.S. \end{array}$$

Parts (b) to (d) follow immediately from Theorem 3.1 (b) to (d) for negations and Theorem 3.3 for disjunctions by rewriting the leftmost implication as a disjunction with a negation. □

We conclude with a theorem about the equivalence of tables with identical headers. Part (a) of the theorem allows equivalence to be established by considering each case the tables describe in turn, while parts (b) and (c) preserve the structure of the tables:

**Theorem 3.6 (Table Equivalence).**

(a) $\left(\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \equiv \begin{array}{c|c} & qv \\ \hline pv & sm \end{array}\right) \Leftarrow \ \wedge i, j \bullet pv_i \wedge qv_j \Rightarrow (rm_{i,j} \equiv sm_{i,j})$

(b) $\left(\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \equiv \begin{array}{c|c} & qv \\ \hline pv & sm \end{array}\right) \Leftarrow \begin{array}{c|c} & qv \\ \hline pv & rm \equiv sm \end{array}$ if $pv$, $qv$ are jointly disjoint

(c) $\left(\begin{array}{c|c} & qv \\ \hline pv & rm \end{array} \equiv \begin{array}{c|c} & qv \\ \hline pv & sm \end{array}\right) \equiv \begin{array}{c|c} & qv \\ \hline pv & rm \equiv sm \end{array}$ if $pv$, $qv$ are total and jointly disjoint

*Proof.* For (a) we have:

$$\begin{array}{rcl} L.H.S. & \equiv & (\vee i, j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \equiv (\vee i, j \bullet pv_i \wedge qv_j \wedge sm_{i,j}) \\ & \Leftarrow & \wedge i, j \bullet pv_i \wedge qv_j \wedge rm_{i,j} \equiv pv_i \wedge qv_j \wedge sm_{i,j} \\ & \equiv & \wedge i, j \bullet pv_i \wedge qv_j \Rightarrow (rm_{i,j} \equiv sm_{i,j}) \\ & \equiv & R.H.S. \end{array}$$

Parts (b) and (c) follow from applying Theorem 3.5 (c) and (d) for implication in both directions and then applying Theorem 3.2 (a) and (b), respectively. □

## 4. Tabular Relations

A relation between elements of types $X$ and $Y$ is a function of type $X \to Y \to Bool$. We define the constant relations $\bot$ (empty relation), $\top$ (universal relation), $Id$ (identity relation), and for relations $P$ and $Q$ the operations $\overline{P}$ (complement), $P^{-1}$ (inverse), $P \cap Q$ (intersection), $P \cup Q$ (union), $P \circ Q$ (relational composition)

as well as the predicates $P \subseteq Q$ and $P \supseteq Q$ (inclusion):

$$\begin{array}{lcllcl} \bot\, x\, y & \equiv & \mathit{false} & (P \cup Q)\, x\, y & \equiv & P\, x\, y \vee Q\, x\, y \\ \top\, x\, y & \equiv & \mathit{true} & (P \cap Q)\, x\, y & \equiv & P\, x\, y \wedge Q\, x\, y \\ \mathit{Id}\, x\, y & \equiv & x = y & (P \circ Q)\, x\, y & \equiv & (\exists z \bullet P\, x\, z \wedge Q\, z\, y) \\ \overline{P}\, x\, y & \equiv & \neg P\, x\, y & (P \subseteq Q) & \equiv & (\forall x, y \bullet P\, x\, y \Rightarrow Q\, x\, y) \\ P^{-1}\, x\, y & \equiv & P\, y\, x & (P \supseteq Q) & \equiv & (\forall x, y \bullet P\, x\, y \Leftarrow Q\, x\, y) \end{array}$$

The above defines $\bot$ and $\top$ to be polymorphic relations on two arbitrary types and $\mathit{Id}$ to be a polymorphic relation between elements of the same type. A relation $P$ is *functional* if $P^{-1} \circ P \subseteq \mathit{Id}$ and *injective* if $P \circ P^{-1} \subseteq \mathit{Id}$. Relation $P$ is called a *condition* if $P \circ \top = P$. The *domain* $\Delta\, P$ of a relation $P$ is defined by $\Delta\, P = P \circ \top$. A relation $P$ is *total* if $\Delta\, P = \top$, or equivalently $\mathit{Id} \subseteq P \circ P^{-1}$. Relation $P$ is *surjective* if $\Delta\, P^{-1} = \top$, or equivalently $\mathit{Id} \subseteq P^{-1} \circ P$. We make use of generalised union $\cup i \in I \bullet P_i$ and generalised intersection $\cap i \in I \bullet P_i$, for arbitrary index set $I$. We use the following facts about relations:

**Lemma 4.1.** Let $P, Q, P_i, Q_i$ be relations and $C$ a condition:

$$\begin{array}{lll} \text{(a)} & P \circ (\cup i \in I \bullet Q_i) = \cup\, i \in I \bullet P \circ Q_i & \\ \text{(b)} & (\cup i \in I \bullet P_i) \circ Q = \cup\, i \in I \bullet P_i \circ Q & \\ \text{(c)} & P \circ (\cap i \in I \bullet Q_i) \subseteq \cap\, i \in I \bullet P \circ Q_i & \\ \text{(d)} & (\cap i \in I \bullet P_i) \circ Q \subseteq \cap\, i \in I \bullet P_i \circ Q & \\ \text{(e)} & P \circ (\cap i \in I \bullet Q_i) = \cap\, i \in I \bullet P \circ Q_i & \text{if } P \text{ is functional} \\ \text{(f)} & (\cap i \in I \bullet P_i) \circ Q = \cap\, i \in I \bullet P_i \circ Q & \text{if } Q \text{ is injective} \\ \text{(g)} & (C \cap P) \circ Q = C \cap (P \circ Q) & \end{array}$$

Tabular relations are defined in analogy to tabular predicates using generalised intersection and union. Let $PV$ and $QV$ be vectors of relations and let $RM$ be a matrix of relations:

$$\begin{array}{c|c} & QV \\ \hline PV & RM \end{array} \equiv \cup i, j \bullet PV_i \cap QV_i \cap RM_{i,j}$$

All operations on relations are pointwise extended to operations on vectors and matrices. On occasion we identify a relation $P$ with a vector or a matrix with all elements being $P$. For example, this allows us to write $P \circ PV$, with the meaning of $(P \circ PV)_i = P \circ PV_i$. There is a direct relationship between tabular predicates and tabular relations. Let $pv$ and $qv$ be vectors of predicates, let $rm$ be a matrix of predicates, let $PV$ and $QV$ be vectors of relations, and let $RM$ be a matrix of relations. If

$$PV_i\, x\, y \equiv pv_i\ , \qquad QV_j\, x\, y \equiv qv_j\ , \qquad RM_{i,j}\, x\, y \equiv rm_{i,j}$$

then the following two definitions of relation $S$ are equivalent:

$$S = \begin{array}{c|c} & QV \\ \hline PV & RM \end{array}\ , \qquad S\, x\, y \equiv \begin{array}{c|c} & qv \\ \hline pv & rm \end{array}$$

This relationship between tabular predicates and tabular relations allows us to switch between them as convenient. This also allows us to lift all theorems on tabular predicates to tabular relations as needed. In particular the notions of disjointness and coverage carry over to relations.

