graph-verification

By christine

April 13, 2015

Contents

theory *Check-Connected-Impl*
imports
  Vcg
  ../Witness-Property/Connected-Components
begin

type-synonym *IVertex* = *nat*
type-synonym *IEdge-Id* = *nat*
type-synonym *IEdge* = *IVertex* × *IVertex*
type-synonym *IPEdge* = *IVertex* ⇒ *IEdge-Id option*
type-synonym *INum* = *IVertex* ⇒ *nat*
type-synonym *IGraph* = *nat* × *nat* × (*IEdge-Id* ⇒ *IEdge*)

abbreviation *ivertex-cnt* :: *IGraph* ⇒ *nat*
  where *ivertex-cnt G* ≡ *fst G*

abbreviation *iedge-cnt* :: *IGraph* ⇒ *nat*
  where *iedge-cnt G* ≡ *fst* (*snd G*)

abbreviation *iedges* :: *IGraph* ⇒ *IEdge-Id* ⇒ *IEdge*
  where *iedges G* ≡ *snd* (*snd G*)

definition *is-wellformed-inv* :: *IGraph* ⇒ *nat* ⇒ *bool* **where**
  is-wellformed-inv G i ≡ ∀ *k* < *i*. *ivertex-cnt G* > *fst* (*iedges G k*)
        ∧ *ivertex-cnt G* > *snd* (*iedges G k*)
ML ⟪ *Toplevel.theory* ⟫
procedures *is-wellformed* (*G* :: *IGraph* | *R* :: *bool*)
  where
    i :: *nat*
    e :: *IEdge*
  in
    ANNO G.⦃ ´*G* = *G* ⦄
      ´*R* :== *True* ;;
      ´*i* :== *0* ;;
      TRY
        WHILE ´*i* < *iedge-cnt* ´*G*

```
    INV {| ´R = is-wellformed-inv ´G ´i ∧
          ´i ≤ iedge-cnt ´G ∧ ´G = G |}
    VAR MEASURE (iedge-cnt ´G − ´i)
    DO
     ´e :== iedges ´G ´i ;;
     IF ivertex-cnt ´G ≤ fst ´e ∨ ivertex-cnt ´G ≤ snd ´e THEN
       ´R :== False ;;
       THROW
     FI ;;
     ´i :== ´i + 1
    OD
  CATCH SKIP END
  {| ´G = G ∧
    ´R = is-wellformed-inv ´G (iedge-cnt ´G) |}
```

definition *parent-num-assms-inv* :: *IGraph* ⇒ *IVertex* ⇒ *IPEdge* ⇒ *INum* ⇒ *nat* ⇒ *bool* **where**

```
  parent-num-assms-inv G r p n k ≡ ∀ i < k. i ≠ r ⟶ (case p i of
      None ⇒ False
    | Some x ⇒ x < iedge-cnt G ∧ snd (iedges G x) = i ∧ n i = n (fst (iedges G
x)) + 1)
```

procedures *parent-num-assms* (*G* :: *IGraph*, *r* :: *IVertex*, *parent-edge* :: *IPEdge*,
num :: *INum* | *R* :: *bool*)

```
  where
    vertex :: IVertex
    edge-id :: IEdge-Id
  in
    ANNO (G,r,p,n).
      {| ´G = G ∧ ´r = r ∧ ´parent-edge = p ∧ ´num = n |}
      ´R :== True ;;
      ´vertex :== 0 ;;
      TRY
        WHILE ´vertex < ivertex-cnt ´G
        INV {| ´R = parent-num-assms-inv ´G ´r ´parent-edge ´num ´vertex
          ∧ ´G = G ∧ ´r = r ∧ ´parent-edge = p ∧ ´num = n
          ∧ ´vertex ≤ ivertex-cnt ´G|}
        VAR MEASURE (ivertex-cnt ´G − ´vertex)
        DO
          IF (´vertex ≠ ´r) THEN
            IF ´parent-edge ´vertex = None THEN
              ´R :== False ;;
              THROW
            FI ;;
            ´edge-id :== the (´parent-edge ´vertex) ;;
            IF ´edge-id ≥ iedge-cnt ´G
                ∨ snd (iedges ´G ´edge-id) ≠ ´vertex
                ∨ ´num ´vertex ≠ ´num (fst (iedges ´G ´edge-id)) + 1 THEN
              ´R :== False ;;
```

```
          THROW
        FI
      FI ;;
      ´vertex :== ´vertex + 1
    OD
  CATCH SKIP END
⦃ ´G = G ∧ ´r = r ∧ ´parent-edge = p ∧ ´num = n
  ∧ ´R = parent-num-assms-inv ´G ´r ´parent-edge ´num (ivertex-cnt ´G)⦄
```

