*Journal of Universal Computer Science, vol. 3, no. 4 (1997), 304-319*
*submitted: 20/12/96, accepted: 21/4/97, appeared: 28/4/97* © *Springer Pub. Co.*

# The Constrained Shortest Path Problem: A Case Study in Using ASMs

Karl Stroetmann
(Siemens AG, 81730 Munich, Germany
Karl.Stroetmann@mchp.siemens.de)

**Abstract:** This paper addresses the correctness problem of an algorithm solving the *constrained shortest path problem.* We define an abstract, nondeterministic form of the algorithm and prove its correctness from a few simple axioms. We then define a sequence of natural refinements which can be proved to be correct and lead from the abstract algorithm to an efficient implementation due to Ulrich Lauther [Lauther 1996] and based on [Desrosiers et al. 1995]. Along the way, we also show that the abstract algorithm can be regarded as a natural extension of Moore's algorithm [Moore 1957] for solving the *shortest path problem.*
**Key Words:** abstract state machine, software verification, shortest path problem, constrained shortest path problem
**Category:** D.2.4, F.3.1, G.2.2

## 1 Introduction

The subject of this case study is an algorithm to solve the *constrained shortest path problem.* This problem is specified in Section 2. We develop an abstract generic algorithm for solving *shortest path problems* in Section 3, and prove its correctness from a few simple axioms in Section 4. This abstract algorithm is non–deterministic. This non-determinism is eliminated in Section 5, yielding a generic deterministic algorithm. In Section 6, this generic algorithm is instantiated in order to solve the constrained shortest path problem. To this end, we provide an implementation of certain operations on step functions. The resulting algorithm can readily be translated into an efficient `C++` program.

The algorithms are presented as *abstract state machines* (ASM), a notion introduced by Gurevich under the name of evolving algebras [Gurevich 1993]. ASMs can be regarded as pseudo code. However, in contrast to pseudo code, ASMs have a rigorous semantics, formally defined in Gurevich [Gurevich 1995]. The notation used to present an ASM is mostly self explanatory. For the reader not familiar with [Gurevich 1995] we explain this notation and its semantics on an informal level.

The basic notion of an ASM is the notion of an *update.* It takes the form

$$f(t_1, \cdots, t_n) \;\texttt{:=}\; s$$

where $f$ is a function symbol and $s$, $t_1$, $\cdots$, $t_n$ are expressions that can be evaluated. If the evaluation of these expressions produces the values $v_0$, $v_1$, $\cdots$, $v_n$, respectively, then the value of $f(v_1, \cdots, v_n)$ is changed to $v_0$, as the effect of this update.

Updates may be combined into *blocks* of updates. A block of $n$ updates takes the form

$$f(\boldsymbol{t_1}) \ \texttt{:=}\ s_1$$
$$\vdots$$
$$f(\boldsymbol{t_n}) \ \texttt{:=}\ s_n$$

where $\boldsymbol{t_1}, \cdots, \boldsymbol{t_n}$ denote tuples of expressions. To execute a block of $n$ updates, all of its updates are executed **simultaneously**. (The fact that these updates are executed simultaneously rather than sequentially is perhaps the only important difference between conventional programming languages and the notion of abstract state machines.)

The *qualified choose* construct can be used to describe non-determinism. Its form is

> **choose** $x$ **in** $S$ **satisfying** $p(x)$
> $\quad B(x)$

where $x$ is a variable, $S$ is a sort, $p(x)$ is a Boolean expression containing the variable $x$, while $B(x)$ is a block of updates that contains expressions mentioning the variable $x$. To execute this *choose* construct we non-deterministically choose a value $v$ with sort $S$ such that $p(v)$ is satisfied. This value is then substituted for $x$ in $B(x)$ and the resulting block $B(v)$ is executed. If there is no value $v$ in $S$ such that $p(v)$ is satisfied, then the computation terminates.

The *guarded block* has the form

> **if** $G$ **then**
> $\quad B$

where $G$ is a Boolean expression referred to as the *guard* of the block $B$. This block is executed only if the guard $G$ evaluates to `true`.

It should be noted that the *choose* construct and the *guarded block* can be mixed freely. All further notations used like, e.g. the *initialization*, are self explanatory. The above explanation of abstract state machines should be sufficient for the rest of this paper. The reader interested in a rigorous definition is advised to consult [Gurevich 1995]. The methodology applied in this paper has been suggested by Börger, cf. [Börger 1995].

## 2 Preliminaries

The *constrained shortest path problem* is a generalization of the *shortest path problem*. For didactic purposes, we define this simpler and well known problem first. To this end, we introduce the notion of a *weighted graph*, i.e. a graph where a *weight* is associated with every edge.

**Definition 1 (weighted graph)** *A* weighted graph *is a triple*

$$\langle \mathbf{Nodes}, \mathbf{Edges}, \mathtt{weight} \rangle$$

*such that the pair* $\langle \mathbf{Nodes}, \mathbf{Edges} \rangle$ *is a directed graph and* $\mathtt{weight} : \mathbf{Edges} \rightarrow \mathbb{N}$ *is a function assigning a natural number to every edge. If* $\mathtt{weight}(e) = l$*, then* $l$ *is also called the* length *of* $e$*. For any edge* $e = \langle v, w \rangle$ *we use the notation* $\mathtt{head}(e) = w$ *and* $\mathtt{tail}(e) = v$*, i.e. an edge is regarded as an arrow pointing from* $\mathtt{tail}(e)$ *to* $\mathtt{head}(e)$*.* $\square$

The definition of a weighted graph given above is actually the definition of a *directed* weighted graph. However, there is no loss in generality since an undirected weighted graph can be seen as a special case of a weighted graph where the set **Edges** and the function `weight` are symmetric.

Paths are defined as usual. The set of all paths will be denoted as **Paths**. The function `weight` is extended to **Paths** by collecting all the weights along the path, i.e.

$$\mathtt{weight}(e_1 e_2 \cdots e_n) \;:=\; \sum_{i=1}^{n} \mathtt{weight}(e_i).$$

If $p = e_1 e_2 \cdots e_n$ is a path in $G$, then we say that $p$ *connects* the node $\mathtt{tail}(e_1)$ with the node $\mathtt{head}(e_n)$. Furthermore, the *empty path* $\varepsilon$ connects every node to itself. The set of all path connecting node $x$ with node $y$ will be denoted as $\mathbf{Paths}(x, y)$.

In the rest of this paper we will assume that every graph is finite. In general, the set $\mathbf{Paths}(x, y)$ is not finite, since there may exist paths containing cycles. However, in the application we have in mind we can restrict our attention to paths containing no cycles and this set is finite.

