# The Floyd-Warshall Algorithm for Shortest Paths 

Simon Wimmer and Peter Lammich

September 13, 2023


#### Abstract

The Floyd-Warshall algorithm [Flo62, Roy59, War62] is a classic dynamic programming algorithm to compute the length of all shortest paths between any two vertices in a graph (i.e. to solve the all-pairs shortest path problem, or APSP for short). Given a representation of the graph as a matrix of weights $M$, it computes another matrix $M^{\prime}$ which represents a graph with the same path lengths and contains the length of the shortest path between any two vertices $i$ and $j$. This is only possible if the graph does not contain any negative cycles. However, in this case the Floyd-Warshall algorithm will detect the situation by calculating a negative diagonal entry. This entry includes a formalization of the algorithm and of these key properties. The algorithm is refined to an efficient imperative version using the Imperative Refinement Framework.


## Contents

1 Floyd-Warshall Algorithm for the All-Pairs Shortest Paths Problem ..... 2
1.1 Introduction ..... 2
1.2 Preliminaries ..... 3
1.3 Definition of the Algorithm ..... 6
1.4 Result Under The Absence of Negative Cycles ..... 9
1.5 Definition of Shortest Paths ..... 12
1.6 Intermezzo: Equivalent Characterizations of Cycle-Freeness ..... 15
1.7 Result Under the Presence of Negative Cycles ..... 17
1.8 More on Canonical Matrices ..... 18
1.9 Additional Theorems ..... 19
1.10 Refinement to Efficient Imperative Code ..... 23

```
theory Floyd-Warshall
    imports Main
begin
```


## 1 Floyd-Warshall Algorithm for the All-Pairs Shortest Paths Problem

### 1.1 Introduction

The Floyd-Warshall algorithm [Flo62, Roy59, War62] is a classic dynamic programming algorithm to compute the length of all shortest paths between any two vertices in a graph (i.e. to solve the all-pairs shortest path problem, or $A P S P$ for short). Given a representation of the graph as a matrix of weights $M$, it computes another matrix $M^{\prime}$ which represents a graph with the same path lengths and contains the length of the shortest path between any two vertices $i$ and $j$. This is only possible if the graph does not contain any negative cycles (then the length of the shortest path is $-\infty$ ). However, in this case the Floyd-Warshall algorithm will detect the situation by calculating a negative diagonal entry corresponding to the negative cycle. In the following, we present a formalization of the algorithm and of the aforementioned key properties.
Abstractly, the algorithm corresponds to the following imperative pseudocode:

```
for \(k=1\).. \(n\) do
    for \(\mathrm{i}=1\).. n do
        for \(j=1\).. \(n\) do
            \(m[i, j]:=\min (m[i, j], m[i, k]+m[k, j])\)
```

However, we will carry out the whole formalization on a recursive version of the algorithm, and refine it to an efficient imperative version corresponding to the above pseudo-code in the end. The main observation underlying the algorithm is that the shortest path from $i$ to $j$ which only uses intermediate vertices from the set $\{0 \ldots k+1\}$, is: either the shortest path from $i$ to $j$ using intermediate vertices from the set $\{0 \ldots k\}$; or a combination of the shortest path from $i$ to $k$ and the shortest path from $k$ to $j$, each of them only using intermediate vertices from $\{0 \ldots k\}$. Our presentation we be slightly more general than the typical textbook version, in that we will factor our the inner two loops as a separate algorithm and show that it has similar properties as the full algorithm for a single intermediate vertex $k$.

## 1．2 Preliminaries

## 1．2．1 Cycles in Lists

abbreviation cnt $x$ xs $\equiv$ length $($ filter $(\lambda y . x=y) x s)$
fun remove－cycles ：：＇a list $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a$ list $\Rightarrow{ }^{\prime} a$ list
where

```
remove-cycles [] - acc \(=\) rev acc |
remove-cycles \((x \# x s)\) y acc \(=\)
    (if \(x=y\) then remove-cycles xs \(y[x]\) else remove-cycles xs \(y\) ( \(x \# a c c\) ))
```

lemma cnt-rev: cnt $x($ rev xs $)=c n t x x s\langle p r o o f\rangle$
value as @ $[x]$ @ $b s$ @ $[x]$ @ $c s @[x]$ @ $d s$
lemma remove－cycles－removes：cnt $x$（remove－cycles $x s x y s) \leq \max 1$（cnt $x$ ys）
$\langle$ proof〉
lemma remove－cycles－id：$x \notin$ set $x s \Longrightarrow$ remove－cycles $x s x y s=r e v y s @$ xs $\langle p r o o f\rangle$
lemma remove－cycles－cnt－id：
$x \neq y \Longrightarrow$ cnt $y$（remove－cycles $x s x y s) \leq$ cnt $y$ ys + cnt $y x s$ $\langle$ proof〉
lemma remove－cycles－ends－cycle：remove－cycles xs $x$ ys $\neq$ rev $y s @ x s \Longrightarrow$ $x \in$ set $x s$
$\langle$ proof $\rangle$
lemma remove－cycles－begins－with：$x \in$ set $x s \Longrightarrow \exists$ zs．remove－cycles xs $x$ $y s=x \# z s \wedge x \notin$ set $z s$ $\langle p r o o f\rangle$
lemma remove－cycles－self：
$x \in$ set $x s \Longrightarrow$ remove－cycles（remove－cycles $x s x y s$ ）$x z s=$ remove－cycles xs $x$ ys
〈proof〉
lemma remove－cycles－one：remove－cycles（as＠$x$ \＃xs）$x y s=$ remove－cycles （ $x \# x s$ ）$x$ ys ＜proof〉
lemma remove-cycles-cycles:
$\exists$ xxs as.as @ concat (map ( $\lambda$ xs. $x \# x s$ ) xxs) @ remove-cycles xs $x$ ys $=x s \wedge x \notin$ set as if $x \in$ set $x s$
$\langle$ proof $\rangle$
fun start-remove :: 'a list $\Rightarrow{ }^{\prime} a \Rightarrow$ 'a list $\Rightarrow$ 'a list
where
start-remove [] - acc $=$ rev acc |
start-remove $(x \# x s)$ y acc $=$
(if $x=y$ then rev acc @ remove-cycles xs $y[y]$ else start-remove xs $y$ ( $x$ \# $a c c)$ )
lemma start-remove-decomp:
$x \in$ set $x s \Longrightarrow \exists$ as $b s . x s=$ as @ $x \#$ bs $\wedge$ start-remove $x s x y s=$ rev $y s$ @ as @ remove-cycles bs $x[x]$
$\langle p r o o f\rangle$
lemma start-remove-removes: cnt $x$ (start-remove xs $x$ ys) $\leq$ Suc (cnt $x$ ys)〈proof〉
lemma start-remove-id $[$ simp $]: x \notin$ set $x s \Longrightarrow$ start-remove xs $x$ ys $=$ rev ys @ $x s$ $\langle p r o o f\rangle$
lemma start-remove-cnt-id:
$x \neq y \Longrightarrow$ cnt $y$ (start-remove xs $x$ ys) $\leq$ cnt $y$ ys + cnt $y x s$
$\langle p r o o f\rangle$
fun remove-all-cycles $::$ ' $a$ list $\Rightarrow$ ' $a$ list $\Rightarrow$ ' $a$ list
where
remove-all-cycles [] xs $=x s \mid$
remove-all-cycles $(x \# x s)$ ys $=$ remove-all-cycles $x s$ (start-remove ys $x[])$
lemma cnt-remove-all-mono:cnt $y$ (remove-all-cycles xs ys) $\leq \max 1$ (cnt y ys)
$\langle p r o o f\rangle$
lemma cnt-remove-all-cycles: $x \in$ set $x s \Longrightarrow$ cnt $x$ (remove-all-cycles $x s$ ys) $\leq 1$
$\langle p r o o f\rangle$

```
lemma cnt-mono:
    cnt a (b# ms) \leqcnt a (b# c##xs)
\langleproof\rangle
lemma cnt-distinct-intro: }\forallx\in\mathrm{ set xs. cnt x xs }\leq1\Longrightarrow\mathrm{ distinct xs
\langleproof\rangle
lemma remove-cycles-subs:
    set (remove-cycles xs x ys)\subseteq set xs \cup set ys
<proof\rangle
lemma start-remove-subs:
    set (start-remove xs x ys)\subseteq set xs \cup set ys
<proof>
lemma remove-all-cycles-subs:
    set (remove-all-cycles xs ys)\subseteq set ys
\langleproof\rangle
lemma remove-all-cycles-distinct: set ys \subseteqset xs \Longrightarrowdistinct (remove-all-cycles
xs ys)
\langleproof\rangle
lemma distinct-remove-cycles-inv: distinct (xs @ ys)\Longrightarrow distinct (remove-cycles
xs x ys)
\langleproof\rangle
```