procedures *check-connected* (*G* :: *IGraph*, *r* :: *IVertex*, *parent-edge* :: *IPEdge*,
  num :: *INum* | *R* :: *bool*)
 where
  R1 :: *bool*
  R2 :: *bool*
  R3 :: *bool*
 in

```
  ´R1 :== CALL is-wellformed(´G) ;;
  ´R2 :== ´r < ivertex-cnt ´G ∧ ´num ´r = 0 ∧ ´parent-edge ´r = None ;;
  ´R3 :== CALL parent-num-assms(´G, ´r, ´parent-edge, ´num) ;;
  ´R :== ´R1 ∧ ´R2 ∧ ´R3
```

end
theory *Check-Connected-Verification*
imports *Vcg Check-Connected-Impl*
begin

definition *no-loops* :: $('a, 'b)$ *pre-digraph* $\Rightarrow$ *bool* **where**
 no-loops $G \equiv \forall e \in$ *arcs* G. *tail* $G\ e \neq$ *head* $G\ e$

definition *abs-IGraph* :: *IGraph* $\Rightarrow$ (*nat*, *nat*) *pre-digraph* **where**
 abs-IGraph $G \equiv$ ⦇ *verts* = {0..<*ivertex-cnt* G}, *arcs* = {0..<*iedge-cnt* G},
  tail = *fst* o *iedges* G, *head* = *snd* o *iedges* G ⦈

lemma *verts-absI*[*simp*]: *verts* (*abs-IGraph* G) = {0..<*ivertex-cnt* G}
 and *arcs-absI*[*simp*]: *arcs* (*abs-IGraph* G) = {0..<*iedge-cnt* G}
 and *tail-absI*[*simp*]: *tail* (*abs-IGraph* G) e = *fst* (*iedges* G e)
 and *head-absI*[*simp*]: *head* (*abs-IGraph* G) e = *snd* (*iedges* G e)
 by (*auto simp*: *abs-IGraph-def*)

lemma *is-wellformed-inv-step*:
 is-wellformed-inv G (*Suc* i) ⟷ *is-wellformed-inv* G i
   ∧ *fst* (*iedges* G i) < *ivertex-cnt* G ∧ *snd* (*iedges* G i) < *ivertex-cnt* G
 by (*auto simp add*: *is-wellformed-inv-def less-Suc-eq*)

lemma (**in** *is-wellformed-impl*) *is-wellformed-spec*:
 $\forall G. \Gamma \vdash_t$ ⦃´G = G⦄ ´R :== *PROC is-wellformed*(´G) ⦃´R = *is-wellformed-inv*
G (*iedge-cnt* G)⦄
 apply *vcg*
 apply (*auto simp*: *is-wellformed-inv-step*)

apply (*auto simp*: *is-wellformed-inv-def*)
done

lemma *parent-num-assms-inv-step*:
parent-num-assms-inv G r p n (*Suc i*) $\longleftrightarrow$ *parent-num-assms-inv G r p n i*
$\wedge$ ($i \neq r \longrightarrow$ (*case p i of*
None $\Rightarrow$ *False*
| *Some x* $\Rightarrow$ $x <$ *iedge-cnt G* $\wedge$ *snd* (*iedges G x*) $= i \wedge n\ i = n$ (*fst* (*iedges G x*)) $+ 1$))
by (*auto simp*: *parent-num-assms-inv-def less-Suc-eq*)