**Standard and Inverted Tables.** We define *standard* and *inverted* tabular relations following Parnas [Par92]. A tabular relation is called *standard* if all its headers are conditions. If we take the table in Fig. 1 for defining the relation *ButtonPressed* $(f)$ over the variables *mode* and *reqs*, then that table is standard. The headers are made up of expressions over the initial state only and the body is made up of expressions relating the initial and final state. We think of the *flow* in the table going from both headers to the body [JK01]. A tabular relation is called *inverted* if the body and all but one header are conditions. If we take the table in Fig. 4 for defining the relation *ButtonPressedMode* $(f)$, then that table is inverted. The left header and the body are made up of expressions over the initial state only and the upper header is made up of expressions over the final state. We think of the flow in the table going from the left header to the body and from there to the upper header. A typical use of inverted tables is if there are a large number of different conditions for only a few possibilities for the outcome. While other kinds of tables are conceivable, we consider here only tables that are either standard or inverted.

| | $mode' = up$ | $mode' = down$ | $mode' = waiting$ |
|---|---|---|---|
| $f > floor$ | $true$ | $false$ | $false$ |
| $f = floor$ | $mode = up$ | $mode = down$ | $mode = waiting$ |
| $f < floor$ | $false$ | $true$ | $false$ |

**Fig. 4.** Example of an inverted table

**Consistency and Completeness.** Tabular representations can help in formulating and analysing the *consistency* and *completeness* of a specification given by a relation. In its simplest form, consistency is defined as the relation being deterministic and completeness as the relation being total. For tabular specifications, consistency and completeness can be defined in a more liberal way through disjointness and coverage. We argue that this typically better suits the problem at hand with some small examples. Consider relation $P\,x\,x'$ defined by:

| $x < 0$ | $x = 0$ | $x > 0$ |
|---|---|---|
| $x' = -x$ | $false$ | $x' = x$ |

If we define completeness to mean that all headers are total, i.e. cover *true*, then above table is complete, even though $P$ as a relation is not total. While $P$ could be equivalently defined with the middle column left out, the inclusion of it indicates that partiality in the case of $x = 0$ is intentional; this kind of redundancy allows checks to be carried out. Consider relation $Q\,x\,x'$ defined by:

| $x < 0$ | $x = 0$ | $x > 0$ |
|---|---|---|
| $x' = -x$ | $true$ | $x' = x$ |

If we define consistency to mean that all headers are disjoint, then above table is consistent, even though $Q$ as a relation is nondeterministic. By confining nondeterminism to the body of tables the source of nondeterminism is more local: unintentional nondeterminism through overlaps in header conditions of (possibly large) tables cannot occur. We point out that an overlap in a header does not necessarily imply that the relation is nondeterministic. Consider relation $R\,x\,x'$ defined by:

| $x \leq 0$ | $x \geq 0$ |
|---|---|
| $x' = -x$ | $x' = x$ |

In this case we could argue whether this is a good style of writing a specification or not. Heitmeyer et al. [HJL96] report on detecting numerous errors in specifications of embedded systems through analysing tables for disjointness and coverage.

**Operations on Tabular Relations.** The domain of a tabular relation in standard form can be determined by taking the domain of each body element. Assume $BV$ and $CV$ are vectors of conditions:

**Theorem 4.1 (Table Domain).**

$$\Delta \left( \begin{array}{c|c} & CV \\ \hline BV & PM \end{array} \right) \;=\; \begin{array}{c|c} & CV \\ \hline BV & \Delta\, PM \end{array}$$

*Proof.* This follows from Lemma 4.1 (b) and (g), and the definition of $\Delta$. □

We study how relational composition distributes into tables. If the first operand of a relational composition is a tabular relation in standard form, we can distribute the second operand into the table body. If the second operand of a relational composition is a tabular relation, we get only inclusion, even if the relation is in standard form. However, we get equality if the first operand is functional:

**Theorem 4.2 (Table Composition).**

(a) $$\begin{array}{c|c} & CV \\ \hline BV & PM \end{array} \;\circ\; Q \;=\; \begin{array}{c|c} & CV \\ \hline BV & PM \circ Q \end{array}$$

(b) $$S \;\circ\; \begin{array}{c|c} & QV \\ \hline PV & RM \end{array} \;\subseteq\; \begin{array}{c|c} & S \circ QV \\ \hline S \circ PV & S \circ RM \end{array}$$

(c) $S \circ \begin{array}{c|c} & QV \\ \hline PV & RM \end{array} = \begin{array}{c|c} & S \circ QV \\ \hline S \circ PV & S \circ RM \end{array}$ if $S$ is functional

*Proof.* Part (a) follows from Lemma 4.1 (b) and (g). For (b) we make use of Lemma 4.1 (a) and (c):

$$\begin{array}{lcl} L.H.S. & = & S \circ (\cup i, j \bullet PV_i \cap QV_j \cap RM_{i,j}) \\ & = & \cup i, j \bullet S \circ (PV_i \cap QV_j \cap RM_{i,j}) \\ & \subseteq & \cup i, j \bullet (S \circ PV_i) \cap (S \circ QV_j) \cap (S \circ RM_{i,j}) \\ & = & R.H.S. \end{array}$$

The proof of (c) is similar, except that we use Lemma 4.1 (a) and (e). □

## 5. Tabular Verification

We use relations to model nondeterministic programs. A relation of type $X \to Y \to \mathit{Bool}$ models a program with initial state space $X$ and final state space $Y$. The domain $\Delta\, P$ of a program $P$ is interpreted either as the *enabledness domain* (or *guard*) of $P$ or as the *termination domain* (or *precondition*) of $P$. The weakest precondition $[P]\, C$ of program $P$ to establish postcondition $C$ characterises those initial states in which $P$ is never going to lead to a state outside $C$:

$$[P]\, C \;=\; \overline{P \circ \overline{C}}$$

If $\Delta\, P$ is interpreted as the enabledness domain or program $P$, then $[P]\, C$ characterises those initial states in which either $P$ is not enabled or $P$ is enabled and leads to a state in $C$. If $\Delta\, P$ is interpreted as the termination domain of program $P$, then $[P]\, C$ characterises those initial states in which either $P$ does not terminate or $P$ terminates and leads to a state in $C$. In this case we would refer to $[P]\, C$ as the *weakest liberal precondition*. Leaving both interpretations open, we uniformly refer to $[P]\, C$ as the weakest precondition for $P$ to establish $C$.