## definition

```
    remove-all x xs =(if x set xs then tl (remove-cycles xs x []) else xs)
```


## definition

```
    remove-all-rev x xs = (if x fet xs then rev (tl (remove-cycles (rev xs) x
[])) else xs)
lemma remove-all-distinct:
    distinct xs \Longrightarrowdistinct ( }x#\mathrm{ remove-all x xs)
\langleproof\rangle
lemma remove-all-removes:
    x\not\in set (remove-all x xs)
\langleproof\rangle
lemma remove-all-subs:
    set (remove-all x xs)\subseteq set xs
```

$\langle p r o o f\rangle$
lemma remove－all－rev－distinct：distinct $x s \Longrightarrow$ distinct $(x \#$ remove－all－rev $x x s$ ）
$\langle p r o o f\rangle$
lemma remove－all－rev－removes：$x \notin$ set（remove－all－rev $x$ xs）
〈proof〉
lemma remove－all－rev－subs：set（remove－all－rev $x$ xs $) \subseteq$ set xs
〈proof〉
abbreviation rem－cycles $i j x s \equiv$ remove－all $i$（remove－all－rev $j$（remove－all－cycles xs $x s$ ））
lemma rem－cycles－distinct＇：$i \neq j \Longrightarrow \operatorname{distinct~}(i \# j \#$ rem－cycles $i j x s)$ $\langle p r o o f\rangle$
lemma rem－cycles－removes－last：$j \notin$ set（rem－cycles ijxs） $\langle p r o o f\rangle$
lemma rem－cycles－distinct：distinct（rem－cycles ij xs）
〈proof〉
lemma rem－cycles－subs：set（rem－cycles $i j$ xs $) \subseteq$ set xs〈proof〉

## 1．3 Definition of the Algorithm

## 1．3．1 Definitions

In our formalization of the Floyd－Warshall algorithm，edge weights are from a linearly ordered abelian monoid．
class linordered－ab－monoid－add $=$ linorder + ordered－comm－monoid－add begin
subclass linordered－ab－semigroup－add $\langle p r o o f\rangle$
end
subclass（in linordered－ab－group－add）linordered－ab－monoid－add $\langle p r o o f\rangle$
context linordered－ab－monoid－add

## begin

```
type-synonym \({ }^{\prime} c\) mat \(=n a t \Rightarrow n a t \Rightarrow{ }^{\prime} c\)
definition upd \(::{ }^{\prime} c\) mat \(\Rightarrow n a t \Rightarrow n a t \Rightarrow{ }^{\prime} c \Rightarrow{ }^{\prime} c\) mat
where
    upd \(m x\) y \(v=m(x:=(m x)(y:=v))\)
definition fw-upd \(::\) 'a mat \(\Rightarrow\) nat \(\Rightarrow\) nat \(\Rightarrow\) nat \(\Rightarrow\) 'a mat where
    fw-upd \(m k i j \equiv \operatorname{upd} m i j(\min (m i j)(m i k+m k j))\)
```

Recursive version of the two inner loops.

```
fun fwi :: 'a mat => nat => nat => nat => nat => ' a mat where
    fwimnk0 0 = fw-upd mk0 0 |
    fwi m n k (Suc i) 0 = fw-upd (fwimnkin)k (Suci) 0|
    fwimnki (Suc j) = fw-upd (fwimn kij)ki(Suc j)
```

Recursive version of the full algorithm.
fun $f w::$ 'a mat $\Rightarrow$ nat $\Rightarrow n a t \Rightarrow$ 'a mat where
fw mn0 $\quad=$ fwi mn0nn|
fw $m n(S u c k)=f w i(f w m n k) n(S u c k) n n$

### 1.3.2 Elementary Properties

lemma fw-upd-mono:
fw-upd $m k i j i^{\prime} j^{\prime} \leq m i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma fw-upd-out-of-bounds1:
assumes $i^{\prime}>i$
shows (fw-upd Mkij) $i^{\prime} j^{\prime}=M i^{\prime} j^{\prime}$
<proof〉
lemma fw-upd-out-of-bounds2:
assumes $j^{\prime}>j$
shows $(f w$-upd $M k i j) i^{\prime} j^{\prime}=M i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma fwi-out-of-bounds1:
assumes $i^{\prime}>n i \leq n$
shows (fwi Mnkij) $i^{\prime} j^{\prime}=M i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma fw-out-of-bounds1:

```
    assumes }\mp@subsup{i}{}{\prime}>
    shows (fw M nk) i' j}\mp@subsup{j}{}{\prime}=M\mp@subsup{i}{}{\prime}\mp@subsup{j}{}{\prime
    <proof\rangle
```

lemma fwi-out-of-bounds2:
assumes $j^{\prime}>n j \leq n$
shows (fwi Mnkij) $i^{\prime} j^{\prime}=M i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma fw-out-of-bounds2:
assumes $j^{\prime}>n$
shows (fw Mnk) $i^{\prime} j^{\prime}=M i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma fwi-invariant-aux-1:
$j^{\prime \prime} \leq j \Longrightarrow$ fwi $m n k i j i^{\prime} j^{\prime} \leq$ fwimnkij$j^{\prime \prime} i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma fwi-invariant:
$j \leq n \Longrightarrow i^{\prime \prime} \leq i \Longrightarrow j^{\prime \prime} \leq j$
$\Longrightarrow$ fwi m n $\bar{k} i j i^{\prime} j^{\prime} \leq$ fwimn $k i^{\prime \prime} j^{\prime \prime} i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma single-row-inv:
$j^{\prime}<j \Longrightarrow$ fwi $m n k i^{\prime} j i^{\prime} j^{\prime}=$ fwimnk $i^{\prime} j^{\prime} i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma single-iteration-inv':
$i^{\prime}<i \Longrightarrow j^{\prime} \leq n \Longrightarrow$ fwi $m n k i j i^{\prime} j^{\prime}=$ fwimnk $i^{\prime} j^{\prime} i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma single-iteration-inv:
$i^{\prime} \leq i \Longrightarrow j^{\prime} \leq j \Longrightarrow j \leq n \Longrightarrow$ fwi $m n k i j i^{\prime} j^{\prime}=$ fwimnki $i^{\prime} j^{\prime} i^{\prime} j^{\prime}$
$\langle p r o o f\rangle$
lemma fwi-innermost-id:
$i^{\prime}<i \Longrightarrow f w i m n k i^{\prime} j^{\prime} i j=m i j$
$\langle p r o o f\rangle$
lemma fwi-middle-id: $j^{\prime}<j \Longrightarrow i^{\prime} \leq i \Longrightarrow f w i m n k i^{\prime} j^{\prime} i j=m i j$
$\langle p r o o f\rangle$
lemma fwi-outermost-mono:

```
    \(i \leq n \Longrightarrow j \leq n \Longrightarrow f w i m n k i j i j \leq m i j\)
\(\langle p r o o f\rangle\)
```