**Definition 2 (shortest path problem)** *Given a distinguished node* `source`*, the* single source shortest path problem *consists in computing the following function:*

$$\begin{aligned} &\mathtt{sp} : \mathbf{Nodes} \to \mathbb{N} \\ &\mathtt{sp}(v) \;:=\; \mathtt{min}\big\{\mathtt{weight}(p) \;:\; p \in \mathbf{Paths}(\mathtt{source}, v)\big\}. \end{aligned}$$ □

For the *constrained shortest path problem* we redefine the notion of a *weighted graph* by letting the function `weight` assign a pair of natural numbers to every edge, i.e. we have $\mathtt{weight} : \mathbf{Edges} \to \mathbb{N} \times \mathbb{N}$. If $\mathtt{weight}(e) = \langle l, c \rangle$, then $l$ is called the *length* of $e$ and $c$ is called the *cost* of $e$. Therefore, we introduce two functions `length` and `cost` satisfying $\mathtt{weight}(e) = \langle \mathtt{length}(e), \mathtt{cost}(e) \rangle$. These functions are extended from **Edges** to **Paths**:

$$\begin{aligned} \mathtt{length}(e_1 e_2 \cdots e_n) \;&:=\; \sum_{i=1}^{n} \mathtt{length}(e_i), \\ \mathtt{cost}(e_1 e_2 \cdots e_n) \;&:=\; \sum_{i=1}^{n} \mathtt{cost}(e_i). \end{aligned}$$

**Definition 3 (SF)** *A function* $f : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ *is a* monotonically decreasing step function *iff*

$$l_1 \leq l_2 \;\Rightarrow\; f(l_2) \leq f(l_1).$$

*The set of all monotonically decreasing step functions is denoted by* **SF**. □

**Definition 4 (constrained shortest path problem)** *Given a distinguished node* `source`*, the* constrained shortest path problem *consists in computing the following function:*

$$\begin{aligned} &\mathtt{csp} : \mathbf{Nodes} \to \mathbf{SF} \\ &\mathtt{csp}(v)(l) \;:=\; \mathtt{min}\big\{\mathtt{cost}(p) \;:\; p \in \mathbf{Paths}(\mathtt{source}, v) \,\wedge\, \mathtt{length}(p) \leq l\big\}. \end{aligned}$$ □

In the form given above, it is not easy to see that the *constrained shortest path problem* and the *shortest path problem* are instances of the same problem. In order to emphasize the similarity between these problems we reformulate the *constrained shortest path problem.* Our first task is to extend the function `weight` from **Edges** to **Paths**. Since $\mathtt{weight}(e)$ is composed of $\mathtt{length}(e)$ and $\mathtt{cost}(e)$, we can use the extensions of these functions in order to generalize `weight`.

**Definition 5** *If $p$ is a path in $G$ and $l$ is a natural number, then we define:*

$$\mathtt{weight}(p)(l) \;:=\; \begin{cases} \infty & \text{if } \; l < \mathtt{length}(p); \\ \mathtt{cost}(p) & \text{if } \; l \geq \mathtt{length}(p). \end{cases} \qquad \Box$$

Note that when the domain of `weight` is changed from **Edges** to **Paths**, the range of `weight` changes from $\mathbb{N} \times \mathbb{N}$ to the function space **SF**.

The above definition of $\mathtt{weight}(p)$ is unsatisfactory because it is structurally different from the definition of `weight` in the case of the *shortest path problem.* To be able to give an inductive definition of `weight`, which is similar to the definition of this function in the shortest path problem, we have to introduce an addition for functions from **SF** and weights from $\mathbb{N} \times \mathbb{N}$.

**Definition 6 (+)** *For $f \in \mathbf{SF}$ and $\langle x, y\rangle \in \mathbb{N} \times \mathbb{N}$ the function $f + \langle x, y\rangle$ is defined as follows:*

$$\big(f + \langle x, y\rangle\big)(t) = \begin{cases} \infty & \text{if } \; t < x; \\ f(t-x) + y & \text{if } \; t \geq x. \end{cases} \qquad \Box$$

The operation + can be used to give an alternative definition of `weight`.

**Definition 7** *For a path $p$, $\mathtt{weight}'(p)$ is defined by induction on $p$:*

1. $\mathtt{weight}'(\varepsilon) \;:=\; \mathbf{0}$.
   *(Here the function $\mathbf{0} \in \mathbf{SF}$ is defined as $\mathbf{0}(l) = 0$.)*
2. $\mathtt{weight}'(p\,e) \;:=\; \mathtt{weight}'(p) + \mathtt{weight}(e)$. $\Box$

**Lemma 8** *For any path $p$ we have:*

$$\mathtt{weight}'(p) = \mathtt{weight}(p) \qquad \Box$$

**Proof**: The proof is by induction on the number of edges in $p$.

**Base case:** $p = \varepsilon$. Then $\mathtt{length}(p) = 0$, $\mathtt{cost}(p) = 0$, and $\mathtt{weight}'(p) = \mathbf{0}$. Therefore, we have

$$\begin{aligned} \mathtt{weight}(\varepsilon)(l) \;&:=\; \begin{cases} \infty & \text{if } \; l < \mathtt{length}(\varepsilon) \\ \mathtt{cost}(\varepsilon) & \text{if } \; l \geq \mathtt{length}(\varepsilon) \end{cases} \\ &=\; \begin{cases} \infty & \text{if } \; l < 0 \\ 0 & \text{if } \; l \geq 0 \end{cases} \\ &=\; \mathbf{0}(l) \\ &=\; \mathtt{weight}'(\varepsilon)(l). \end{aligned}$$

**Step case:** $p = q\,e$. If $\mathtt{weight}(e) = \langle x, y\rangle$, i.e. $\mathtt{length}(e) = x$ and $\mathtt{cost}(e) = y$, then $\mathtt{cost}(p) = \mathtt{cost}(q) + y$ and $\mathtt{length}(p) = \mathtt{length}(q) + x$. Since $\mathtt{weight}'(p) = \mathtt{weight}'(q) + \langle x, y\rangle$ we have

$$\mathtt{weight}'(p)(l) \;=\; \begin{cases} \infty & \text{if } \; l < x; \\ \mathtt{weight}'(q)(l-x) + y & \text{if } \; l \geq x. \end{cases}$$

By induction hypothesis we have $\mathtt{weight}'(q) = \mathtt{weight}(q)$, yielding

$$\mathtt{weight}'(p)(l) \;=\; \begin{cases} \infty & \text{if } l < x; \\ \mathtt{weight}(q)(l-x)+y & \text{if } l \geq x. \end{cases}$$