The weakest precondition can equivalently be defined in terms of predicates. We assume that the state consists of a vector $xv$ of variables and that the initial and final state space are products of the same type:

**Theorem 5.1 (Weakest Precondition).**

$$[P]\, C\, xv\, xv' \;\equiv\; \forall xv' \bullet P\, xv\, xv' \Rightarrow C\, xv'\, xv'$$

*Proof.* We observe that if $C$ is a condition, the $C\, xv\, yv \equiv C\, xv\, zv$ for arbitrary, not necessarily distinct $xv, yv, zv$. We also note that if $C$ is a condition, the $[P]\, C$ is also a condition, allowing us to conclude that $[P]\, C\, xv\, xv' = [P]\, C\, xv\, xv''$. With this the theorem follows from the definition of $[P]\, C$. □

A typical use of weakest preconditions is for checking invariance properties: an operation $P$ *establishes* condition $C$ if $[P]\, C = \top$ and $P$ *preserves* $C$ if $C \subseteq [P]\, C$. Consequently we give theorems for deducing that a weakest precondition—if expressed as a predicate—is either universally true or is weaker than a given precondition. We make use of the following facts about weakest preconditions. Assume $P, P_i$ are relations, for an arbitrary index set $I$, and $B, C$ are conditions:

**Lemma 5.1.**

(a) $[\cup i \in I \bullet P_i]\, C \;=\; \cap\, i \in I \bullet [P_i]\, C$

(b) $[B \cap P]\, C \;=\; \overline{B} \cup [P]\, C$

We give some theorems for determining weakest preconditions of operations in tabular form. For a matrix $PV$ and a condition $C$ let $[PM]\, C$ stand for $PM$ with the weakest precondition applied to each element, formally $([PM]\, C)_{i,j} = [PM_{i,j}]\, C$:

**Theorem 5.2 (Tabular Weakest Precondition).**

(a) $\begin{array}{c|c} & CV \\ \hline BV & [PM]\, C \end{array} \subseteq \left[\begin{array}{c|c} & CV \\ \hline BV & PM \end{array}\right] C$ if $BV$, $CV$ are total

(b) $\begin{array}{c|c} & CV \\ \hline BV & [PM]\, C \end{array} \supseteq \left[\begin{array}{c|c} & CV \\ \hline BV & PM \end{array}\right] C$ if $BV$, $CV$ are jointly disjoint

(c) $\dfrac{\begin{array}{c|c} & CV \\ \hline BV & [PM]\,C \end{array}}{} = \left[\dfrac{\begin{array}{c|c} & CV \\ \hline BV & PM \end{array}}{}\right] C$ if $BV$, $CV$ are total and jointly disjoint

*Proof.* For (a) we make use of Theorems 4.2 (a) and of 3.1 (b) lifted to relations:

$$\begin{array}{lcl} L.H.S. & = & \overline{\dfrac{\begin{array}{c|c} & CV \\ \hline BV & PM \end{array}}{} \circ \overline{C}} \\ & = & \overline{\dfrac{\begin{array}{c|c} & CV \\ \hline BV & PM \circ \overline{C} \end{array}}{}} \\ & \subseteq & \dfrac{\begin{array}{c|c} & CV \\ \hline BV & \overline{PM \circ \overline{C}} \end{array}}{} \\ & = & R.H.S. \end{array}$$

The proof of (b) is analogous, except that we use Theorem 3.1 (c) instead. Finally, (c) follows from both (a) and (b). □

We note that typically only (c) is useful as (a) results in a precondition that may be too restrictive and (b) may not result in a precondition for the given postcondition at all. While (c) allows the precondition to be determined by considering each case in the body of the program in turn, it does have the side conditions of totality and disjointness. We give an alternative theorem that does not have these side conditions but allows only inclusion to be shown, although it gives a necessary and sufficient condition for it. Thus it can always be used to verify that a tabular relation under a given precondition establishes a given postcondition:

**Theorem 5.3 (Tabular Verification).**

$$B \subseteq \left[\dfrac{\begin{array}{c|c} & CV \\ \hline BV & PM \end{array}}{}\right] C \quad \equiv \quad \wedge\, i,j \bullet B \cap BV_i \cap CV_j \subseteq [PM_{i,j}]\, C$$

*Proof.* We make use of Lemma 5.1:

$$\begin{array}{lcl} L.H.S. & \equiv & B \subseteq [\cup i,j \bullet BV_i \cap CV_j \cap PM_{i,j}]\, C \\ & \equiv & B \subseteq \cap i,j \bullet [BV_i \cap CV_j \cap PM_{i,j}]\, C \\ & \equiv & \wedge i,j \bullet B \subseteq [BV_i \cap CV_j \cap PM_{i,j}]\, C \\ & \equiv & \wedge i,j \bullet B \subseteq \overline{BV_i \cap CV_j} \cup [PM_{i,j}]\, C \\ & \equiv & \wedge i,j \bullet B \cap BV_i \cap CV_j \subseteq [PM_{i,j}]\, C \\ & \equiv & R.H.S. \end{array}$$

□

For the case that the table is given by a tabular predicate and the postcondition by a predicate, we can give the analogue of Theorem 5.2. For brevity, we give only the analogue of Theorem 5.2 (c). We assume that the state consists of a vector $xv$ of variables. If $pm$ is a matrix of predicates, we write $\forall x \bullet pm$ for every matrix element universally quantified over $x$, formally $(\forall x \bullet pm)_{i,j} \equiv (\forall x \bullet pm_{i,j})$. Let $f[xv \backslash ev]$ stand for expression $f$ with each variable in $xv$ simultaneously substituted by the corresponding expressions in $ev$.

**Theorem 5.4 (Weakest Precondition with Predicates).** If standard relation $P$ and condition $C$ are given by

$$P\ xv\ xv' \ \equiv\ \dfrac{\begin{array}{c|c} & cv \\ \hline bv & pm \end{array}}{}\,, \qquad C\ xv\ xv' \ \equiv\ c$$

and if $bc$, $cv$ are total and jointly disjoint we have:

$$[P]\ C\ xv\ xv' \ \equiv\ \dfrac{\begin{array}{c|c} & cv \\ \hline bv & \forall xv' \bullet pm \Rightarrow c[xv \backslash xv'] \end{array}}{}$$

*Proof.* The theorem follows from Theorems 5.2 (c) and 5.1. □

Next we give a theorem that does not have the side conditions of totality and disjointness of the headers and does not even require the table to be in standard form. Hence it can also be applied to inverted tables:

| | $mode = waiting$ | $mode \neq waiting$ |
|---|---|---|
| $f > floor$ | $floor \notin \{f\}$ | $floor \notin reqs \cup \{f\}$ |
| $f = floor$ | $floor \notin \{\}$ | $floor \notin reqs$ |
| $f < floor$ | $floor \notin \{f\}$ | $floor \notin reqs \cup \{f\}$ |

**Fig. 5.** The condition $[ButtonPressed\,(f)]\ FloorNotInReqs$ expressed by a tabular predicate over the variables $mode, reqs$.