## lemma fwi-mono:

fwi m $n k i^{\prime} j^{\prime} i j \leq m i j$ if $i \leq n j \leq n$
$\langle p r o o f\rangle$
lemma Suc-innermost-mono:

```
    \(i \leq n \Longrightarrow j \leq n \Longrightarrow f w m n(S u c k) i j \leq f w m n k i j\)
    \(\langle p r o o f\rangle\)
```

lemma fw-mono:

```
\(i \leq n \Longrightarrow j \leq n \Longrightarrow f w m n k i j \leq m i j\)
\(\langle p r o o f\rangle\)
```

Justifies the use of destructive updates in the case that there is no negative cycle for $k$.
lemma fwi-step:
$m k k \geq 0 \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow k \leq n \Longrightarrow f w i m n k i j i j=\min$ (mij) $(m i k+m k j)$
$\langle p r o o f\rangle$

### 1.4 Result Under The Absence of Negative Cycles

If the given input graph does not contain any negative cycles, the FloydWarshall algorithm computes the unique shortest paths matrix corresponding to the graph. It contains the shortest path between any two nodes $i, j$ $\leq n$.

### 1.4.1 Length of Paths

```
fun len \(::\) 'a mat \(\Rightarrow\) nat \(\Rightarrow\) nat \(\Rightarrow\) nat list \(\Rightarrow\) ' \(a\) where
    len \(m u v[]=m u v \mid\)
    len \(m u v(w \# w s)=m u w+\) len \(m w v w s\)
lemma len-decomp: xs =ys @ y \#zs len mxzxs=len mxys+
len \(m y z z s\)
\(\langle p r o o f\rangle\)
```

lemma len-comp: len $m a c(x s @ b \# y s)=l e n m a b x s+l e n m b c y s$ $\langle p r o o f\rangle$

### 1.4.2 Canonicality

The unique shortest path matrices are in a so-called canonical form. We will say that a matrix $m$ is in canonical form for a set of indices $I$ if the following holds:
definition canonical-subs :: nat $\Rightarrow$ nat set $\Rightarrow$ 'a mat $\Rightarrow$ bool where
canonical-subs $n I m=(\forall i j k . i \leq n \wedge k \leq n \wedge j \in I \longrightarrow m i k \leq m i$ $j+m j k)$

Similarly we express that $m$ does not contain a negative cycle which only uses intermediate vertices from the set $I$ as follows:
abbreviation cyc-free-subs :: nat $\Rightarrow$ nat set $\Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ bool where cyc-free-subs $n I m \equiv \forall i x s . i \leq n \wedge$ set $x s \subseteq I \longrightarrow$ len $m i$ ixs $\geq 0$

To prove the main result under the absence of negative cycles, we will proceed as follows:

- we show that an invocation of fwi m nknn extends canonicality to index $k$,
- we show that an invocation of $f w m n n$ computes a matrix in canonical form,
- and finally we show that canonical forms specify the lengths of shortest paths, provided that there are no negative cycles.

Canonical forms specify lower bounds for the length of any path.
lemma canonical-subs-len:
$M i j \leq l e n M i j x s$ if canonical-subs $n I M i \leq n j \leq n$ set $x s \subseteq I I \subseteq$ $\{0 . . n\}$
$\langle p r o o f\rangle$
This lemma justifies the use of destructive updates under the absence of negative cycles.
lemma fwi-step':
fwi $m n k i^{\prime} j^{\prime} i j=\min (m i j)(m i k+m k j)$ if $m k k \geq 0 i^{\prime} \leq n j^{\prime} \leq n k \leq n i \leq i^{\prime} j \leq j^{\prime}$
$\langle$ proof $\rangle$
An invocation of fwi extends canonical forms.
lemma fwi-canonical-extend:
canonical-subs $n(I \cup\{k\})(f w i m n k n n)$ if canonical-subs $n I m I \subseteq\{0 . . n\} 0 \leq m k k k \leq n$ $\langle$ proof $\rangle$

An invocation of fwi will not produce a negative diagonal entry if there is no negative cycle.

```
lemma fwi-cyc-free-diag:
    fwi m nknn i i\geq0 if
    cyc-free-subs n Im 0 \leqmkkk\leqnk}\inIi\leq
    <proof\rangle
```

lemma cyc-free-subs-diag:
mii$\geq 0$ if cyc-free-subs $n$ Im $i \leq n$
$\langle p r o o f\rangle$
lemma fwi-cyc-free-subs':
cyc-free-subs $n(I \cup\{k\})($ fwi $m n k n n)$ if
cyc-free-subs $n$ I m canonical-subs $n \operatorname{Im} I \subseteq\{0 . . n\} k \leq n$
$\forall i \leq n$. fwi m n $k$ n n $i i \geq 0$
$\langle p r o o f\rangle$

## lemma fwi-cyc-free-subs:

cyc-free-subs $n(I \cup\{k\})$ (fwi m n $k n n$ ) if
cyc-free-subs $n(I \cup\{k\}) m$ canonical-subs $n \operatorname{Im} I \subseteq\{0 . . n\} k \leq n$
$\langle p r o o f\rangle$
lemma canonical-subs-empty [simp]:
canonical-subs $n$ \{\} m
$\langle$ proof $\rangle$
lemma fwi-neg-diag-neg-cycle:
$\exists i \leq n . \exists x s$. set $x s \subseteq\{0 . . k\} \wedge$ len $m i i x s<0$ if fwimnknnii<
$0 i \leq n k \leq n$
$\langle p r o o f\rangle$
fwi preserves the length of paths.
lemma fwi-len:
$\exists$ ys. set ys $\subseteq$ set $x s \cup\{k\} \wedge$ len $(f w i m n k n n) i j x s=$ len $m i j y s$ if $i \leq n j \leq n k \leq n m k k \geq 0$ set $x s \subseteq\{0 . . n\}$
$\langle p r o o f\rangle$
lemma fwi-neg-cycle-neg-cycle:
$\exists i \leq n . \exists$ ys. set $y s \subseteq$ set $x s \cup\{k\} \wedge$ len miiys $<0$ if
len (fwimnknn) iixs<0ínksn set $x s \subseteq\{0 . . n\}$
〈proof〉
If the Floyd-Warshall algorithm produces a negative diagonal entry, then there is a negative cycle.
lemma fw-neg-diag-neg-cycle:
$\exists i \leq n . \exists$ ys. set $y s \subseteq$ set $x s \cup\{0 . . k\} \wedge$ len $m i i y s<0$ if len $(f w m n k) i i x s<0 i \leq n k \leq n$ set $x s \subseteq\{0 . . n\}$
$\langle p r o o f\rangle$
Main theorem under the absence of negative cycles.
theorem fw-correct:
canonical-subs $n\{0 . . k\}(f w m n k) \wedge$ cyc-free-subs $n\{0 . . k\}(f w m n k)$ if cyc-free-subs $n\{0 . . k\} m k \leq n$
$\langle p r o o f\rangle$
lemmas fw-canonical-subs $=$ fw-correct[THEN conjunct1]
lemmas fw-cyc-free-subs $=$ fw-correct[THEN conjunct2]
lemmas cyc-free-diag $=$ cyc-free-subs-diag