lemma (**in** *parent-num-assms-impl*) *parent-num-assms-spec*:
$\forall G\ r\ p\ n.\ \Gamma \vdash_t \{\!|\ \acute{}G = G \wedge \acute{}r = r \wedge \acute{}$*parent-edge* $= p \wedge \acute{}$*num* $= n|\!\}$
$\acute{}R$:== *PROC parent-num-assms*($\acute{}G$, $\acute{}r$, $\acute{}$*parent-edge*, $\acute{}$*num*)
$\{\!|\ \acute{}R =$ *parent-num-assms-inv G r p n* (*ivertex-cnt G*)$|\!\}$
apply *vcg*
apply (*simp-all add*: *parent-num-assms-inv-step*)
apply (*auto simp*: *parent-num-assms-inv-def*)
done

lemma *connected-components-locale-eq-invariants*:
$\bigwedge G\ r\ p\ n.$
connected-components-locale (*abs-IGraph G*) *n p r* =
(*is-wellformed-inv G* (*iedge-cnt G*) $\wedge$
$r <$ *ivertex-cnt G* $\wedge\ n\ r = 0 \wedge p\ r =$ *None* $\wedge$
parent-num-assms-inv G r p n (*ivertex-cnt G*))
proof –
fix *G r p n*
let *?aG* = *abs-IGraph G*
have *is-wellformed-inv G* (*iedge-cnt G*) = *fin-digraph ?aG*
unfolding *is-wellformed-inv-def fin-digraph-def fin-digraph-axioms-def wf-digraph-def*
by *auto*
moreover
have ($\forall v \in$ *verts ?aG*. $v \neq r \longrightarrow$
($\exists e \in$ *arcs ?aG*. $p\ v =$ *Some e* $\wedge$
head ?aG e $= v \wedge$
$n\ v =\ n$ (*tail ?aG e*) $+ 1$))
= *parent-num-assms-inv G r p n* (*ivertex-cnt G*)
proof –
{ fix *i* **assume** (*case p i of None* $\Rightarrow$ *False*
| *Some x* $\Rightarrow x <$ *iedge-cnt G* $\wedge$ *snd* (*iedges G x*) $= i \wedge n\ i = n$ (*fst* (*iedges G x*)) $+ 1$)
then have $\exists x \in \{0..<$*iedge-cnt G*$\}$. $p\ i =$ *Some x* $\wedge$ *snd* (*iedges G x*) $= i \wedge n\ i = n$ (*fst* (*iedges G x*)) $+ 1$
by (*case-tac p i*) *auto* **}**
then show *?thesis*
by (*auto simp*: *parent-num-assms-inv-def*)
qed

ultimately
show *?thesis G r p n*
  unfolding *connected-components-locale-def*
  connected-components-locale-axioms-def **by** *auto*
qed

theorem (**in** *check-connected-impl*) *check-connected-eq-locale*:
  $\forall\, G\ r\ p\ n.\ \Gamma \vdash_t \{\!|\ \acute{}G = G \wedge \acute{}r = r \wedge \acute{}parent\text{-}edge = p \wedge \acute{}num = n\ |\!\}$
    $\acute{}R$:== *PROC check-connected* ($\acute{}G$, $\acute{}r$, $\acute{}parent\text{-}edge$, $\acute{}num$)
    $\{\!|\ \acute{}R = connected\text{-}components\text{-}locale\ (abs\text{-}IGraph\ G)\ n\ p\ r|\!\}$
by *vcg* (*auto simp*: *connected-components-locale-eq-invariants*)

lemma *connected-components-locale-imp-correct*:
  assumes *connected-components-locale* (*abs-IGraph G*)*n p r*
  assumes $u \in$ *pverts* (*mk-symmetric* (*abs-IGraph G*))
  assumes $v \in$ *pverts* (*mk-symmetric* (*abs-IGraph G*))
  shows $\exists\, p$. *pre-digraph.apath* (*mk-symmetric* (*abs-IGraph G*)) *u p v*
proof −
  interpret *S*: *pair-wf-digraph mk-symmetric* (*abs-IGraph G*)
    by (*intro wf-digraph.wellformed-mk-symmetric*
      connected-components-locale.ccl-wellformed[*OF assms*(*1*)])
  show *?thesis*
    using *connected-components-locale.connected-by-path*[*OF assms*]
    by (*simp only*: *S.reachable-apath*)
qed