**Theorem 5.5 (Tabular Verification with Predicates).** If conditions $B, C$ and relation $P$ are given by

$$B\ xv\ xv' \ \equiv\ b\ , \qquad P\ xv\ xv' \ \equiv\ \begin{array}{c|c} & qv \\ \hline pv & rm \end{array}\ , \qquad C\ xv\ xv' \ \equiv\ c$$

we have:

$$B \subseteq [P]\ C \ \equiv\ \ \wedge\, i, j \bullet b \wedge pv_i \wedge qv_j \wedge rm_{i,j} \Rightarrow c[xv \backslash xv']$$

*Proof.* We make use of Theorem 5.1:

$$\begin{array}{lll} L.H.S. & \equiv & B\ xv\ xv' \Rightarrow ([P]\ C)\ xv\ xv' \\ & \equiv & b \Rightarrow (\forall xv' \bullet P\ xv\ xv' \Rightarrow C\ xv'\ xv') \\ & \equiv & b \Rightarrow (\forall xv' \bullet P\ xv\ xv' \Rightarrow c[xv \backslash xv']) \\ & \equiv & b \Rightarrow ((\vee i, j \bullet pv_i \wedge qv_j \wedge rm_{i,j}) \Rightarrow c[xv \backslash xv']) \\ & \equiv & \wedge i, j \bullet b \Rightarrow (pv_i \wedge qv_j \wedge rm_{i,j} \Rightarrow c[xv \backslash xv']) \\ & \equiv & R.H.S. \end{array}$$

□

**Example 5.1 (Verifying Invariant).** We illustrate how to verify that an invariant is preserved by an operation in tabular form. Consider the table in Fig. 1 for defining the relation $ButtonPressed\,(f)$ over the variables *mode* and *reqs*. We want to show that the current floor cannot be in the set of requested floors, formally expressed by condition *NoReqForFloor*:

$$FloorNotInReqs\,(mode, reqs)\,(mode', reqs') \ \equiv\ floor \notin reqs$$

Applying Theorem 5.5 to show that $FloorNotInReqs \subseteq [ButtonPressed\,(f)]\ FloorNotInReqs$ results in six proof obligations, one for each body element:

1. $floor \notin reqs \wedge f > floor \wedge mode = waiting \wedge reqs' = \{f\} \wedge mode' = up \ \Rightarrow\ floor \notin reqs'$
2. $floor \notin reqs \wedge f > floor \wedge mode \neq waiting \wedge reqs' = reqs \cup \{f\} \wedge mode' = mode \ \Rightarrow\ floor \notin reqs'$
3. $floor \notin reqs \wedge f = floor \wedge mode = waiting \wedge reqs' = \{f\} \wedge mode' = waiting \ \Rightarrow\ floor \notin reqs'$
4. $floor \notin reqs \wedge f = floor \wedge mode \neq waiting \wedge reqs' = reqs \wedge mode' = mode \ \Rightarrow\ floor \notin reqs'$
5. $floor \notin reqs \wedge f < floor \wedge mode = waiting \wedge reqs' = \{f\} \wedge mode' = down \ \Rightarrow\ floor \notin reqs'$
6. $floor \notin reqs \wedge f < floor \wedge mode \neq waiting \wedge reqs' = reqs \cup \{f\} \wedge mode' = mode \ \Rightarrow\ floor \notin reqs'$

All six proof obligations can easily be seen to hold. We emphasise how the structure of the table leads to decomposing a larger proof into six smaller ones; we also observe that if one of the six proof obligations would not hold, then the source of the error can be easily traced back.

Alternatively we can verify the invariant by first deriving the weakest precondition for operation to establish the invariant and then showing in a second step that the invariant implies the weakest precondition. As $ButtonPressed\,(f)$ is defined by a table in standard form and the table headers are total and jointly disjoint, we apply Theorem 5.4 to determine the weakest precondition. The result after simplifications is given in Fig. 5. As both headers cover *true* we can apply Theorem 3.4 (c) to show that $floor \notin reqs$ implies the table. This again results in six proof conditions that are identical to the six above after eliminating the primed variables in those with the one-point rule.

**Vector Tables.** Of the various tables Parnas proposes we also consider *vector tables*, see Fig. 6. The upper header of a vector table is a vector of predicates, the left header is a vector of variables, and the body is a

| | $door = closing \wedge$ $mode = up$ | $door = closing \wedge$ $mode = down$ |
|---|---|---|
| $door' =$ | $closed$ | $closed$ |
| $motor' =$ | $up$ | $down$ |
| $mode' =$ | $mode$ | $mode$ |

| $door = closing \wedge$ $mode = up$ | $door = closing \wedge$ $mode = down$ |
|---|---|
| $door' = closed \wedge$ $motor' = up \wedge$ $mode' = mode$ | $door' = closed \wedge$ $motor' = down \wedge$ $mode' = mode$ |

**Fig. 6.** An example of a vector table (left) and an equivalent predicate table (right).

matrix of expressions. The meaning of this table is that the variable to the left is equal to all the expressions of that row. Vector tables have a two-dimensional structure, but are here defined in terms of one-dimensional predicate tables. Let $em^j$ stand for the $j$-th column of matrix $em$, formally $(em^j)_i = em_{i,j}$. Let $yv$ be a vector of variables and let $yv = em$ stand for a vector such that $(yv = em)_j \equiv yv = em^j$. We define:

$$\begin{array}{c|c} & pv \\ \hline yv = & em \end{array} \quad \equiv \quad \begin{array}{c} pv \\ \hline yv = em \end{array}$$

The typical use of vector tables is for defining relations. Consider the vector table in Fig. 6 for defining the relation *DoorClosing* over the variables *door*, *motor*, *mode*; it that models the state transition when the elevator door becomes closed. This table is in standard form as the upper header and the body mention only the variables of the initial state; the variables of the final state appear only in the left header. If the upper header of a vector table is total, then the relation is total. If the upper header is disjoint, then the relation is deterministic (functional). Hence this allows a relation to be analysed for totality and determinism by looking at one header only. In our example, we conclude from the upper header that *DoorClosing* is not total but deterministic.

For an operation given by a vector table we have a simplified rule for determining its precondition. Let $f[xv \backslash em]$ stand for a vector of expressions, with each element obtained by substituting $xv$ with one column of matrix $em$ in $f$, formally $(f[xv \backslash em])_j = f[xv \backslash em^j]$.

**Theorem 5.6 (Weakest Precondition of Vector Table).** If standard vector relation $V$ and condition $C$ are given by

$$V\ xv\ xv' \ \equiv \ \begin{array}{c|c} & bv \\ \hline xv' = & em \end{array}\ , \qquad C\ xv\ xv' \equiv c$$

we have:

$$[\,V\,]\ C\ xv\ xv' \ \equiv \ \ \wedge j \bullet bv_j \Rightarrow c[xv \backslash em^j]$$

*Proof.* We make use of Theorem 5.1:

$$\begin{array}{lll} L.H.S. \equiv & \forall xv' \bullet V\ xv\ xv' \Rightarrow C\ xv'\ xv' \\ \equiv & \forall xv' \bullet (\vee j \bullet bv_j \wedge xv' = em^j) \Rightarrow C\ xv'\ xv' \\ \equiv & \wedge j \bullet \forall xv' \bullet bv_j \wedge xv' = em^j \Rightarrow C\ xv'\ xv' \\ \equiv & R.H.S. \end{array}$$

□

While the theorem allows the precondition to be calculated, the precondition is a conjunction rather than a table. We can give an alternative theorem that gives a tabular precondition but has side conditions:

**Theorem 5.7 (Tabular Weakest Precondition of Vector Table).** If standard vector relation $V$ and condition $C$ are given by

$$V\ xv\ xv' \ \equiv \ \begin{array}{c|c} & bv \\ \hline xv' = & em \end{array}\ , \qquad C\ xv\ xv' \ \equiv \ c$$

and if $bv$ is total and disjoint we have:

$$[\,V\,]\ C\ xv\ xv' \ \equiv \ \begin{array}{c} bv \\ \hline c[xv \backslash em] \end{array}$$

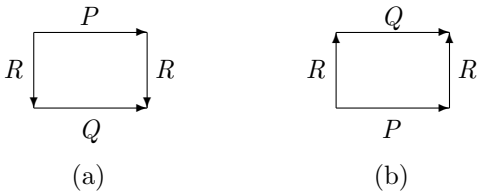


**Fig. 7.** Data refinement: (a) encoding from abstract to concrete with $R$ and (b) decoding from concrete to abstract with $R$.