Substituting the definition of $\mathtt{weight}(q)$ we arrive at

$$\mathtt{weight}'(p)(l) \;=\; \begin{cases} \infty & \text{if } l < x; \\ \infty & \text{if } l - x < \mathtt{length}(q) \wedge l \geq x; \\ \mathtt{cost}(q)+y & \text{if } l - x \geq \mathtt{length}(q) \wedge l \geq x. \end{cases}$$

However, since $l - x < \mathtt{length}(q)$ is equivalent to $l < \mathtt{length}(q) + x = \mathtt{length}(p)$ and, furthermore, $\mathtt{cost}(p) = \mathtt{cost}(q) + y$, we have found that

$$\begin{aligned} \mathtt{weight}'(p)(l) \;&=\; \begin{cases} \infty & \text{if } l < \mathtt{length}(p) \\ \mathtt{cost}(p) & \text{if } l \geq \mathtt{length}(p) \end{cases} \\ &=\; \mathtt{weight}(p)(l). \end{aligned}$$

□

Using $\mathtt{weight}$, we can give a different formulation of the *constrained shortest path problem* that resembles the formulation of the *shortest path problem* more closely. However, there is one important difference: In contrast to the set $\mathbb{N}$ of natural numbers, the function space **SF** is not linearly ordered. Therefore, in general the minimum of two functions $f_1, f_2 \in \mathbf{SF}$ does not exist. Instead, the *greatest lower bound* ($\mathtt{glb}$) has to be used. It is defined as the pointwise minimum of $f_1$ and $f_2$, i.e. $\mathtt{glb}(f_1, f_2)(l) = \mathtt{min}\big(f_1(l), f_2(l)\big)$. Then the *constrained shortest path problem* can be reformulated as shown by the following theorem.

**Theorem 9**

$$\mathtt{csp}(v) = \mathtt{glb}\big\{\mathtt{weight}(p) \;:\; p \in \mathbf{Paths}(\mathtt{source}, v)\big\}.$$

**Proof**: For all $l \in \mathbb{N}$ we have the following chain of equations:

$$\begin{aligned} \mathtt{csp}(v)(l) \;&=\; \mathtt{min}\big\{\mathtt{cost}(p) \;:\; p \in \mathbf{Paths}(\mathtt{source}, v) \wedge \mathtt{length}(p) \leq l\big\} \\ &=\; \mathtt{min}\big\{\mathtt{weight}(p)(l) \;:\; p \in \mathbf{Paths}(\mathtt{source}, v) \wedge \mathtt{length}(p) \leq l\big\} \\ &\quad\;\; \text{(by the definition of } \mathtt{weight} : \mathbf{Paths} \to \mathbf{SF}\text{)} \\ &=\; \mathtt{min}\big\{\mathtt{weight}(p)(l) \;:\; p \in \mathbf{Paths}(\mathtt{source}, v)\big\} \\ &\quad\;\; \text{(since } \mathtt{weight}(p)(l) = \infty \text{ if } l < \mathtt{length}(p)\text{)} \\ &=\; \mathtt{glb}\big\{\mathtt{weight}(p) \;:\; p \in \mathbf{Paths}(\mathtt{source}, v)\big\}(l) \\ &\quad\;\; \text{(since } \mathtt{glb} \text{ is defined pointwise)} \end{aligned}$$

□

## 3 A Generic Algorithm for Solving Shortest Path Problems

In this section we present an algorithm that is able to solve generalized shortest path problems. The algorithm is *generic*. This is achieved by working with an *abstract data type*. The benefit of this approach is twofold. Firstly, it is quite universal. The algorithm presented in this section can be used to solve the classical shortest path problem as well as the constrained shortest path problem. Secondly, this approach simplifies the development of the algorithm: By separating the implementation of the abstract data type from the implementation of the generic algorithm to solve the shortest path problem we have effectively split our original problem into smaller problems that are easier to solve.

Proceeding in this spirit, we generalize the notion of a weighted graph given in the last section by working with an abstract version of the function $\mathtt{weight}$, i.e. we do no longer assume that the result of $\mathtt{weight}$ is a pair of positive natural numbers. Rather, the signature of `weight` is now given as

$$\texttt{weight} : \mathbf{Edges} \to \mathbb{W},$$

where the set $\mathbb{W}$ is a set of abstractly given *weights*. Furthermore, we assume that there is a set $\mathbb{M}$ the elements of which are called *measures*. This set is characterized via the following axioms:

1. $\mathbb{M}$ is partially ordered via a well–founded relation $\prec$ with
   (a) a largest element $\infty$,
   (b) a smallest element $0$, and
   (c) for any two elements $m_1, m_2 \in \mathbb{M}$ the *greatest lower bound* $\texttt{glb}(m_1, m_2)$ exists. It is characterized by
      i. $\texttt{glb}(m_1, m_2) \preceq m_1 \ \wedge\ \texttt{glb}(m_1, m_2) \preceq m_2$,
      ii. $m_3 \preceq m_1 \ \wedge\ m_3 \preceq m_2 \ \Rightarrow\ m_3 \preceq \texttt{glb}(m_1, m_2)$.
2. There is a function $+ : \mathbb{M} \times \mathbb{W} \to \mathbb{M}$ for "adding" the weight of an edge $e$ to the measure of a path when this path is extended by $e$. This function has the following properties:
   (a) $+$ is *monotone*, i.e. $m_1 \prec m_2 \ \Rightarrow\ m_1 + w \prec m_2 + w$.
   (b) `glb` is *distributive* with respect to $+$, i.e.
   $$\texttt{glb}(m_1 + w, m_2 + w) = \texttt{glb}(m_1, m_2) + w.$$

The function $+$ is used to generalize the function $\texttt{weight} : \mathbf{Edges} \to \mathbb{W}$ to a function with signature $\texttt{weight} : \mathbf{Paths} \to \mathbb{M}$ by induction:

1. $\texttt{weight}(\varepsilon) \ := \ 0$.
2. $\texttt{weight}(p\,e) \ := \ \texttt{weight}(p) + \texttt{weight}(e)$.

Since $\prec$ is well–founded, the function `glb` can be extended to countable sets.

**Theorem 10** *If $M = \{m_i \,:\, i \in \mathbb{N}\}$ is a countable set, then the* greatest lower bound *of $M$ exists.*

**Proof**: We define a sequence $(g_i)_{i \in \mathbb{N}}$ by induction.

1. $g_0 \ := \ m_0$.
2. $g_{i+1} \ := \ \texttt{glb}(g_i, m_{i+1})$.