### 1.5 Definition of Shortest Paths

We define the notion of the length of the shortest simple path between two vertices, using only intermediate vertices from the set $\{0 \ldots k\}$.
definition $D::$ 'a mat $\Rightarrow n a t \Rightarrow n a t \Rightarrow n a t \Rightarrow{ }^{\prime} a$ where
$D m i j k \equiv \operatorname{Min}\{l e n m i j x s \mid x s$. set $x s \subseteq\{0 . . k\} \wedge i \notin$ set $x s \wedge j \notin$ set $x s \wedge$ distinct $x s\}$
lemma distinct-length-le:finite $s \Longrightarrow$ set $x s \subseteq s \Longrightarrow$ distinct $x s \Longrightarrow$ length $x s \leq$ card $s$
$\langle p r o o f\rangle$
lemma finite-distinct: finite $s \Longrightarrow$ finite $\{x s$. set $x s \subseteq s \wedge$ distinct $x s\}$
$\langle p r o o f\rangle$
lemma D-base-finite:
finite $\{$ len $m i j x s \mid$ xs. set $x s \subseteq\{0 . . k\} \wedge$ distinct $x s\}$
$\langle p r o o f\rangle$
lemma D-base-finite':
finite $\{$ len $m i j x s \mid x s$. set $x s \subseteq\{0 . . k\} \wedge \operatorname{distinct~}(i \# j \# x s)\}$
$\langle p r o o f\rangle$
lemma D-base-finite ${ }^{\prime \prime}$ :
finite $\{$ len $m i j x s \mid x s$. set $x s \subseteq\{0 . . k\} \wedge i \notin$ set $x s \wedge j \notin$ set $x s \wedge$ distinct $x s\}$
$\langle$ proof $\rangle$
definition cycle-free $::$ 'a mat $\Rightarrow$ nat $\Rightarrow$ bool where
cycle－free $m n \equiv \forall i x s . i \leq n \wedge$ set $x s \subseteq\{0 . . n\} \longrightarrow$
$(\forall j . j \leq n \longrightarrow$ len $m i j($ rem－cycles $i j x s) \leq$ len $m i j x s) \wedge$ len $m i i$ $x s \geq 0$
lemma $D$－eqI：
fixes $m n i j k$
defines $A \equiv\{$ len $m i j x s \mid x s$ ．set $x s \subseteq\{0 . . k\}\}$
defines $A$－distinct $\equiv\{$ len $m i j x s \mid x s$ ．set $x s \subseteq\{0 . . k\} \wedge i \notin$ set $x s \wedge j \notin$ set $x s \wedge$ distinct $x s\}$
assumes cycle－free m $n i \leq n j \leq n k \leq n(\bigwedge y . y \in A$－distinct $\Longrightarrow x \leq$ y）$x \in A$
shows $D m i j k=x\langle p r o o f\rangle$
lemma D－base－not－empty：
$\{$ len $m i j x s \mid x s$. set $x s \subseteq\{0 . . k\} \wedge i \notin$ set $x s \wedge j \notin$ set $x s \wedge$ distinct $x s\}$ $\neq\{ \}$
$\langle$ proof $\rangle$
lemma Min－elem－dest：finite $A \Longrightarrow A \neq\{ \} \Longrightarrow x=\operatorname{Min} A \Longrightarrow x \in A$〈proof〉
lemma D－dest：$x=D m i j k \Longrightarrow$
$x \in\{$ len $m i j x s \mid x s$ ．set $x s \subseteq\{0$ ．．Suc $k\} \wedge i \notin$ set $x s \wedge j \notin$ set $x s \wedge$ distinct $x s\}$
〈proof $\rangle$
lemma $D$－dest ${ }^{\prime}: x=D m i j k \Longrightarrow x \in\{$ len $m i j x s \mid x s$ ．set $x s \subseteq\{0 . . S u c$ $k\}\}$
$\langle p r o o f\rangle$
lemma $D$－dest ${ }^{\prime \prime}: x=D m i j k \Longrightarrow x \in\{$ len $m i j x s \mid x s$. set $x s \subseteq\{0 . . k\}\}$ $\langle$ proof $\rangle$
lemma cycle－free－loop－dest：$i \leq n \Longrightarrow$ set $x s \subseteq\{0 . . n\} \Longrightarrow$ cycle－free $m n$ $\Longrightarrow$ len $m$ i $i x s \geq 0$
$\langle p r o o f\rangle$
lemma cycle－free－dest：
cycle－free $m n \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow$ set $x s \subseteq\{0 . . n\}$
$\Longrightarrow$ len mij（rem－cycles $i j x s) \leq$ len $m i j x s$
$\langle p r o o f\rangle$
definition cycle－free－up－to ：：＇a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool where cycle－free－up－to $m k n \equiv \forall i x s . i \leq n \wedge$ set $x s \subseteq\{0 . . k\} \longrightarrow$

```
    (\forall j.j\leqn\longrightarrowlen mij(rem-cycles ij xs ) \leqlen mijxs)^len mi i
xs\geq0
lemma cycle-free-up-to-loop-dest:
    i\leqn\Longrightarrow set xs \subseteq{0..k}\Longrightarrow cycle-free-up-to m kn\Longrightarrowlen m i ixs \geq0
<proof>
lemma cycle-free-up-to-diag:
    assumes cycle-free-up-to mkni\leqn
    shows mii\geq0
<proof>
lemma D-eqI2:
    fixes mnijk
    defines }A\equiv{len mijxs|xs.set xs \subseteq{0..k}
```



```
& set xs ^ distinct xs}
    assumes cycle-free-up-to m kni\leqnj\leqnk\leqn
        (\y.y\inA-distinct \Longrightarrowx\leqy)x\inA
    shows Dmijk=x \proof\rangle
```


## 1．5．1 Connecting the Algorithm to the Notion of Shortest Paths

Under the absence of negative cycles，the Floyd－Warshall algorithm correctly computes the length of the shortest path between any pair of vertices $i, j$ ．
lemma canonical－D：

## assumes

cycle－free－up－to $m k n$ canonical－subs $n\{0 . . k\} m i \leq n j \leq n k \leq n$ shows Dmijk＝mij
$\langle p r o o f\rangle$

## theorem fw－subs－len：

（fw mnk）$i j \leq l e n m i j x s$ if
cyc－free－subs $n\{0 . . k\} m k \leq n i \leq n j \leq n$ set $x s \subseteq I I \subseteq\{0 . . . k\}$
〈proof〉
This shows that the value calculated by $f w i$ for a pair $i, j$ always corresponds to the length of an actual path between $i$ and $j$ ．

## lemma fwi－len＇：

$\exists$ xs．set $x s \subseteq\{k\} \wedge f w i m n k i^{\prime} j^{\prime} i j=$ len $m i j x s$ if $m k k \geq 0 i^{\prime} \leq n j^{\prime} \leq n k \leq n i \leq i^{\prime} j \leq j^{\prime}$ $\langle$ proof〉