*Proof.* This follows from Theorem 5.6 and Lemma 3.2 (c). □

Finally we give a theorem that does not have the side conditions of totality and disjointness. It follows directly from Theorem 5.5.

**Theorem 5.8 (Verification with Vector Table).** If conditions $B, C$ and vector relation $V$ are given by

$$B\ xv\ xv' \ \equiv\ b\ , \qquad V\ xv\ xv' \ \equiv\ \begin{array}{c|c} & pv \\ \hline xv' = & em \end{array}\ , \qquad C\ xv\ xv' \ \equiv\ c$$

we have:

$$B \subseteq [V]\ C \ \equiv\ \ \wedge j \bullet b \wedge pv_j \Rightarrow c[xv \backslash em^j]$$

**Example 5.2 (Verifying Invariant with Vector Table).** Consider the table in Fig. 6 for defining the relation *DoorClosing* over the variables *door*, *motor*, *mode*. We like to show that the direction of *motor* must correspond to *mode* by showing that the condition *MotorUpMode* is preserved:

$$MotorUpMode\,(door, motor, mode)\,(door', motor', mode') \quad \equiv \quad motor = up \Rightarrow mode = up$$

We note that we cannot apply Theorem 5.7 as the header is not total and we cannot use Theorem 2.7 for making the header of a vector table total. Thus we can only apply Theorem 5.6 or 5.8. Applying the former we get

$$\begin{array}{ll} & [DoorClosing]\ MotorUpMode\,(door, motor, mode)\,(door', motor', mode') \\ \equiv & (door = closing \wedge mode = up \Rightarrow (up = up \Rightarrow mode = up)) \wedge \\ & (door = closing \wedge mode = down \Rightarrow (down = up \Rightarrow mode = up)) \end{array}$$

which is always true, and hence the invariant is trivially preserved.

## 6. Refinement

For programs $P$ and $Q$, if $P \subseteq Q$ then we say that $P$ *refines* $Q$. Refinement is a process that allows nondeterminism to be reduced. Refinement is reflexive, $P \subseteq P$, meaning that each program is refined by itself. Refinement is also transitive, $P \subseteq Q$ and $Q \subseteq R$ implies $P \subseteq R$, meaning that programs can be refined in a stepwise manner. If $P \subseteq Q$ holds, then $P$ is called the (more) concrete and $Q$ the (more) abstract program. We are interested in generalising this notion of *algorithmic refinement* to *data refinement*, where the concrete and abstract program work on different state spaces. We consider two variants of data refinement, *downward (forward) data refinement* and *upward (backward) data refinement* [HHS86], noting that in the predicate transformer setting these two variants can be unified to a single notion of data refinement [BW00]. An extensive discussion of data refinement in the relational setting is in [RE98]. We follow the lines of [BW00] and adopt that approach to the relational setting.

Downward data refinement is defined through following subcommutativity property, see Fig. 7 (a). Suppose $P, Q$ are homogeneous relations of possibly different types. Program $Q$ downward refines program $P$ via relation $R$ if $R \circ Q \subseteq P \circ R$ holds. Relation $R$ is called the *refinement relation* or *encoding relation*, relating the state space of the (more) abstract program $P$ to that of the (more) concrete program $Q$. We use downward refinement when the abstract program and the encoding relation are given and the concrete program is to be determined in a systematic way. For this we introduce an *encoding operator* $P \downarrow R$:

$$P \downarrow R \ \ = \ \ R^{-1} \circ P \circ R \qquad \text{provided } R \text{ is injective}$$

The restriction to an injective encoding relation is motivated by following theorem:

**Theorem 6.1 (Soundness of Encoding).**

$$Q \subseteq P \downarrow R \;\Rightarrow\; R \circ Q \subseteq P \circ R \qquad \text{if } R \text{ is injective}$$

*Proof.* We apply the definition of encoding and injectivity:

$$\begin{array}{lcl} L.H.S. & \equiv & Q \subseteq R^{-1} \circ P \circ R \\ & \Rightarrow & R \circ Q \subseteq R \circ R^{-1} \circ P \circ R \\ & \Rightarrow & R.H.S. \end{array}$$

□

Upward data refinement is defined through a similar subcommutativity property, see Fig. 7 (b). Program $P$ upward refines program $Q$ via relation $R$ if $P \circ R \subseteq R \circ Q$ holds. Relation $R$ is called the *abstraction relation* or *decoding relation*, relating the state space of the (more) concrete program $P$ to that of the (more) abstract program $Q$. We use upward refinement when the concrete program and the decoding relation are given and the abstract program is to be determined in a systematic way. For this we introduce a *decoding operator* $P \uparrow R$:

$$P \uparrow R \;=\; R^{-1} \circ P \circ R \qquad \text{provided } R \text{ is total}$$

The restriction to a total decoding relation is motivated by following theorem:

**Theorem 6.2 (Soundness of Decoding).**

$$P \uparrow R \subseteq Q \;\Rightarrow\; P \circ R \subseteq R \circ Q \qquad \text{if } R \text{ is total}$$

*Proof.* We apply the definition of decoding and totality:

$$\begin{array}{lcl} L.H.S. & \equiv & R^{-1} \circ P \circ R \subseteq Q \\ & \Rightarrow & R \circ R^{-1} \circ P \circ R \subseteq R \circ Q \\ & \Rightarrow & R.H.S. \end{array}$$

□

As encoding and decoding differ only in the restriction of $R$, we define a general *coding operator* $P \updownarrow R$ without the restriction to $R$ being either injective or total. While this allows us to state properties that apply to both encoding and decoding only once, the use of coding is only sound (i.e. implies data refinement) if $R$ is either injective or total:

$$P \updownarrow R \;=\; R^{-1} \circ P \circ R$$

We give first theorems about coding in general and then theorems that apply only to encoding or to decoding. We note that coding is monotonic in its first argument (but not in its second), which follows directly from its definition:

**Theorem 6.3 (Monotonicity of Coding).**

$$P \subseteq Q \;\Rightarrow\; P \updownarrow R \subseteq Q \updownarrow R$$

We state some facts about the first argument of the coding operator.