We have $g_{i+1} \preceq g_i$ for all $i \in \mathbb{N}$. Since $\prec$ is well–founded, there exists a $k \in \mathbb{N}$ such that $g_i = g_k$ for all $i \geq k$. It is straightforward to see that $g_k$ is the greatest lower bound of $M$. □

### 3.1 The Algorithm

We assume to be given a distinguished node `source`. Our goal is to compute the function $\texttt{min\_weight} : \mathbf{Nodes} \to \mathbb{M}$ defined as

$$\texttt{min\_weight}(v) \ := \ \texttt{glb}\big\{\texttt{weight}(p) \,:\, p \in \mathbf{Paths}(\texttt{source}, v)\big\}.$$

Since the set of all paths from `source` to $v$ is at most countable, Theorem 10 shows that the above greatest lower bound exists. (Theorem 10 is needed because, although there are only finitely many nodes, the set $\mathbf{Paths}(\texttt{source}, v)$ need not be finite. After all, there may exist paths containing cycles!)

In order to compute the function `min_weight` the idea is to define a function

$$\texttt{label} : \mathbf{Nodes} \rightarrow \mathbb{M}$$

assigning a label to every node such that this label is an approximation from above of $\texttt{min\_weight}(v)$, i.e. we always have $\texttt{min\_weight}(v) \preceq \texttt{label}(v)$. This function is successively improved until no further improvement is possible. Therefore, the algorithm proceeds as follows:

1. Initially, the only node $v$ that is known to be connected to the `source` is the source itself. Therefore we label `source` with 0 and all nodes different from `source` are labeled with $\infty$.
2. Then, the following step is repeated as long as possible: We look for an edge $e = \langle v, w \rangle$ such that $\texttt{label}(w) \not\preceq \texttt{label}(v) + \texttt{weight}(e)$. If we are able to find an edge $e$ with this property, then we relabel $w$ with the new label $\texttt{glb}\big(\texttt{label}(w), \texttt{label}(v) + \texttt{weight}(e)\big)$. Otherwise, the algorithm terminates.

Note that in the second step the label of a node can only be decreased since we always have

$$\texttt{glb}\big(\texttt{label}(w), \texttt{label}(v) + \texttt{weight}(e)\big) \preceq \texttt{label}(w).$$

The above inequality is strict iff $\texttt{label}(w) \not\preceq \texttt{label}(v) + \texttt{weight}(e)$.

**asm** `shortest_path1(` $\texttt{label} : \mathbf{Nodes} \rightarrow \mathbb{M}$; $\texttt{source} : \mathbf{Nodes}$ `)`
**initialization**
  $\forall x \in \mathbf{Nodes} \backslash \{\texttt{source}\} : \texttt{label}(x) := \infty$
  `label(source) := 0`
**transition** *relabeling*
  **choose** $e = \langle v, w \rangle$ **in Edges**
   **satisfying** $\texttt{label}(w) \not\preceq \texttt{label}(v) + \texttt{weight}(e)$
  $\texttt{label}(w)$ `:=` $\texttt{glb}(\,\texttt{label}(w),\, \texttt{label}(v) + \texttt{weight}(e)\,)$

**Figure 1:** The ASM `shortest_path1`.

A formalization is given by the ASM `shortest_path1` in Figure 1. The computation of this ASM stops as soon as we have $\texttt{label}(w) \preceq \texttt{label}(v) + \texttt{weight}(e)$ for every edge $e = \langle v, w \rangle$.

## 4 Correctness of `shortest_path1`

In this section we show the correctness of the ASMs `shortest_path1`.

**Lemma 11** *For any node $v$ and any moment in the computation of the ASM* `shortest_path1` *the following invariant holds:*

$$\texttt{min\_weight}(v) \preceq \texttt{label}(v).$$

**Proof**: The proof is done by induction on the computation of the ASM. In order to show the claim to be true after the *initialization*, we have to deal with two cases.

1. $v = \mathtt{source}$. We have $\varepsilon \in \mathbf{Paths}(\mathtt{source}, v)$ and $\mathtt{weight}(\varepsilon) = 0$, showing $\mathtt{min\_weight}(v) = 0$. Since $0 \preceq m$ for all $m \in \mathbb{M}$ this proves the claim.
2. $v \neq \mathtt{source}$. Then we have $\mathtt{label}(v) = \infty$ and since $m \preceq \infty$ for all $m \in \mathbb{M}$ the claim is obvious.

For the induction step assume the edge $e = \langle v, w \rangle$ to be chosen. Then $\mathtt{label}(w)$ is updated to the value

$$\mathtt{glb}\big(\,\mathtt{label}(w),\, \mathtt{label}(v) + \mathtt{weight}(e)\,\big).$$

Therefore we have to show that

$$\mathtt{min\_weight}(w) \preceq \mathtt{glb}\big(\,\mathtt{label}(w),\, \mathtt{label}(v) + \mathtt{weight}(e)\,\big)$$

holds. This is equivalent to the conjunction of (1) and (2) below:

$$\mathtt{min\_weight}(w) \preceq \mathtt{label}(w) \qquad (1)$$

$$\mathtt{min\_weight}(w) \preceq \mathtt{label}(v) + \mathtt{weight}(e). \qquad (2)$$

(1) is true by the induction hypothesis stated for $w$ and (2) follows from

$$\mathtt{min\_weight}(w) \preceq \mathtt{min\_weight}(v) + \mathtt{weight}(e) \qquad (3)$$

and the induction hypothesis for $v$. In order to prove (3), we note that

$$\begin{aligned} &\big\{\mathtt{weight}(p\,e) \,:\, p \in \mathbf{Paths}(\mathtt{source}, v)\big\} \\ \subseteq\ &\big\{\mathtt{weight}(q) \,:\, q \in \mathbf{Paths}(\mathtt{source}, w)\big\} \end{aligned}$$

holds, since $p \in \mathbf{Paths}(\mathtt{source}, v)$ implies $p\,e \in \mathbf{Paths}(\mathtt{source}, w)$. Therefore,

$$\begin{aligned} &\mathtt{glb}\big\{\mathtt{weight}(q) \,:\, q \in \mathbf{Paths}(\mathtt{source}, w)\big\} \\ \preceq\ &\mathtt{glb}\big\{\mathtt{weight}(p\,e) \,:\, p \in \mathbf{Paths}(\mathtt{source}, v)\big\}. \end{aligned}$$

Since $\mathtt{weight}(p\,e) = \mathtt{weight}(p) + \mathtt{weight}(e)$ and $\mathtt{glb}$ is distributive with respect to $+$, (3) follows from the definition of $\mathtt{min\_weight}$. $\Box$

While the last lemma showed that $\mathtt{label}(v)$ is never too small, the next lemma shows that, once the algorithm has terminated, $\mathtt{label}(x)$ is not too big either.