The same result for $f w$ ．
lemma fw－len：
$\exists$ xs．set $x s \subseteq\{0 . . k\} \wedge f w m n k i j=$ len $m i j x s$ if cyc－free－subs $n\{0 . . k\} m i \leq n j \leq n k \leq n$
$\langle p r o o f\rangle$

## 1．6 Intermezzo：Equivalent Characterizations of Cycle－Freeness

## 1．6．1 Shortening Negative Cycles

lemma remove－cycles－neg－cycles－aux：
fixes $i$ xs ys
defines $x s^{\prime} \equiv i \# y s$
assumes $i \notin$ set $y s$
assumes $i \in$ set $x s$
assumes $x s=a s @ \operatorname{concat}(\operatorname{map}((\#) i) x s s) @ x s^{\prime}$
assumes len mijys＞len mijxs
shows $\exists$ ys．set $y s \subseteq$ set $x s \wedge$ len miiys $<0\langle$ proof $\rangle$
lemma add－lt－neutral：$a+b<b \Longrightarrow a<0$
〈proof〉
lemma remove－cycles－neg－cycles－aux＇：
fixes $j$ xs $y s$
assumes $j \notin$ set $y s$
assumes $j \in$ set $x s$
assumes $x s=y s$＠$j \#$ concat（map（ $\lambda$ xs．xs＠$[j]$ ）xss）＠as
assumes len mijys＞len mijxs
shows $\exists$ ys．set $y s \subseteq$ set $x s \wedge$ len $m j j y s<0\langle p r o o f\rangle$
lemma add－le－impl：$a+b<a+c \Longrightarrow b<c$
〈proof〉
lemma start－remove－neg－cycles：
len $m i j$（start－remove xs $k[])>$ len $m i j x s \Longrightarrow \exists$ ys．set $y s \subseteq$ set $x s \wedge$
len $m k k y s<0$
$\langle p r o o f\rangle$
lemma remove－all－cycles－neg－cycles：
len mij（remove－all－cycles ys xs）＞len mijxs
$\Longrightarrow \exists$ ys $k$ ．set $y s \subseteq$ set $x s \wedge k \in$ set $x s \wedge$ len $m k k y s<0$
$\langle$ proof $\rangle$
lemma concat－map－cons－rev：

```
    rev (concat (map ((#) j) xss)) = concat (map (\lambda xs.xs @ [j]) (rev (map
rev xss)))
\langleproof\rangle
```

lemma negative-cycle-dest: len mij (rem-cycles $i j x s)>$ len mijxs
$\Longrightarrow \exists i^{\prime}$ ys. len $m i^{\prime} i^{\prime} y s<0 \wedge$ set $y s \subseteq$ set $x s \wedge i^{\prime} \in \operatorname{set}(i \# j \#$
xs)
$\langle$ proof $\rangle$

### 1.6.2 Cycle-Freeness

lemma cycle-free-alt-def:
cycle-free $M n \longleftrightarrow$ cycle-free-up-to $M n n$
$\langle$ proof $\rangle$
lemma negative-cycle-dest-diag:
$\neg$ cycle-free-up-to $m k n \Longrightarrow k \leq n \Longrightarrow \exists i x s . i \leq n \wedge$ set $x s \subseteq\{0 . . k\}$
$\wedge$ len miixs<0
$\langle p r o o f\rangle$
lemma negative-cycle-dest-diag':
$\neg$ cycle-free $m n \Longrightarrow \exists i x s . i \leq n \wedge$ set $x s \subseteq\{0 . . n\} \wedge$ len $m i i x s<0$ $\langle$ proof $\rangle$
abbreviation cyc-free :: 'a mat $\Rightarrow$ nat $\Rightarrow$ bool where
cyc-free $m n \equiv \forall$ ixs. $i \leq n \wedge$ set $x s \subseteq\{0 . . n\} \longrightarrow$ len $m$ i ixs $\geq 0$
lemma cycle-free-diag-intro:
cyc-free $m \mathrm{n} \Longrightarrow$ cycle-free $m n$
$\langle$ proof $\rangle$
lemma cycle-free-diag-equiv:
cyc-free $m n \longleftrightarrow$ cycle-free $m n\langle$ proof $\rangle$
lemma cycle-free-diag-dest:
cycle-free $m n \Longrightarrow$ cyc-free $m n$
〈proof〉
lemma cycle-free-upto-diag-equiv:
cycle-free-up-to $m k n \longleftrightarrow$ cyc-free-subs $n\{0 . . k\} m$ if $k \leq n$ $\langle$ proof $\rangle$
theorem fw-shortest-path-up-to:
$D m i j k=f w m n k i j$ if cyc-free-subs $n\{0 . . k\} m i \leq n j \leq n k \leq n$
$\langle p r o o f\rangle$
We do not need to prove this because the definitions match．

## lemma

cyc－free $m n \longleftrightarrow$ cyc－free－subs $n\{0 . . n\} m\langle p r o o f\rangle$
lemma cycle－free－cycle－free－up－to：
cycle－free $m n \Longrightarrow k \leq n \Longrightarrow$ cycle－free－up－to $m k n$
〈proof〉
lemma cycle－free－diag：
cycle－free $m n \Longrightarrow i \leq n \Longrightarrow 0 \leq m i i$
〈proof〉

## corollary fw－shortest－path：

cyc－free $m n \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow k \leq n \Longrightarrow D m i j k=f w m n k i$ j
$\langle p r o o f\rangle$
corollary fw－shortest：
assumes cyc－free m $n i \leq n j \leq n k \leq n$
shows fw mnnij$\leq f w m n n i k+f w m n n k j$
〈proof〉

## 1．7 Result Under the Presence of Negative Cycles

Under the presence of negative cycles，the Floyd－Warshall algorithm will detect the situation by computing a negative diagonal entry．
lemma not－cylce－free－dest：$\neg$ cycle－free $m n \Longrightarrow \exists k \leq n$ ．$\neg$ cycle－free－up－to $m k n$
$\langle p r o o f\rangle$
lemma $D$－not－diag－le：
$(x:: ' a) \in\{l e n m i j x s \mid x s$. set $x s \subseteq\{0 . . k\} \wedge i \notin$ set $x s \wedge j \notin$ set $x s \wedge$ distinct $x s\}$

$$
\Longrightarrow D m i j k \leq x\langle p r o o f\rangle
$$

lemma D－not－diag－le＇：set $x s \subseteq\{0 . . k\} \Longrightarrow i \notin$ set $x s \Longrightarrow j \notin$ set $x s \Longrightarrow$ distinct xs
$\Longrightarrow D m i j k \leq l e n m i j x s\langle p r o o f\rangle$
lemma nat－upto－subs－top－removal＇：
$S \subseteq\{0 .$. Suc $n\} \Longrightarrow$ Suc $n \notin S \Longrightarrow S \subseteq\{0 . . n\}$
$\langle$ proof〉

## lemma nat-upto-subs-top-removal:

$$
S \subseteq\{0 . . n:: n a t\} \Longrightarrow n \notin S \Longrightarrow S \subseteq\{0 . . n-1\}
$$

$\langle p r o o f\rangle$
Monotonicity with respect to $k$.
lemma fw-invariant:

$$
k^{\prime} \leq k \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow k \leq n \Longrightarrow f w m n k i j \leq f w m n k^{\prime} i j
$$ $\langle p r o o f\rangle$

lemma negative-len-shortest:
length $x s=n \Longrightarrow$ len $m$ i ixs $<0$
$\Longrightarrow \exists j$ ys. distinct $(j \# y s) \wedge$ len $m j j y s<0 \wedge j \in \operatorname{set}(i \# x s) \wedge$ set $y s \subseteq$ set $x s$
$\langle$ proof $\rangle$
lemma fw-upd-leI:
fw-upd $m^{\prime} k i j i j \leq f w-u p d m k i j i j$ if
$m^{\prime} i k \leq m i k m^{\prime} k j \leq m k j m^{\prime} i j \leq m i j$
$\langle p r o o f\rangle$