theorem (**in** *check-connected-impl*) *check-connected-spec*:
  $\bigwedge G\ r\ p\ n.\ \Gamma \vdash_t \{\!|\ \acute{}G = G \wedge \acute{}r = r \wedge \acute{}parent\text{-}edge = p \wedge \acute{}num = n\ |\!\}$
    $\acute{}R$:== *PROC check-connected*($\acute{}G$, $\acute{}r$, $\acute{}parent\text{-}edge$, $\acute{}num$)
    $\{\!|\ \acute{}R \longrightarrow$
      ($\forall\, u \in$ *pverts* (*mk-symmetric* (*abs-IGraph G*)).
        $\forall\, v \in$ *pverts* (*mk-symmetric* (*abs-IGraph G*)).
        $\exists\, p$. *pre-digraph.apath* (*mk-symmetric* (*abs-IGraph G*)) *u p v*)$|\!\}$
using *connected-components-locale-eq-invariants*
      connected-components-locale-imp-correct
by *vcg metis*

end
theory *Check-Shortest-Path-Impl*
imports
  Vcg
  ../Witness-Property/Shortest-Path-Theory
~~/src/HOL/Statespace/StateSpaceLocale
begin

type-synonym *IVertex* = *nat*
type-synonym *IEdge-Id* = *nat*
type-synonym *IEdge* = *IVertex* × *IVertex*
type-synonym *ICost* = *IEdge-Id* ⇒ *nat*

```
type-synonym IDist = IVertex ⇒ ereal
type-synonym IPEdge = IVertex ⇒ IEdge-Id option
type-synonym INum = IVertex ⇒ enat
type-synonym IGraph = nat × nat × (IEdge-Id ⇒ IEdge)

abbreviation ivertex-cnt :: IGraph ⇒ nat
  where ivertex-cnt G ≡ fst G

abbreviation iedge-cnt :: IGraph ⇒ nat
  where iedge-cnt G ≡ fst (snd G)

abbreviation iarcs :: IGraph ⇒ IEdge-Id ⇒ IEdge
  where iarcs G ≡ snd (snd G)

definition is-wellformed-inv :: IGraph ⇒ nat ⇒ bool where
  is-wellformed-inv G i ≡ ∀ k < i. ivertex-cnt G > fst (iarcs G k)
        ∧ ivertex-cnt G > snd (iarcs G k)

procedures is-wellformed (G :: IGraph | R :: bool)
  where
    i :: nat
    e :: IEdge
  in
    ANNO G.
      {| ´G = G |}
      ´R :== True ;;
      ´i :== 0 ;;
      TRY
        WHILE ´i < iedge-cnt ´G
        INV {| ´R = is-wellformed-inv ´G ´i ∧ ´i ≤ iedge-cnt ´G ∧ ´G = G |}
        VAR MEASURE (iedge-cnt ´G − ´i)
        DO
         ´e :== iarcs ´G ´i ;;
         IF ivertex-cnt ´G ≤ fst ´e ∨ ivertex-cnt ´G ≤ snd ´e THEN
           ´R :== False ;;
           THROW
         FI ;;
         ´i :== ´i + 1
        OD
      CATCH SKIP END
      {| ´G = G ∧ ´R = is-wellformed-inv ´G (iedge-cnt ´G) |}


definition trian-inv :: IGraph ⇒ IDist ⇒ ICost ⇒ nat ⇒ bool where
  trian-inv G d c m ≡
    ∀ i < m. d (snd (iarcs G i)) ≤ d (fst (iarcs G i)) + ereal (c i)

procedures trian (G :: IGraph, dist :: IDist, c :: ICost | R :: bool)
  where
```

```
    edge-id :: IEdge-Id
  in
    ANNO (G,dist,c).
      {| ´G = G ∧ ´dist = dist ∧ ´c = c |}
      ´R :== True ;;
      ´edge-id :== 0 ;;
      TRY
        WHILE ´edge-id < iedge-cnt ´G
        INV {| ´R = trian-inv ´G ´dist ´c ´edge-id
          ∧ ´G = G ∧ ´dist = dist ∧ ´c = c
          ∧ ´edge-id ≤ iedge-cnt ´G|}
        VAR MEASURE (iedge-cnt ´G − ´edge-id)
        DO
          IF  ´dist (snd (iarcs ´G ´edge-id)) >
               ´dist (fst (iarcs ´G ´edge-id)) +
               ereal (´c ´edge-id) THEN
            ´R :== False ;;
            THROW
          FI ;;
          ´edge-id :== ´edge-id + 1
        OD
      CATCH SKIP END
      {| ´G = G ∧ ´dist = dist ∧ ´c = c
      ∧ ´R = trian-inv ´G ´dist ´c (iedge-cnt ´G) |}


definition just-inv ::
  IGraph ⇒ IDist ⇒ ICost ⇒ IVertex ⇒ INum ⇒ IPEdge ⇒ nat ⇒ bool where
  just-inv G d c s n p k ≡
    ∀ v < k. v ≠ s ∧ n v ≠ ∞ ⟶
      (∃ e. e = the (p v) ∧ e < iedge-cnt G ∧
        v = snd (iarcs G e) ∧
        d v = d (fst (iarcs G e)) + ereal (c e) ∧
        n v = n (fst (iarcs G e)) + (enat 1))

procedures just (G :: IGraph, dist :: IDist, c :: ICost,
    s :: IVertex, enum :: INum, pred :: IPEdge | R :: bool)
  where
    v :: IVertex
    edge-id :: IEdge-Id
  in
    ANNO (G,dist, c, s ,enum, pred).
      {| ´G = G ∧ ´dist = dist ∧ ´c = c ∧ ´s = s ∧ ´enum = enum ∧ ´pred =
pred|}
      ´R :== True ;;
      ´v :== 0 ;;
      TRY
        WHILE ´v < ivertex-cnt ´G
        INV {| ´R = just-inv ´G ´dist ´c ´s ´enum ´pred ´v
```

```
        ∧ ´G = G ∧ ´c = c ∧ ´s = s ∧ ´dist = dist
        ∧ ´enum = enum ∧ ´pred = pred
        ∧ ´v ≤ ivertex-cnt ´G⦄
      VAR MEASURE (ivertex-cnt ´G − ´v)
      DO
        ´edge-id :== the (´pred ´v) ;;
        IF (´v ≠ ´s) ∧  ´enum ´v ≠ ∞ ∧
           (´edge-id ≥ iedge-cnt ´G
            ∨ snd (iarcs ´G ´edge-id) ≠ ´v
            ∨ ´dist ´v ≠
               ´dist (fst (iarcs ´G ´edge-id)) + ereal (´c ´edge-id)
            ∨ ´enum ´v ≠ ´enum (fst (iarcs ´G ´edge-id)) + (enat 1)) THEN
          ´R :== False ;;
          THROW
        FI;;
        ´v :== ´v + 1
      OD
    CATCH SKIP END
  ⦃ ´G = G ∧ ´dist = dist ∧ ´c = c ∧ ´s = s ∧ ´enum = enum ∧ ´pred = pred
    ∧ ´R = just-inv ´G ´dist ´c ´s ´enum ´pred (ivertex-cnt ´G) ⦄
```