**Lemma 12** *If the algorithm has terminated, then for any node $w$ and any path $q \in \mathbf{Paths}(\mathtt{source}, w)$ the following holds:*

$$\mathtt{label}(w) \preceq \mathtt{weight}(q).$$

**Proof**: The proof is by induction on the number $n$ of edges of $q$.

**Base Case:** $n = 0$ and therefore $q = \varepsilon$ is the empty path. Then $w = \mathtt{source}$ and the claim is trivial since, initially, $\mathtt{label}(\mathtt{source}) = 0$ and subsequent *relabeling* can only decrease the value of the function $\mathtt{label}$.

**Step Case:** The path has the form $q = p\,e$. If $e = \langle v, w \rangle$, then $p$ is a path connecting $\mathtt{source}$ to $v$, i.e. $p \in \mathbf{Paths}(\mathtt{source}, v)$. By induction hypothesis $\mathtt{label}(v) \preceq \mathtt{weight}(p)$. Using the monotonicity of $+$ this gives

$$\mathtt{label}(v) + \mathtt{weight}(e) \preceq \mathtt{weight}(p) + \mathtt{weight}(e). \qquad (1)$$

If the algorithm has terminated, then

$$\mathtt{label}(w) \preceq \mathtt{label}(v) + \mathtt{weight}(e), \qquad (2)$$

since otherwise the *choose construct* would be able to choose the edge $e$ and the algorithm would not have been terminated. Taken together, (1) and (2) yield

$$\mathtt{label}(w) \preceq \mathtt{weight}(p) + \mathtt{weight}(e) = \mathtt{weight}(p\,e) = \mathtt{weight}(q). \ \Box$$

**Theorem 13 (partial correctness)** *When the algorithm terminates, then we have* $\texttt{label}(v) = \texttt{min\_weight}(v)$ *for any node* $v$.

**Proof**: This is an immediate consequence of Lemma 11 and Lemma 12. □

**Theorem 14 (termination)** *Every computation of the ASM* `shortest_path1` *terminates.*

**Proof**: Every time the *relabeling* rule is applied, the label for one node $w$ strictly decreases and the other labels remain unchanged. Since there are only finitely many nodes and the relation $\prec$ is well–founded, this rule can be applied only a finite number of times. □

## 5 Refining the ASM `shortest_path1`

We refine the ASM `shortest_path1` to `shortest_path2` by eliminating the non-deterministic choice of edges. The method that we use is known in the literature as the *labeling and scanning method*, cf. [Tarjan 1983]. It is essentially a book-keeping method that works by maintaining a set $\mathcal{S}$ of nodes that still need to be *scanned*, where *scanning* a node $v$ is carried out in three steps:

1. The set $\mathcal{E}$ of all edges originating in $v$ is computed. For this purpose, we use the function `adjacent` defined as
   $$\texttt{adjacent}(v) = \big\{e \in \textbf{Edges} : \texttt{tail}(e) = v\big\}.$$
2. For every edge $e = \langle v, w\rangle$ in $\mathcal{E}$ such that
   $$\texttt{label}(w) \not\preceq \texttt{label}(v) + \texttt{weight}(e)$$
   we relabel $w$ with $\texttt{glb}\big(\texttt{label}(w), \texttt{label}(v) + \texttt{weight}(e)\big)$. Then $w$ is added to the set $\mathcal{S}$ of nodes that still need to be scanned, since it might then be possible to lower the labeling of nodes reachable via edges originating in $w$.
3. We delete $v$ from the set $\mathcal{S}$.

This is formalized by the ASM `shortest_path2` given in Figure 2. Note that the computation of this ASM terminates iff in the *scanning* rule the set $\mathcal{S}$ becomes empty, since then the *choose* construct is unable to produce a node $v \in \mathcal{S}$.

### 5.1 Correctness of the Refinement

**Theorem 15** *The ASM* `shortest_path2` *computes the same value for the function* `label` *as the ASM* `shortest_path1`.

**Proof**: We show that every computation of `shortest_path2` can be regarded as a computation of `shortest_path1`. Indeed, if we abstract from the variables $\mathcal{S}$, $\mathcal{E}$, and `mode`, then

- the initialization of `shortest_path2` is the same as that of `shortest_path1`,
- the transition rules "*scanning*" and "*back to scanning*" have no effect, and
- the transition rule "*relabeling*" has the same effect on the function `label` for both ASMs.

```
asm shortest_path2( label : Nodes → 𝕄; source : Nodes )

initialization
            ∀x ∈ Nodes\{source} : label(x) := ∞
            label(source) := 0
            𝒮 := {source}
            mode := scan

transition scanning
     if     mode = scan
     then   choose v in Nodes satisfying v ∈ 𝒮
            ℰ := adjacent(v)
            𝒮 := 𝒮 − {v}
            mode := relabel

transition relabeling
     if     mode = relabel
        &   ℰ ≠ ∅
     then   choose e = ⟨v, w⟩ in Edges satisfying e ∈ ℰ
            ℰ := ℰ − {e}
            if      label(w) ≰ label(v) + weight(e)
            then    label(w) := glb( label(w), label(v) + weight(e) )
                    𝒮 := 𝒮 ∪ {w}

transition back to scanning
     if     mode = relabel
        &   ℰ = ∅
     then   mode := scan.
```

**Figure 2:** The ASM `shortest_path2`.

We have seen already that the *relabeling* rule can decrease the value of the function `label` only a finite number of times, but the ASM `shortest_path2` can execute this rule without changing the value of the function `label`. Therefore we have to exclude the possibility of infinite computations of `shortest_path2`. To this end, assume that $(s_n)_n$ is a sequence of states resulting from a computation of `shortest_path2`. Then there must be a time $t$ such that for all $n \geq t$ the value of the function `label` remains unchanged in the transition from $s_n$ to $s_{n+1}$. But from that time on the set $\mathcal{S}$ can never be increased, since $\mathcal{S}$ is only increased when the function `label` is decreased. The computations of `shortest_path2` consist of cycles described by the following regular expression:

$$\text{“}\textit{scanning}\text{” “}\textit{relabeling}\text{”}^{*} \text{ “}\textit{back to scanning}\text{”}.$$

Every *relabeling* step decreases the size of $\mathcal{E}$, therefore after a finite number of *relabeling* steps, $\mathcal{E}$ will be empty and the transition *back to scanning* is executed. This is followed by a *scanning* step that decreases the size of $\mathcal{S}$. Therefore, termination of `shortest_path2` is guaranteed.