## lemma fwi-fw-upd-mono:

fwi m $n k i j i j \leq f w-u p d m k i j i j$ if $k \leq n i \leq n j \leq n$
$\langle p r o o f\rangle$
The Floyd-Warshall algorithm will always detect negative cycles. The argument goes as follows: In case there is a negative cycle, then we know that there is some smallest $k$ for which there is a negative cycle containing only intermediate vertices from the set $\{0 \ldots k\}$. We will show that then fwi $m n$ $k$ computes a negative entry on the diagonal, and thus, by monotonicity, $f w$ $m n n$ will compute a negative entry on the diagonal.
theorem $F W$-neg-cycle-detect:
$\neg$ cyc-free $m n \Longrightarrow \exists i \leq n$. fw $m n n i i<0$
$\langle$ proof $\rangle$
end

### 1.8 More on Canonical Matrices

## abbreviation

canonical $M n \equiv \forall i j k . i \leq n \wedge j \leq n \wedge k \leq n \longrightarrow M i k \leq M i j+M$ $j k$
lemma canonical-alt-def:

```
canonical M n \longleftrightarrow canonical-subs n {0..n} M
```

〈proof〉

## lemma fw－canonical：

canonical（fw m $n n$ ）$n$ if cyc－free $m n$
〈proof〉
lemma canonical－len：
canonical $M n \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow$ set $x s \subseteq\{0 . . n\} \Longrightarrow M i j \leq l e n$ Mijxs
〈proof〉

## 1．9 Additional Theorems

lemma D－cycle－free－len－dest：
cycle－free $m n$
$\Longrightarrow \forall i \leq n . \forall j \leq n . D m i j n=m^{\prime} i j \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow$ set $x s \subseteq\{0 . . n\}$
$\Longrightarrow \exists$ ys．set $y s \subseteq\{0 . . n\} \wedge$ len $m^{\prime} i j x s=$ len $m i j y s$
$\langle p r o o f\rangle$
lemma D－cyc－free－preservation：
cyc－free $m n \Longrightarrow \forall i \leq n . \forall j \leq n . D m i j n=m^{\prime} i j \Longrightarrow$ cyc－free $m^{\prime} n$ $\langle p r o o f\rangle$
abbreviation $F W m n \equiv f w m n n$
lemma $F W$－out－of－bounds1：
assumes $i>n$
shows $(F W M n) i j=M i j$
$\langle p r o o f\rangle$
lemma $F W$－out－of－bounds2：
assumes $j>n$
shows $(F W M n) i j=M i j$
$\langle$ proof $\rangle$
lemma $F W$－cyc－free－preservation：
cyc－free $m n \Longrightarrow$ cyc－free $(F W m n) n$ $\langle$ proof $\rangle$
lemma cyc－free－diag－dest＇：
cyc－free $m n \Longrightarrow i \leq n \Longrightarrow m i i \geq 0$ $\langle$ proof $\rangle$
lemma $F W$-diag-neutral-preservation:
$\forall i \leq n . M i i=0 \Longrightarrow$ cyc-free $M n \Longrightarrow \forall i \leq n .(F W M n) i i=0$ $\langle p r o o f\rangle$
lemma $F W$-fixed-preservation:
fixes $M$ :: ('a::linordered-ab-monoid-add) mat
assumes $A$ : $i \leq n M 0 i+M i 0=0$ canonical (FWMn) n cyc-free (FWMn) $n$
shows $F W M n 0 i+F W M n i 0=0\langle p r o o f\rangle$
lemma diag-cyc-free-neutral:
cyc-free $M n \Longrightarrow \forall k \leq n . M k k \leq 0 \Longrightarrow \forall i \leq n . M i i=0$
<proof $\rangle$
lemma fw-upd-canonical-subs-id:
canonical-subs $n\{k\} M \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow f w-u p d M k i j=M$ $\langle p r o o f\rangle$
lemma fw-upd-canonical-id:
canonical $M n \Longrightarrow i \leq n \Longrightarrow j \leq n \Longrightarrow k \leq n \Longrightarrow f w-u p d M k i j=M$〈proof〉
lemma fwi-canonical-id:
fwi $M n k i j=M$ if canonical-subs $n\{k\} M i \leq n j \leq n k \leq n$ $\langle p r o o f\rangle$
lemma fw-canonical-id:
fw $M n k=M$ if canonical-subs $n\{0 . . k\} M k \leq n$ $\langle p r o o f\rangle$
lemmas $F W$-canonical-id $=f w$-canonical-id $[O F-$ order.refl, unfolded canon-ical-alt-def[symmetric]]
definition $F W I M n k \equiv f w i M n k n n$
The characteristic property of fwi.
theorem fwi-characteristic:
canonical-subs $n(I \cup\{k:: n a t\})(F W I M n k) \vee(\exists i \leq n$. FWI M n $k i i$ $<0)$ if canonical-subs $n I M I \subseteq\{0 . . n\} k \leq n$ $\langle p r o o f\rangle$
end

```
theory Recursion-Combinators
    imports Refine-Imperative-HOL.IICF
begin
```


## context

begin
private definition for-comb where
for-comb f a0 $n=$ nfoldli $[0 . .<n+1](\lambda x . \operatorname{True})(\lambda k a .(f a k)) a 0$
fun for-rec :: ('a nat $\Rightarrow^{\prime} a$ nres $) \Rightarrow{ }^{\prime} a \Rightarrow$ nat $\Rightarrow{ }^{\prime} a$ nres where for-rec f a $0=$ fal 0
for-rec $f a($ Suc $n)=$ for-rec $f$ a $n \gg(\lambda x . f x($ Suc $n))$
private lemma for-comb-for-rec: for-comb fan=for-rec $f$ a $n$
$\langle p r o o f\rangle$ definition for-rec2 ${ }^{\prime}$ where
for-rec2' ${ }^{\prime}$ a $n i j=$
(if $i=0$ then RETURN a else for-rec ( $\lambda a$ i. for-rec $(\lambda a . f a i) a n) a$ $(i-1))$

$$
\gg(\lambda a . \text { for-rec }(\lambda a . f a i) a j)
$$

fun for-rec2 : : ('a nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime}$ a nres $) \Rightarrow{ }^{\prime} a \Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ 'a nres where

$$
\text { for-rec2 } f \text { a } n(S u c i) 0=\text { for-rec2 } f \text { a } n i n \gg(\lambda a . f a(S u c i) 0) \mid
$$

$$
\text { for-rec2 } f \text { a } n i(S u c j)=\text { for-rec2 } f \text { a } n i j \gg(\lambda \text { a.f a } i(S u c j))
$$