definition *no-path-inv* :: *IGraph* ⇒ *IDist* ⇒ *INum* ⇒ *nat* ⇒ *bool* **where**
no-path-inv G d n k ≡ ∀ *v* < *k*. (*d v* = ∞ ⟷ *n v* = ∞)

procedures *no-path* (*G* :: *IGraph*, *dist* :: *IDist*, *enum* :: *INum* | *R* :: *bool*)
where
v :: *IVertex*
in

```
    ANNO (G,dist,enum).
      ⦃ ´G = G ∧ ´dist = dist ∧ ´enum = enum ⦄
      ´R :== True ;;
      ´v :== 0 ;;
      TRY
        WHILE ´v < ivertex-cnt ´G
        INV ⦃ ´R = no-path-inv ´G ´dist ´enum ´v
          ∧ ´G = G ∧ ´dist = dist ∧ ´enum = enum
          ∧ ´v ≤ ivertex-cnt ´G⦄
        VAR MEASURE (ivertex-cnt ´G − ´v)
        DO
          IF ¬(´dist ´v = ∞ ⟷ ´enum ´v = ∞) THEN
            ´R :== False ;;
            THROW
          FI ;;
          ´v :== ´v + 1
        OD
      CATCH SKIP END
      ⦃ ´G = G ∧ ´dist = dist ∧ ´enum = enum
      ∧ ´R = no-path-inv ´G ´dist ´enum (ivertex-cnt ´G) ⦄
```