Furthermore, we have to exclude the possibility that the computation of the ASM `shortest_path2` terminates prematurely. This could happen if $\mathcal{S} = \emptyset$ in the scanning step although there is an edge $e = \langle v, w \rangle$ such that

$$\texttt{label}(w) \not\preceq \texttt{label}(v) + \texttt{weight}(e). \qquad (*)$$

Define $t_0$ as the earliest time such that the above inequality holds from $t_0$ on up to the termination of the ASM. We first show that then $v \in \mathcal{S}$ must hold at time $t_0$. To prove this claim, we need a case distinction:

1. Case: $t_0 = 0$. Since at $t_0$ the only node satisfying $\texttt{label}(v) \neq \infty$ is $v = \texttt{source}$, we have $v = \texttt{source}$ and therefore $v \in \mathcal{S}$ at $t_0$.
2. Case: $t_0 > 0$. Then the *relabeling* rule must have decreased the value of $\texttt{label}(v)$ at time $t_0 - 1$. (After all, the only way for the inequality $\texttt{label}(w) \preceq \texttt{label}(v) + \texttt{weight}(e)$ to become false is when $\texttt{label}(v)$ is decreased. A decrease of of $\texttt{label}(w)$ would surely leave this inequality valid.) But if $\texttt{label}(v)$ has been decreased at time $t_0 - 1$, then $v$ must have been added to $\mathcal{S}$ at $t_0 - 1$ and we have $v \in \mathcal{S}$ at $t_0$.

Since $v \in \mathcal{S}$ at $t_0$ and the ASM terminates only when $\mathcal{S}$ is empty, there must be a time $t_1 > t_0$ such that $v$ is selected in the *scanning* step. Because in the following sequence of *relabeling* steps all edges leaving $v$ are eventually dealt with, there is a time $t_2 > t_1$ such that the edge $e = \langle v, w \rangle$ is selected at this time by the *relabeling* rule. Since then the update

$$\texttt{label}(w) \texttt{ := } \texttt{glb}(\, \texttt{label}(w), \texttt{label}(v) + \texttt{weight}(e)\,)$$

is executed, we have $\texttt{label}(w) = \texttt{glb}(\, \texttt{label}(w), \texttt{label}(v) + \texttt{weight}(e)\,)$ at time $t_2 + 1$, contradicting $(*)$. □

### 5.2 Moore's Algorithm

The algorithm presented in subsection 5.1 above still contains non-determinism since it makes use of the *choose* construct. We eliminate this non-determinism in this subsection. In order to do this we have to decide how the sets $\mathcal{S}$ and $\mathcal{E}$ should be represented. We implement $\mathcal{S}$ as a queue, while $\mathcal{E}$ is implemented as a stack. Then, we arrive at the following implementation shown in Figure 3, where we make use of some functions working on queues and stacks that are specified below:

- $\texttt{Queue} : T \to \texttt{Queue}(T)$

  This function is a constructor of the polymorphic data type `Queue`. If $v$ is a node, then $\texttt{Queue}(v)$ is a queue containing precisely the node $v$.
- $\texttt{empty} : \texttt{Queue}(T) \to \texttt{bool}$

  The call $\texttt{empty}(Q)$ evaluates to true iff the queue $Q$ is empty.
- $\texttt{head} : \texttt{Queue}(T) \to T$

  If $Q$ is not empty, then $\texttt{head}(Q)$ returns the first element of $Q$.
- $\in : T \times \texttt{Queue}(T) \to \texttt{bool}$

  The call $t \in Q$ yields true iff $t$ is an element of the queue $Q$.
- $\texttt{append} : \texttt{Queue}(T) \times T$

  The call $\texttt{append}(Q, t)$ appends the element $t$ at the end of the queue $Q$.

Furthermore, we use a bit of PROLOG notation in Figure 3: `[]` denotes the empty stack, while `[X|Xs]` denotes a non-empty stack with top element `X` and tail `Xs`, i.e. we have that `Xs` is the result of removing `X` from the stack `[X|Xs]`.

It should be noted that the *choose* construct used with `shortest_path2` is replaced by appropriately strengthening the guards of the corresponding transition rules in `shortest_path3`. Note further that the ASM `shortest_path3` terminates when the queue $\mathcal{S}$ is empty and `mode = scan` since then no rule is applicable.

**asm** `shortest_path3(` `label` : **Nodes** $\to \mathbb{M}$; `source` : **Nodes** `)`

**initialization**
$\forall x \in$ **Nodes**$\backslash\{$`source`$\}$ : `label`$(x) := \infty$,
`label(source) :=` $0$,
$\mathcal{S}$ `:= Queue(source)`,
`mode := scan`.

**transition** *scanning*
**if** `mode = scan`
**&** $\neg$ `empty`$(\mathcal{S})$
**then** $\mathcal{E}$ `:= adjacent(head(`$\mathcal{S}$`))`
$\mathcal{S}$ `:=` $\mathcal{S} - \{$`head`$(\mathcal{S})\}$
`mode := relabel`

**transition** *relabeling*
**if** `mode = relabel`
**&** $\mathcal{E} = [\,\langle v, w\rangle \mid \mathcal{E}'\,]$
**then** $\mathcal{E}$ `:=` $\mathcal{E}'$
**if** `label`$(w) \not\leq$ `label`$(v) +$ `weight`$(e)$
**then** `label`$(w)$ `:= glb(` `label`$(w)$, `label`$(v) +$ `weight`$(e)$ `)`
**if** $w \notin \mathcal{S}$
**then** $\mathcal{S}$ `:= append(`$\mathcal{S}, w$`)`

**transition** *back to scanning*
**if** `mode = relabel`
**&** $\mathcal{E} = []$
**then** `mode := scan.`

**Figure 3:** The ASM `shortest_path3`.

When comparing the ASMs `shortest_path2` and `shortest_path3`, there is a subtle difference to note: The update $\mathcal{S}$ `:= append(`$\mathcal{S}, w$`)` in the ASM `shortest_path3` is guarded by the condition $w \notin \mathcal{S}$, while the corresponding update $\mathcal{S}$ `:=` $\mathcal{S} \cup \{w\}$ in `shortest_path2` is not subject to a similar condition. This guard takes care of maintaining the invariant that the queue $\mathcal{S}$ contains every element at most once. This invariant is necessary since, without this precaution, a queue represents a multiset rather than a set. We skip the (standard) proof that this queue implementation is correct. (The sceptical reader may look at the more elaborate proof given for the implementation of the slightly more complex queue concept in the Transputer using ASMs in [Börger and Durdanovic 1996].)

## 6 Instantiation of the ASM `shortest_path3`

Up to now the ASMs we have presented are generic since they all contain the abstract data type $\mathbb{M}$. In this section we instantiate $\mathbb{M}$ with concrete data types and thereby solve both the *shortest path problem* and the *constrained shortest path problem*.