## private lemma for-rec2-for-rec2 ':

for-rec2 f a $n i j=$ for-rec2' $f$ a $n i j$
$\langle$ proof $\rangle$
fun for-rec3 :: ('a nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a$ nres $) \Rightarrow{ }^{\prime} a \Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime}$ a nres
where
private definition for-rec3' where

$$
\begin{aligned}
& \text { for-rec3 } f \text { mn } 0 \quad 0 \quad 0 \quad=f m 0001 \\
& \text { for-rec3 } f m n(\text { Suc } k) 0 \quad 0 \quad=\text { for-rec3 } f m n k n n \gg(\lambda a . f a \\
& \text { (Suc k) } 00 \text { ) | } \\
& \text { for-rec3 f } m n k \quad(\text { Suc } i) 0 \quad=\text { for-rec3 } f m n k i n \gg(\lambda a . f a k \\
& \text { (Suc i) 0) | } \\
& \text { for-rec3 } f m n k \quad i \quad(S u c j)=\text { for-rec3 } f m n k i j \gg(\lambda a . f a k \\
& i(S u c j))
\end{aligned}
$$

```
    for-rec3'f a \(n k i j=\)
    (if \(k=0\) then RETURN a else for-rec ( \(\lambda\) a \(k\). for-rec2' \((\lambda a . f a k\) ) a \(n\)
n n) \(a(k-1))\)
    \(\gg\left(\lambda a\right.\). for-rec \(\left.^{\prime}(\lambda a . f a k) a n i j\right)\)
```

private lemma for-rec3-for-rec3':
for-rec3 f a $n k i j=$ for-rec3'f a $n k i j$
$\langle p r o o f\rangle$ lemma for-rec2'-for-rec:
for-reč' ${ }^{\prime}$ f a $n$ n $n=$
for-rec ( $\lambda a$ i. for-rec ( $\lambda a . f a i$ ) a n) a $n$
$\langle p r o o f\rangle$ lemma for-rec3'-for-rec:
for-rec3'fan $n$ n $n=$
for-rec ( $\lambda$ a $k$. for-rec ( $\lambda a$ i. for-rec $(\lambda a . f a k i) a n$ ) an) an
〈proof〉
theorem for-rec-eq:
for-rec fan=nfoldli $[0 . .<n+1](\lambda x$. True $)(\lambda k a . f a k) a$
$\langle p r o o f\rangle$
theorem for-rec2-eq:
for-rec2 $f$ a $n$ n $n=$
nfoldli $[0 . .<n+1](\lambda x$. True $)$
( $\lambda i$. nfoldli $[0 . .<n+1](\lambda x . \operatorname{True})(\lambda j a . f a i j)) a$
$\langle p r o o f\rangle$
theorem for-rec3-eq:
for-rec3 f $a$ $n$ n $n$ n =
nfoldli $[0 . .<n+1](\lambda x$. True $)$
( $\lambda$ k. nfoldli $[0 . .<n+1](\lambda x$. True $)$
$(\lambda i . n f o l d l i[0 . .<n+1](\lambda x . \operatorname{True})(\lambda j a . f a k i j)))$
$a$
$\langle p r o o f\rangle$
end
lemmas [intf-of-assn] $=$ intf-of-assnI $\left[\right.$ where $R=i s-m t x ~ n$ and ' $a={ }^{\prime} b i$-mtx
for $n$ ]
declare param-upt[sepref-import-param]
end
theory $F W$-Code
imports

```
    Recursion-Combinators
    Floyd-Warshall
begin
```


### 1.10 Refinement to Efficient Imperative Code

We will now refine the recursive version of the Floyd-Warshall algorithm to an efficient imperative version. To this end, we use the Sepref framework, yielding an implementation in Imperative HOL.

```
definition fw-upd' \(::\) ('a::linordered-ab-monoid-add) \(m t x \Rightarrow\) nat \(\Rightarrow\) nat \(\Rightarrow\)
nat \(\Rightarrow{ }^{\prime}\) 'a mtx nres where
    fw-upd' mkij=
    RETURN
        op-mtx-set \(m(i, j)(\min (o p-m t x-g e t m(i, j))(o p-m t x-g e t m(i, k)+\)
op-mtx-get \(m(k, j)))\)
    )
lemma fw-upd'-alt-def:
    fw-upd' \(m k i j=\)
    RETURN (
        let
            \(e=o p-m t x-g e t m(i, k)+o p-m t x-g e t m(k, j)\)
        in if \(e<o p-m t x-g e t ~ m(i, j)\) then op-mtx-set \(m(i, j)\) e else \(m\)
)
\(\langle p r o o f\rangle\)
```

definition fwi' :: ('a::linordered-ab-monoid-add) $m t x \Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a \mathrm{mtx}$ nres
where

```
fwi' \(m n k i j=R E C T(\lambda f w(m, k, i, j)\).
        case ( \(i, j\) ) of
        ( 0,0 ) \(\Rightarrow\) fw-upd' mkoll 0
        (Suc \(i, 0) \Rightarrow d o\left\{m^{\prime} \leftarrow f w(m, k, i, n) ; f w-u p d^{\prime} m^{\prime} k(\right.\) Suc i) 0\(\} \mid\)
        \((i, S u c j) \Rightarrow d o\left\{m^{\prime} \leftarrow f w(m, k, i, j) ; f w-u p d^{\prime} m^{\prime} k i(\right.\) Suc \(\left.j)\right\}\)
    ) \((m, k, i, j)\)
lemma fwi'-simps:
    fwi' \(m n k 0 \quad 0 \quad=f w\)-upd \(d^{\prime} m k 00\)
    \(f w i^{\prime} m n k(S u c i) 0=d o\left\{m^{\prime} \leftarrow f w i^{\prime} m n k i n ; f w-u p d^{\prime} m^{\prime} k(S u c\right.\)
i) 0\(\}\)
    \(f w i^{\prime} m n k i \quad(S u c j)=d o\left\{m^{\prime} \leftarrow f w i^{\prime} m n k i j ; f w-u p d^{\prime} m^{\prime} k i(S u c\right.\)
j) \(\}\)
\(\langle p r o o f\rangle\)
```


## lemma

```
    fwi' mnkij \(\leq \operatorname{SPEC}(\lambda r . r=\operatorname{uncurry}(f w i(\operatorname{curry} m) n k i j))\)
```

$\langle p r o o f\rangle$
lemma fw－upd＇－spec：
fw－upd＇$M k i j \leq \operatorname{SPEC}\left(\lambda M^{\prime} . M^{\prime}=\operatorname{uncurry}(f w-u p d(\operatorname{curry} M) k i j)\right)$ $\langle p r o o f\rangle$
lemma for－reç－fwi：
for－rec2（ $\lambda$ M．fw－upd ${ }^{\prime} M k$ ）Mnij$\leq \operatorname{SPEC}\left(\lambda M^{\prime} . M^{\prime}=u n c u r r y ~(f w i\right.$ （ （urry M）$n k i j$ ）
〈proof〉
definition $f w^{\prime}$ ：：（＇a：：linordered－ab－monoid－add）$m t x \Rightarrow n a t \Rightarrow n a t \Rightarrow{ }^{\prime} a$ mtx nres where
$f w^{\prime} m n k=n f o l d l i[0 . .<k+1](\lambda-$. True $)\left(\lambda k M . f o r-r e c \mathcal{D}\left(\lambda M . f w-u p d^{\prime}\right.\right.$ Mk）$M n n n$ ）$m$
lemma $f w^{\prime}$－spec：

```
    fw'mnk\leqSPEC (\lambda M'. M' = uncurry (fw (curry m) nk))
```