**Theorem 6.4.** Suppose $I$ is an index set and $C$ is a condition:

(a) $\bot \updownarrow R \;=\; \bot$

(b) $(\cup i \in I \bullet P_i) \updownarrow R \;=\; (\cup i \in I \bullet P_i \updownarrow R)$

(c) $(\cap i \in I \bullet P_i) \updownarrow R \;\subseteq\; (\cap i \in I \bullet Q_i \updownarrow R)$

(d) $(C \cap P) \updownarrow R \;\subseteq\; (R^{-1} \circ C) \cap (P \updownarrow R)$

*Proof.* Part (a) follows immediately from the definition. Part (b) follows from Lemma 4.1 (a) and (b). Part (c) follows from Lemma 4.1 (c) and (d). Finally, (d) follows from Lemma 4.1 (c) and (g). □

We note that for a relation $R$ and condition $C$, the condition $R^{-1} \circ C$ is the image of $C$ under $R$. As Theorem 6.4 (c) and (d) state inclusion and not equality, they are only useful for decoding when distributing the decoding operator into conjunctions. Next we state how coding behaves in its second argument:

**Theorem 6.5.** Suppose $I$ is an index set:

(a) $P \updownarrow \bot \;=\; \bot$
(b) $P \updownarrow \top \;=\; \top \qquad$ if $P \neq \bot$
(c) $P \updownarrow Id \;=\; P$
(d) $P \updownarrow (R \circ S) \;=\; (P \updownarrow R) \updownarrow S$
(e) $(\cup i \in I \bullet P \updownarrow R_i) \;\subseteq\; P \updownarrow (\cup i \in I \bullet R_i)$
(f) $P \updownarrow (\cap i \in I \bullet R_i) \;\subseteq\; (\cap i \in I \bullet P \updownarrow R_i)$

*Proof.* Parts (a) to (c) all follow from basic properties of relations. For (d) we use that inversion distributes through relational composition, $(R \circ S)^{-1} = S^{-1} \circ R^{-1}$. For (e) we use Lemma 4.1 (a) and (b), and that inversion distributes through generalised union:

$$\begin{array}{lcl} R.H.S. & = & (\cup i \in I \bullet R_i)^{-1} \circ P \circ (\cup i \in I \bullet R_i) \\ & = & (\cup i \in I \bullet R_i^{-1}) \circ P \circ (\cup i \in I \bullet R_i) \\ & = & (\cup i \in I \bullet R_i^{-1} \circ P \circ (\cup i \in I \bullet R_i)) \\ & \supseteq & (\cup i \in I \bullet R_i^{-1} \circ P \circ R_i) \\ & = & L.H.S. \end{array}$$

The proof of (f) is similar, except that we use Lemma 4.1 (c) and (d), and that inversion distributes through generalised intersection. □

We continue with theorems that apply only to encoding. Distributivity through conjunctions in the first argument can be strengthened to equality with an injective encoding relation. Encoding subdistributes through relational composition:

**Theorem 6.6.** Suppose $R$ is an injective relation:

(a) $(\cap i \in I \bullet P_i) \downarrow R \;=\; (\cap i \in I \bullet Q_i) \downarrow R$
(b) $(C \cap P) \downarrow R \;=\; (R^{-1} \circ C) \cap (P \downarrow R)$
(c) $(P_1 \downarrow R) \circ (P_2 \downarrow R) \;\subseteq\; (P_1 \circ P_2) \downarrow R$

*Proof.* Part (a) follows from Lemma4.1 (e) and (f), and the fact that $R^{-1}$ is functional. Part (b) follows from Lemma 4.1 (g) and (e). Part (c) follows from the definition of injectivity. □

We conclude with a theorem that applies only to decoding. Decoding also subdistributes through relational composition, though in the other direction than encoding:

**Theorem 6.7.** Suppose $R$ is a total relation:

$$(P_1 \circ P_2) \uparrow R \;\subseteq\; (P_1 \uparrow R) \circ (P_2 \uparrow R)$$

While our encoding and decoding operators are similar to those of [BW00], the differences are subtle but substantial. In the predicate transformer setting a relation $R$ can be lifted to either an angelically updating statement $\{R\}$ or to a demonically updating statement $[R]$. Encoding of statement $S$ with relation $R$ is defined as $S \downarrow R = \{R^{-1}\}; S; [R]$ and decoding as $S \uparrow R = [R^{-1}]; S; \{R\}$. Both are unconditionally sound in the sense that $S; [R] \sqsubseteq [R]; T \equiv S \downarrow R \sqsubseteq T \equiv S \sqsubseteq T \uparrow R \equiv \{R^{-1}\}; S \sqsubseteq T; \{R^{-1}\}$ for any statements $S, T$ and any relation $R$, where $S \sqsubseteq S'$ means that $S$ is refined by $S'$. The relational setting has only one kind of nondeterminism, so the definitions of encoding and decoding coincide. However, Theorem 6.1 states soundness of encoding only if $R$ is injective and only gives an implication. Likewise, Theorem 6.2 states soundness of decoding only if $R$ is total and only gives an implication. While Theorems 6.3, 6.4, and 6.5 hold in similar form in the predicate transformer setting, Theorems 6.6 and 6.7 can be generalised to hold for arbitrary relations $R$ with predicate transformers.

## 7. Tabular Refinement

We analyse how specifications can be transformed into more concrete or more abstract ones, where either the concrete or the abstract or both are given in tabular form. First we consider that both specifications are over the same state space. Assume $PV$ and $QV$ are vectors of relations, $RM$ is a matrix of relations, and $S$ is a relation:

**Theorem 7.1 (Refining to Table).**

(a) $\dfrac{\begin{array}{c|c} & QV \\ \hline PV & RM \end{array}}{} \subseteq S \equiv \wedge i, j \bullet PV_i \cap QV_j \cap RM_{i,j} \subseteq S$

(b) $\dfrac{\begin{array}{c|c} & QV \\ \hline PV & RM \end{array}}{} \subseteq \dfrac{\begin{array}{c|c} & QV \\ \hline PV & SM \end{array}}{} \Leftarrow \wedge i, j \bullet PV_i \cap QV_j \cap RM_{i,j} \subseteq SM_{i,j}$

Refining to a vector table allows for a simplified rule:

**Theorem 7.2 (Refining to Vector Table).** If vector relation $P$ and relation $Q$ are given by

$$P\ xv\ xv' \equiv \frac{\begin{array}{c|c} & pv \\ \hline xv' = & em \end{array}}{}, \qquad Q\ xv\ xv' \equiv \frac{pv}{qv}$$

we have:

$$P \subseteq Q \Leftarrow (\wedge j \bullet pv_j \Rightarrow qv[xv' \backslash em^j])$$

Note that while above theorem can be applied even if $Q$ is not a standard relation, $P$ is a standard vector relation only if $Q$ is a standard relation. We now give a general theorem when the concrete and abstract state are related through relation $R$:

**Theorem 7.3 (Data Refining a Table).** Assume $BV, CV$ are vectors of conditions:

(a) $\left(\dfrac{\begin{array}{c|c} & CV \\ \hline BV & PM \end{array}}{}\right) \downarrow R = \dfrac{\begin{array}{c|c} & R^{-1} \circ CV \\ \hline R^{-1} \circ BV & PM \downarrow R \end{array}}{}$ if $R$ is injective

(b) $\left(\dfrac{\begin{array}{c|c} & CV \\ \hline BV & PM \end{array}}{}\right) \updownarrow R \subseteq \dfrac{\begin{array}{c|c} & R^{-1} \circ CV \\ \hline R^{-1} \circ BV & PM \updownarrow R \end{array}}{}$

Removing the side condition that the refinement relation is injective in part (a) only gives the inclusion of part (b). While (b) is applicable to both decoding and encoding, the direction of the inclusion makes it only useful for decoding. Thus when using (a) for refinement, we get an exact refinement, but when using (b) for abstraction, we get only an approximation.