### 6.1 Solving the Shortest Path Problem

To solve the *shortest path problem*, we instantiate $\mathbb{W}$ with $\mathbb{N}$ and $\mathbb{M}$ with the set $\mathbb{N} \cup \infty$. Then $\prec$ is interpreted as the usual ordering $<$ on natural numbers and $\mathtt{glb}(x, y)$ is interpreted as the minimum $\mathtt{min}(x, y)$. Finally, $x + y$ is interpreted as the sum of $x$ and $y$ if $x$ is a natural number and as $\infty$ if $x$ equals $\infty$. It is trivial to verify that the conditions postulated in Section 3 are satisfied by this instantiation. Using this instantiation we obviously have

$$\mathtt{min\_weight}(v) = \mathtt{sp}(v),$$

showing that the ASM `shortest_path3` solves the *shortest path problem.*

### 6.2 Solving the Constrained Shortest Path Problem

To solve the *constrained shortest path problem*, we instantiate $\mathbb{W}$ with $\mathbb{N} \times \mathbb{N}$ and $\mathbb{M}$ with the the function space **SF**. The ordering $\prec$ is defined pointwise, i.e.

$$f \preceq g \overset{\text{def}}{\Longleftrightarrow} \forall t \in \mathbb{N} : f(t) \leq g(t)$$

and $f \prec g \overset{\text{def}}{\Longleftrightarrow} f \preceq g \wedge f \neq g$. The greatest lower bound $\mathtt{glb}(f_1, f_2)$ is defined as the pointwise minimum of $f_1$ and $f_2$ and addition has already been defined in Section 2.

Using these definitions it is straightforward to verify that the specification of the abstract data type $\mathbb{M}$ given in Section 3 is satisfied. The only requirement that is not trivial to check is the well–foundedness of $\prec$. For technical reasons, we defer the proof of this property to the next subsection.

It is easy to see that, with the instantiations given above, we have

$$\mathtt{min\_weight}(v) = \mathtt{glb}\big\{\mathtt{weight}(p) \,:\, p \in \mathbf{Paths}(\mathtt{source}, v)\big\}.$$

Therefore, Theorem 9 shows that the ASM `shortest_path3` solves the *constrained shortest path problem.*

### 6.3 Representation of SF

In order to make the instantiation $\mathbb{M} \mapsto \mathbf{SF}$ work, we have to implement the operations $\prec$, `glb`, and $+$ for functions from **SF**. To this end, we have to choose a representation for the set **SF**.

**Definition 16 ($\mathbf{SF}'$)** *The set of representations* $\mathbf{SF}'$ *is defined as the set of all lists of pairs of natural numbers of the form*

$$\big[\,\langle x_1, y_1\rangle,\ \cdots,\ \langle x_n, y_n\rangle\,\big]$$

*satisfying the following* representation invariant*:*

1. $x_i, y_i \in \mathbb{N}$ *for all* $i = 1, \cdots, n$,
2. $x_i < x_{i+1}$ *for all* $i = 1, \cdots, n-1$,
3. $y_i > y_{i+1}$ *for all* $i = 1, \cdots, n-1$.

**Definition 17 ($[\![f]\!]$)** *If* $f = \big[\,\langle x_1, y_1\rangle,\ \cdots,\ \langle x_n, y_n\rangle\,\big] \in \mathbf{SF}'$, *then the function* $[\![f]\!] \in \mathbf{SF}$ *represented by* $f$ *is defined as follows:*

$$[\![f]\!](t) \ := \ \begin{cases} \infty & \text{if } \ t < x_1; \\ y_1 & \text{if } \ x_1 \le t < x_2; \\ \vdots & \\ y_{n-1} & \text{if } \ x_{n-1} \le t < x_n; \\ y_n & \text{if } \ x_n \le t. \end{cases}$$

□

Equivalently, $[\![f]\!]$ could be defined via:

$$[\![f]\!](t) = \mathtt{min}\big(\{y_i \,:\, x_i \le t,\ i = 1, \cdots, n\}\big).$$

We have $[\![[]]\!] = \infty$ and $[\![[\langle 0, 0\rangle]]\!] = \mathbf{0}$.

**Lemma 18** $\prec$ *is well–founded.*

**Proof**: To every $f = \big[\,\langle x_1, y_1\rangle,\ \cdots,\ \langle x_n, y_n\rangle\,\big] \in \mathbf{SF}$ we assign an ordinal $o(f)$:

$$o(f) \ := \ \omega * (x_1 + y_n) + \sum_{i=1}^{n-1} (x_{i+1} - x_i) * y_i.$$

We show that $f' \prec f$ implies $o(f') < o(f)$. Assume $f = [\langle x_1, y_1\rangle, \cdots, \langle x_n, y_n\rangle]$ and $f' = [\langle x'_1, y'_1\rangle, \cdots, \langle x'_{n'}, y'_{n'}\rangle]$. Then $x'_1 \le x_1$ and $y'_{n'} \le y_n$. If either of these inequations is strict, then obviously $o(f') < o(f)$. Assume therefore $x_1 = x'_1$ and $y_n = y'_{n'}$. We have

$$\sum_{i=1}^{n-1} (x_{i+1} - x_i) * y_i = \int_{x_1}^{x_n} f(t)\,\mathrm{d}t$$

and a similar equation holds for $f'$. As we have assumed $x_1 = x'_1$ and $y_n = y'_{n'}$, the assumption $f' \prec f$ implies

$$\int_{x_1}^{x_n} f'(t)\,\mathrm{d}t < \int_{x_1}^{x_n} f(t)\,\mathrm{d}t$$

and, since $x'_{n'} \le x_n$, we conclude $o(f') < o(f)$. □

We need to define an auxiliary function `merge` in order to compute the greatest lower bound of two functions from $\mathbf{SF}$.

**Definition 19** *For* $f_1, f_2 \in \mathbf{SF}'$, $\mathtt{merge}(f_1, f_2)$ *is defined recursively:*

1. $\mathtt{merge}([], f) = f$.
2. $\mathtt{merge}(f, []) = f$.

3. *If* $x_1 \leq x_2$, *then* $\texttt{merge}\big([\,\langle x_1, y_1\rangle \mid f_1\,], [\,\langle x_2, y_2\rangle \mid f_2\,]\big)$ *equals*
$$[\,\langle x_1, y_1\rangle \mid \texttt{merge}\big(f_1, [\,\langle x_2, y_2\rangle \mid f_2\,]\big)\,]$$
4. *If* $x_1 > x_2$, *then* $\texttt{merge}\big([\,\langle x_1, y_1\rangle \mid f_1\,], [\,\langle x_2, y_2\rangle \mid f_2\,]\big)$ *equals*
$$= \ [\,\langle x_2, y_2\rangle \mid \texttt{merge}\big([\,\langle x_1, y_1\rangle \mid f_1\,], f_2\big)\,] \qquad \Box$$

Note that $\texttt{merge}(f_1, f_2)$ is, in general, not an element of $\mathbf{SF}'$ since the representation invariant is not maintained. For example, we have

$$\texttt{merge}\big([\langle 1,1\rangle], [\langle 2,2\rangle]\big) = \big[\langle 1,1\rangle, \langle 2,2\rangle\big]$$

We need a function `contract` that reestablishes the representation invariant.