〈proof〉

## context

fixes $n::$ nat
fixes dummy ：：＇a：：\｛linordered－ab－monoid－add，zero，heap\}

## begin

lemma［sepref－import－param］：$((+),(+):: ' a \Rightarrow-) \in I d \rightarrow I d \rightarrow I d\langle$ proof $\rangle$
lemma［sepref－import－param］：（min，min：：＇a $\Rightarrow-$ ）$\in I d \rightarrow I d \rightarrow I d\langle$ proof $\rangle$

```
abbreviation node-assn \(\equiv\) nat-assn
abbreviation mtx-assn \(\equiv\) asmtx-assn (Suc n) id-assn::('a mtx \(\Rightarrow\)-)
sepref-definition fw-upd-impl1 is
    uncurry2 (uncurry fw-upd) ::
    \([\lambda(((-, k), i), j) . k \leq n \wedge i \leq n \wedge j \leq n]_{a} m t x-a s s n^{d} *_{a}\) node-assn \({ }^{k} *_{a}\)
node-assn \({ }^{k} *_{a}\) node-assn \({ }^{k}\)
    \(\rightarrow m t x-a s s n\)
    〈proof〉
```

sepref-definition fw-upd-impl is
uncurry2 (uncurry fw-upd ${ }^{\prime}$ ) ::
$[\lambda(((-, k), i), j) . k \leq n \wedge i \leq n \wedge j \leq n]_{a} m t x-$ assn $^{d} *_{a}$ node－assn ${ }^{k} *_{a}$ node－assn ${ }^{k} *_{a}$ node－assn ${ }^{k}$
$\rightarrow$ mtx－assn
$\langle$ proof $\rangle$
sepref－register $f w$－upd＇$::{ }^{\prime} a \operatorname{i-mtx} \Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a$ i－mtx nres

## definition

fwi－impl＇（ $M$ ：：＇a mtx）$k=$ for－rec2（ $\lambda$ M．fw－upd＇$M k$ ）$M n n n$

## definition

$f w-i m p l l^{\prime}\left(M::{ }^{\prime} a m t x\right)=f w^{\prime} M n n$

## context

notes $[$ id－rules $]=$ itypeI $[$ of $n$ TYPE（nat）］
and $[$ sepref－import－param $]=I d I[$ of $n]$
begin
sepref－definition $f w$－$i m p l$ is
fw－impl ${ }^{\prime}:: m t x-a s s n^{d} \rightarrow_{a} m t x$－assn
〈proof〉
sepref－definition $f w$－impl1 is
fw－impl ${ }^{\prime}:: m t x-a s s n^{d} \rightarrow{ }_{a} m t x-a s s n$
〈proof〉
sepref－definition fwi－impl is
uncurry fwi－impl $l^{\prime}::[\lambda(-, k) . k \leq n]_{a} m t x-a s s n^{d} *_{a}$ node－assn ${ }^{k} \rightarrow m t x$－assn〈proof〉
sepref－definition fwi－impl1 is
uncurry fwi－impl＇$::[\lambda(-, k) . k \leq n]_{a} m t x-$ assn $^{d} *_{a}$ node－assn ${ }^{k} \rightarrow$ mtx－assn〈proof〉
end
end
export－code $f w-i m p l$ in $S M L-i m p$
A compact specification for the characteristic property of the Floyd－Warshall algorithm．
definition fw－spec where

```
fw-spec \(n M \equiv \operatorname{SPEC}\left(\lambda M^{\prime}\right.\).
    if \(\left(\exists i \leq n . M^{\prime} i i<0\right)\)
    then \(\neg\) cyc-free \(M n\)
    else \(\forall i \leq n . \forall j \leq n . M^{\prime} i j=D M i j n \wedge\) cyc-free \(\left.M n\right)\)
lemma D-diag-nonnegI:
    assumes cycle-free \(M n i \leq n\)
    shows \(D M\) i \(i n \geq 0\)
    〈proof〉
lemma fw-fw-spec:
    RETURN \((F W M n) \leq f w\)-spec \(n M\)
\(\langle p r o o f\rangle\)
```


## definition

```
mat-curry-rel \(=\{(M u, M c)\). curry \(M u=M c\}\)
```

```
mat-curry-rel \(=\{(M u, M c)\). curry \(M u=M c\}\)
```


## definition

```
\(m t x-c u r r y-a s s n ~ n=h r-c o m p(m t x-a s s n ~ n)(b r \operatorname{curry}(\lambda-. \operatorname{True}))\)
```

$m t x-c u r r y-a s s n ~ n=h r-c o m p(m t x-a s s n ~ n)(b r \operatorname{curry}(\lambda-. \operatorname{True}))$
declare mtx-curry-assn-def[symmetric, fcomp-norm-unfold]
lemma fw-impl'-correct:
$(f w-i m p l \prime, f w-s p e c) \in I d \rightarrow$ br curry $(\lambda-. \operatorname{True}) \rightarrow\langle b r \operatorname{curry}(\lambda-. \operatorname{True})\rangle$
nres-rel
$\langle p r o o f\rangle$

```

\subsection*{1.10.1 Main Result}

This is one way to state that the \(f w\)-impl fulfills the specification \(f w\)-spec.
```

theorem fw-impl-correct:
(fw-impl n, fw-spec n) \in(mtx-curry-assn n)d}\mp@subsup{)}{a}{d}mtx\mathrm{ -curry-assn n
<proof\rangle

```

An alternative version: a Hoare triple for total correctness.
```

corollary
$<m t x$-curry-assn $n M \operatorname{Mi}>f w-i m p l n M i<\lambda M i^{\prime} . \exists_{A} M^{\prime}$. mtx-curry-assn
$n M^{\prime} M i^{\prime} * \uparrow$
(if $\left(\exists i \leq n . M^{\prime} i i<0\right)$
then $\neg$ cyc-free $M n$
else $\forall i \leq n . \forall j \leq n . M^{\prime} i j=D M i j n \wedge c y c$-free $\left.M n\right)>_{t}$
$\langle p r o o f\rangle$

```

\subsection*{1.10.2 Alternative versions for Uncurried Matrices.}
```

definition FWI' = uncurry ooo FWI o curry
lemma fwi-impl'-refine-FWI':
(fwi-impl' n, RETURN oo PR-CONST (\lambda M.FWI' M n)) \in Id }->\mathrm{ Id }
<Id\rangle nres-rel
<proof\rangle

```
lemmas fwi-impl-refine-FWI' \(=\) fwi-impl.refine \([F C O M P\) fwi-impl'-refine-FWI]
definition \(F W^{\prime}=\) uncurry oo \(F W\) o curry
definition \(F W^{\prime \prime} n M=F W^{\prime} M n\)
lemma \(f w\)-impl'-refine- \(F W^{\prime \prime}\) :
    \(\left(f w-\right.\) impl \({ }^{\prime} n\), RETURN o PR-CONST \(\left.\left(F W^{\prime \prime} n\right)\right) \in I d \rightarrow\langle I d\rangle\) nres-rel
    <proof〉
lemmas fw-impl-refine- \(F W^{\prime \prime}=\) fw-impl.refine \(\left[F C O M P\right.\) fw-impl'-refine- \(\left.F W^{\prime \prime}\right]\)
lemmas fw-impl1-refine- \(F W^{\prime \prime}=\) fw-impl1.refine \(\left[F C O M P\right.\) fw-impl'-refine- \(\left.F W^{\prime \prime}\right]\)
end

\section*{References}
[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. Commun. ACM, 5(6):345-, June 1962.
[Roy59] Bernard Roy. Transitivité et connexité. In Extrait des comptes rendus des séances de lAcadémie des Sciences, pages 216-218. Gauthier-Villars, July 1959. http://gallica.bnf.fr/ark:/12148/ bpt6k3201c/f222.image.langFR.
[War62] Stephen Warshall. A theorem on boolean matrices. J. ACM, 9(1):11-12, January 1962.```