*Proof.* For (a) make use of Theorem 4.2 (a) and (c) as $R^{-1}$ is functional:

$$\begin{array}{lcl} L.H.S. & = & R^{-1} \circ \dfrac{\begin{array}{c|c} & CV \\ \hline BV & PM \end{array}}{} \circ R \\ & = & \dfrac{\begin{array}{c|c} & R^{-1} \circ CV \\ \hline R^{-1} \circ BV & R^{-1} \circ PM \circ R \end{array}}{} \\ & = & R.H.S. \end{array}$$

For (b) we make use of Theorem 4.2 (a) and (b):

$$\begin{array}{lcl} L.H.S. & = & R^{-1} \circ \dfrac{\begin{array}{c|c} & CV \\ \hline BV & PM \end{array}}{} \circ R \\ & \subseteq & \dfrac{\begin{array}{c|c} & R^{-1} \circ CV \\ \hline R^{-1} \circ BV & R^{-1} \circ PM \circ R \end{array}}{} \\ & = & L.H.S. \end{array}$$

□

To allow a direct application of above theorem, we derive the corresponding theorem when the relation is given by a tabular predicate. We extend the use of existential quantifications to matrices of predicates, with the meaning that the quantification is applied to each element, formally $(\exists x \bullet pm)_{i,j} \equiv (\exists x \bullet pm_{i,j})$:

**Theorem 7.4 (Data Refining with Predicates).** Given relation $P$ in standard form and relation $R$ by

$$P\,xv\,xv' \;\equiv\; \begin{array}{c|c} & cv \\ \hline bv & pm \end{array}, \qquad R\,xv\,yv \;\equiv\; r$$

and writing $r'$ for $r$ with $xv, yy$ substituted by $xv', yv'$ we have:

$$\text{(a)} \quad (P \downarrow R)\,yv\,yv' \;=\; \left(\begin{array}{c|c} & \exists xv \bullet r \wedge cv \\ \hline \exists xv \bullet r \wedge bv & \exists xv, xv' \bullet r \wedge pm \wedge r' \end{array}\right) \qquad \text{if } R \text{ is injective}$$

$$\text{(b)} \quad (P \updownarrow R)\,yv\,yv' \;\subseteq\; \left(\begin{array}{c|c} & \exists xv \bullet r \wedge cv \\ \hline \exists xv \bullet r \wedge bv & \exists xv, xv' \bullet r \wedge pm \wedge r' \end{array}\right)$$

We consider the case that the refinement relation rather than the specification is in tabular form. More precisely, we consider the refinement relation being defined by an inverted vector table, that is a table in which only the variables of the initial state appear in the left header and variables of the final state appear only in the upper header and body. For simplicity we consider a refinement relation with only two columns.

**Theorem 7.5 (Data Refinement with Vector Table).** Assume inverted vector relation $R$ is given by:

$$R\,xv\,yv \;\equiv\; \begin{array}{c|c|c} & c & d \\ \hline xv = & ev & fv \end{array}$$

Writing $c'$, $d'$, $ev'$, $fv'$ for $c$, $d$, $ev$, $fv$ with $yv$ substituted by $yv'$ we have:

$$(P \updownarrow R)\,yv\,yv' \;\equiv\; \begin{array}{c|c|c} & c' & d' \\ \hline c & P\,ev\,ev' & P\,ev\,fv' \\ \hline d & P\,fv\,ev' & P\,fv\,fv' \end{array}$$

*Proof.*

$$\begin{array}{lcl} L.H.S. & \equiv & \exists xv, xv' \bullet R^{-1}\,yv\,xv \wedge P\,xv\,xv' \wedge R\,xv'\,yv' \\ & \equiv & \exists xv, xv' \bullet ((c \wedge xv = ev) \vee (d \wedge xv = fv)) \wedge P\,xv\,xv' \wedge ((c' \wedge xv' = ev') \vee (d' \wedge xv' = fv')) \\ & \equiv & (c \wedge P\,ev\,ev' \wedge c') \vee (c \wedge P\,ev\,fv' \wedge d') \vee (d \wedge P\,fv\,ev' \wedge c') \vee (d \wedge P\,fv\,fv' \wedge d') \\ & \equiv & R.H.S. \end{array}$$

□

We discuss the use of this theorem for encoding. For the encoding relation $R$ to be injective it is sufficient if the header of $R$ is disjoint, i.e. $\neg c \vee \neg d$ holds. The header defines the *concrete invariant* $c \vee d$ and the body with $ev$ and $fv$ defines the *abstraction function* in a piecewise manner; that is, if $c$ holds on the concrete state then the abstraction function is given by $ev$ and if $d$ holds it is given by $fv$. For the refinement $P \downarrow R$ we observe that either $c$ or $d$ must hold initially and either $c'$ or $d'$ must hold finally, giving in total four combinations of how the refinement may establish $c'$ or $d'$ from $c$ or $d$. Note that the resulting table is neither in standard nor in inverted form. However, we may flatten the table after transposing it, thus eliminating the upper header and bringing it into standard form.

For the use of this theorem for decoding, we observe that the decoding relation $R$ assigns values to the concrete variables in terms of the abstract variables. As the relation has also to be total, this makes the application of the theorem to decoding too restrictive.

**Example 7.1 (Data Abstraction).** We illustrate the use of Theorem 7.4 (b) for a data abstraction that reduces the state space. Consider applying decoding to the relation *ButtonPressed* ($f$) over variables *mode* and *reqs* as defined in Fig. 1. Our intention is to abstract variable *reqs* with a boolean variable $r$ that only reflects if *reqs* is empty and to abstract variable *mode* with a boolean variable $w$ that only reflects whether *mode* is *waiting* or not. Thus this abstraction reduces the state space to two boolean variables. A typical use of such an abstraction is to allow (automated) proofs about the abstraction, for example the property that if there are no requests then the mode must be *waiting*. Formally our decoding relation is:

$$RW\,(reqs, mode)\,(r, w) \;\equiv\; (r \equiv reqs \neq \{\}) \wedge (w \equiv mode = waiting)$$

The result of applying Theorem 7.4 (b) and further simplifications is the left table in Fig. 8. We now use Theorem 2.9 (with transposition) to join the first and last row and Theorem 2.8 (a) to simplify the

| | $w$ | $\neg w$ |
|---|---|---|
| $f > \mathit{floor}$ | $r' \wedge \neg w'$ | $r' \wedge w' = w$ |
| $f = \mathit{floor}$ | $\neg r' \wedge w'$ | $r' = r \wedge w' = w$ |
| $f < \mathit{floor}$ | $r' \wedge \neg w'$ | $r' \wedge w' = w$ |

| | $w$ | $\neg w$ |
|---|---|---|
| $f \neq \mathit{floor}$ | $r' \wedge \neg w'$ | $r' \wedge \neg w'$ |
| $f = \mathit{floor}$ | $\neg r' \wedge w'$ | $r' = r \wedge \neg w'$ |

**Fig. 8.** Two abstractions of the specification in Fig. 1 using decoding relation $RW$.

rightmost column. The final result is the right table in Fig. 8. For example, we can now show that if there are no requests then mode must be waiting proving that $\neg r \Rightarrow w$ is preserved by this abstraction. Recall that Theorem 7.4 (b) states only inclusion. The example shows that the approximate abstraction we get from this theorem is still useful.