**Definition 20 (**`contract`**)** *The value* $\texttt{contract}(f)$ *is defined by induction on the length of the list* $f$*:*

1. $\texttt{contract}([]) = []$.
2. $\texttt{contract}\big([\langle x, y\rangle]\big) = [\langle x, y\rangle]$.
3. *If* $x_1 = x_2$, *then*
$\texttt{contract}\big([\,\langle x_1, y_1\rangle, \langle x_2, y_2\rangle \mid f\,]\big) \ = \ \texttt{contract}\big([\,\langle x_1, \min(y_1, y_2)\rangle \mid f\,]\big)$.
4. *If* $x_1 \neq x_2$ *and* $y_1 > y_2$, *then*
$\texttt{contract}\big([\,\langle x_1, y_1\rangle, \langle x_2, y_2\rangle \mid f\,]\big) = [\,\langle x_1, y_1\rangle \mid \texttt{contract}\big([\,\langle x_2, y_2\rangle \mid f\,]\big)\,]$.
5. *If* $x_1 \neq x_2$ *and* $y_1 \leq y_2$, *then*
$\texttt{contract}\big([\,\langle x_1, y_1\rangle, \langle x_2, y_2\rangle \mid f\,]\big) \ = \ \texttt{contract}\big([\,\langle x_1, y_1\rangle \mid f\,]\big)$. $\Box$

**Definition 21 (**$\texttt{glb}'$**)** *For* $f_1, f_2 \in \mathbf{SF}'$ *we define*

$$\texttt{glb}'(f_1, f_2) = \texttt{contract}\big(\texttt{merge}(f_1, f_2)\big) \qquad \Box$$

The next lemma shows that the above definition of $\texttt{glb}'$ computes the greatest lower bound of the set $\mathbf{SF}$.

**Lemma 22** *If* $f_1, f_2 \in \mathbf{SF}'$, *then* $\texttt{glb}'(f_1, f_2) \in \mathbf{SF}'$. *Furthermore,*

$$[\![\texttt{glb}'(f_1, f_2)]\!] = \texttt{glb}\big([\![f_1]\!], [\![f_2]\!]\big). \qquad \Box$$

**Proof**: This lemma can be shown by a simple expansion of the definitions. $\Box$

Implementing the relation $\prec$ is now straightforward:

**Definition 23 (**$\prec$**)** $\quad f_1 \prec f_2 \ \stackrel{\text{def}}{\Longleftrightarrow} \ \texttt{glb}'(f_1, f_2) = f_1$. $\Box$

We proceed to define addition for $\mathbf{SF}'$.

**Definition 24 (**$+'$**)**

$$[\,\langle x_1, y_1\rangle, \cdots, \langle x_n, y_n\rangle\,] +' \langle x, y\rangle \ := \ [\langle x_1 + x, y_1 + y\rangle, \cdots, \langle x_n + x, y_n + y\rangle\,]. \quad \Box$$

**Lemma 25** *If* $f \in \mathbf{SF}'$ *and* $\langle x, y\rangle \in \mathbb{N} \times \mathbb{N}$, *then* $f +' \langle x, y\rangle \in \mathbf{SF}'$. *Furthermore,*

$$[\![f +' \langle x, y\rangle]\!] = [\![f]\!] + \langle x, y\rangle. \qquad \Box$$

**Proof**: This lemma can be shown by a simple expansion of the definitions. $\Box$

**Concluding Remark:** At this point the implementation of **SF** is complete and we have developed an algorithm to solve the constrained shortest path problem.

Based on the precise semantical ASM description given in [Wallace 1995] for the semantics of C++ we can transform the ASM obtained here to a C++-program and further optimize this program in the spirit of [Lauther 1996]. In this way we can prove the correctness of a highly sophisticated program that solves a non-trivial graph theoretical problem.

**Acknowledgment:** First of all, I would like to thank Egon Börger both for introducing me to his method of proving the correctness of software using abstract state machines and, furthermore, for various discussions during the case study presented here. Without these discussions, this case study would not have been completed in a satisfactory way. After having read an earlier draft of this paper, Peter Päppinghaus and Egon Börger both suggested numerous improvements.

Furthermore, I would like to thank Ulrich Lauther for introducing me to the constrained shortest path problem and for explaining his program solving this problem. Finally, I thank Bertil A. Brandin, Michael Hofmeister, Martin Müller, and Doris Tesch for helpful discussions.

This project has been sponsored from the German BMBF via the research grant no. 01 IS 519 A 9.

## References

[Börger 1995] Börger, E.: "Why Use Evolving Algebras for Hardware and Software Engineering". *In* Miroslav Bartosek, Jan Staudek, and Jiri Wiedermann, editors, *SOFSEM '95, 22nd Seminar on Current Trends in Theory and Practice of Informatics*, volume 1012 of *Lecture Notes in Computer Science*, pages 236–271, Springer, 1995.

[Börger and Durdanovic 1996] Börger, E. and Durdanovic, I.: "Correctness of Compiling Occam to Transputer Code". In *The Computer Journal*, volume 39, no. 1, pages 52–92, 1996.

[Desrosiers et al. 1995] Desrosiers, J., Dumas Y., Solomon M., and Soumis, F.: "Time Constrained Routing and Scheduling". *In* M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 4, pages 35–140. North-Holland, 1995.

[Gurevich 1993] Gurevich, Y.: "Evolving Algebras: An Attempt to Discover Semantics". *In* G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 266–292. World Scientific, 1993.

[Gurevich 1995] Gurevich, Y.: "Evolving Algebras 1993: Lipari Guide". *In* Egon Börger, editor, *Specification and Validation Methods*, pages 3–36. Oxford University Press, 1995.

[Lauther 1996] Lauther, U.: "C++ Implementation of Constrained Shortest Path Calculations". Personal communication, 1996.

[Moore 1959] Moore, E. F.: "The Shortest Path Through a Maze". In *Proc. International Symposium on the Theory of Switching, Part* II, volume 30 of *The Annals of the Computation Laboratory of Harvard University*, Cambridge, MA, 1959. Harvard University Press.

[Tarjan 1983] Tarjan, R. E.: "Data Structures and Network Algorithms." volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, PA, 1983.

[Wallace 1995] Wallace, C.: "The Semantics of the C++ Programming Language". *In* Egon Börger, editor, *Specification and Validation Methods*, pages 131–164. Oxford University Press, 1995.