## 8. Outlook

Among the presented theorems we point out two promising classes: one class are theorems that decompose a potentially large proof condition into a set of smaller proof conditions following the structure of the table. Theorems 3.4 (Predicate-Table Implication) and 5.3 (Tabular Verification) belong to this class. Thus for a table with $m$ rows and $n$ columns these theorems give $m \times n$ proof conditions. Example 5.1 (Verifying Invariant) illustrates that compared to conventional proofs, this leads to more, but simpler proof conditions. Given that both automated theorem proving and state enumeration (model checking) techniques can cope disproportionately better with smaller conditions, there is the potential of higher automation of proofs.

The other class of theorems are those that preserve the structure of tables in transformations by distributing an operation into a table. Theorems 4.2 (Table Composition), 5.2 (Tabular Weakest Precondition) and 7.3 (Data Refining a Table) belong to this class and Example 7.1 (Data Abstraction) illustrates a use. While the potential of these structure preserving transformations remains to be further explored, a benefit is that they ease tracing back errors. Particularly this class of theorems would be awkward if expressed in textual form.

Beside the potential for automation we stress the readability that tables offer. Our own experience in teaching predicate tables is that students make fewer errors when writing specifications than with plain predicates. Likewise, students find it easier to read and implement a specification given in tabular form than one given as a plain predicate.

Of the ten kinds of tables that Parnas proposes, predicate expression tables correspond to our predicate tables; normal relation tables and characteristic predicate tables correspond to our relation tables, with the difference that Parnas' tables are made up of predicates and come with conventions on how the variables of the initial and final state are identified. Here we always give them explicitly as the parameters of the relation. Inverted relation tables correspond to our inverted tables and vector relation tables are similar to our vector tables, except that our vector tables denote predicates, not relations. We have not covered Parnas' normal function tables, inverted function tables, vector function tables, mixed vector tables, and generalised decision tables. We note that in principle functions can always be represented by deterministic relations. We have not covered tables of higher dimensions, tables with structured headers, and nested tables. Our experience is that mixed vector tables—tables in which some variables are "assigned" values by equalities and some by predicates—are handy when describing relations that are deterministic in some variables and nondeterministic in others. Also, structured headers in which some expressions may span over several rows or columns allow avoiding repetitions. Incorporating these would be desirable. We doubt that with tables of higher dimensions than two we would be able to retain the visual structure of our theorems. While in principle our tables can be nested—after all each table just denotes a predicate or a relation—our experience is that even plain tables too quickly fill up a sheet so that nesting them does not appear to be useful.

It is interesting to note that the study of algebraic laws of conditional expressions $(p_1 \rightarrow e_1, \ldots, p_n \rightarrow e_n)$ goes back to early work of McCarthy [McC63]. Such conditional expressions are similar to Parnas' function tables, except that they are evaluated from left to right and have a special undefined value if all $p_i$ are false. McCarthy uses conditional expressions to construct computable functions, whereas Parnas' tables are meant for specification. Hoare gives a number of algebraic laws of conditionals of the form $p \lhd q \rhd r$, read "$p$ if $q$

else $r$" [Hoa85]. Such conditionals are just one-dimensional tables in canonical form:

$$p \lhd q \rhd r \quad \equiv \quad \begin{array}{c|c} q & \neg q \\ \hline p & r \end{array}$$

Despite the simple structure of tables, more applications of tables and more mathematical properties remain to be explored. We studied the composition of tables with identical headers. It would be interesting to see how this can be generalised, for example for showing that the table in Fig. 1 implies the one in Fig. 4. Another application of tables would be for writing pre- and postconditions in tabular form in addition to writing operations in tabular. The verification rules like Theorem 5.2 do indeed lead to such tables. More experience is needed to determine the usefulness of such generalisations.

Finally, we note that moving from tabular predicates to tabular relations relied only on basic properties of $\wedge$ and $\vee$. One could move on further to tabular predicate transformers, with conjunction and disjunction becoming demonic choice and angelic choice [BW98]. One may conjecture that for constructing tables of the kind discussed here a lattice structure is sufficient.

**Acknowledgement.** The author likes to thank the reviewers; their comments led to a significant improvement. Eerke Boiten provided further helpful comments and pointed out an error.

## References

[Abr97] R. Abraham. Evaluating generalized tabular expressions in software documentation. CRL Report 346, McMaster University, February 1997.

[BW98] R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.

[BW00] R.-J. Back and J. von Wright. Encoding, decoding and data refinement. *Formal Aspects of Computing*, 12(5):313–349, 2000.

[Hen80] K. L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering*, 6(1):2–13, 1980.

[HHS86] Jifeng He, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *European Symposium on Programming*, Lecture Notes in Computer Science 213. Springer-Verlag, 1986.

[HJL96] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specification. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, 1996.

[HL96] M. P. E. Heimdahl and N. G. Leveson. Completeness and consistency in hierarchical state-based requirements. *IEEE Transactions on Software Engineering*, 22(6):363–377, 1996.

[Hoa85] C. A. R. Hoare. A couple of novelties in the propositional calculus. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 31(2):173–178, 1985.

[Jan95] R. Janicki. Towards a formal semantics of Parnas tables. In *17th International Conference on Software Engineering*, pages 231–240, Seattle, Washington, USA, 1995. ACM Press.

[JK01] R. Janicki and R. Khedri. On a formal semantics of tabular expressions. *Science of Computer Programming*, 39(2–3):189–213, 2001.

[LHHR94] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684–707, 1994.

[LMFM00] M. Lawford, J. McDougall, P. Froebel, and G. Moum. Practical application of functional and relational methods for the specification and verification of safety critical software. In T. Rus, editor, *Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000*, Lecture Notes in Computer Science 1816, pages 73–88, Iowa City, Iowa, USA, 2000. Springer-Verlag.

[McC63] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland, Amsterdam, 1963.

[ORS97] S. Owre, J. Rushby, and N. Shankar. Integration in PVS: Tables, types, and model checking. In *Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1217, pages 336–383, Enschede, The Netherlands, 1997. Springer-Verlag.

[Par92] D. L. Parnas. Tabular representation of relations. CRL Report 260, McMaster University, October 1992.

[RE98] W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge Tracts in Theoretical Computer Science 47. Cambridge University Press, 1998.

[WT95] A. Wilder and J. Tucker. System documentation using tables—a short course. http://www.swan.ac.uk/compsci/ResearchGroups/TheoryGroups/AlgMethFolder/DSTFolder/TablesForSystems/CourseNotesWithPictures.ps.Z, Department of Computer Science, University of Wales Swansea, May 1995.

[Zuc96] J. I. Zucker. Transformations of normal and inverted function tables. *Formal Aspects of Computing*, 8(6):679–705, 1996.