definition *non-neg-cost-inv* :: *IGraph* ⇒ *ICost* ⇒ *nat* ⇒ *bool* **where**
  non-neg-cost-inv G c m ≡ ∀ *e* < *m*. *c e* ≥ *0*

procedures *non-neg-cost* (*G* :: *IGraph*, *c* :: *ICost* | *R* :: *bool*)
  where
    edge-id :: *IEdge-Id*
  in
    ANNO (*G*,*c*).
      ⦃ ´*G* = *G* ∧ ´*c* = *c* ⦄
      ´*R* :== *True* ;;
      ´*edge-id* :== *0* ;;
      TRY
        WHILE ´*edge-id* < *iedge-cnt* ´*G*
        INV ⦃ ´*R* = *non-neg-cost-inv* ´*G* ´*c* ´*edge-id*
          ∧ ´*G* = *G* ∧ ´*c* = *c*
          ∧ ´*edge-id* ≤ *iedge-cnt* ´*G*⦄
        VAR MEASURE (*iedge-cnt* ´*G* − ´*edge-id*)
        DO
          IF ´*c* ´*edge-id* < *0 THEN*
            ´*R* :== *False* ;;
            THROW
          FI ;;
          ´*edge-id* :== ´*edge-id* + *1*
        OD
      CATCH SKIP END
      ⦃ ´*G* = *G* ∧ ´*c* = *c*
      ∧ ´*R* = *non-neg-cost-inv* ´*G* ´*c* (*iedge-cnt* ´*G*) ⦄

procedures *check-basic-just-sp* (*G* :: *IGraph*, *dist* :: *IDist*, *c* :: *ICost*,
    s :: *IVertex*, *enum* :: *INum*, *pred* :: *IPEdge* | *R* :: *bool*)
  where
    R1 :: *bool*
    R2 :: *bool*
    R3 :: *bool*
    R4 :: *bool*
  in
    ´*R1* :== *CALL is-wellformed* (´*G*) ;;
    ´*R2* :== ´*dist* ´*s* ≤ *0* ;;
    ´*R3* :== *CALL trian* (´*G*, ´*dist*, ´*c*) ;;
    ´*R4* :== *CALL just* (´*G*, ´*dist*, ´*c*, ´*s*, ´*enum*, ´*pred*) ;;
    ´*R* :== ´*R1* ∧ ´*R2* ∧ ´*R3* ∧ ´*R4*

procedures *check-sp* (*G* :: *IGraph*, *dist* :: *IDist*, *c* :: *ICost*,
    s :: *IVertex*, *enum* :: *INum*, *pred* :: *IPEdge* | *R* :: *bool*)
  where

R1 :: *bool*
R2 :: *bool*
R3 :: *bool*
R4 :: *bool*
in
´*R1* :== *CALL check-basic-just-sp* (´*G*, ´*dist*, ´*c*, ´*s*, ´*enum*, ´*pred*) ;;
´*R2* :== ´*s* < *ivertex-cnt* ´*G* ∧ ´*dist* ´*s* = *0* ;;
´*R3* :== *CALL no-path* (´*G*, ´*dist*, ´*enum*) ;;
´*R4* :== *CALL non-neg-cost* (´*G*, ´*c*) ;;
´*R* :== ´*R1* ∧ ´*R2* ∧ ´*R3* ∧ ´*R4*

end
theory *Check-Shortest-Path-Verification*
imports
Vcg
../Simpl-Verification/Check-Shortest-Path-Impl

begin

definition *no-loops* :: (*'a*, *'b*) *pre-digraph* ⇒ *bool* **where**
no-loops G ≡ ∀ *e* ∈ *arcs G*. *tail G e* ≠ *head G e*

definition *abs-IGraph* :: *IGraph* ⇒ (*nat*, *nat*) *pre-digraph* **where**
abs-IGraph G ≡ (| *verts* = {*0*..<*ivertex-cnt G*}, *arcs* = {*0*..<*iedge-cnt G*},
tail = *fst o iarcs G*, *head* = *snd o iarcs G* |)

lemma *verts-absI*[*simp*]: *verts* (*abs-IGraph G*) = {*0*..<*ivertex-cnt G*}
and *arcs-absI*[*simp*]: *arcs* (*abs-IGraph G*) = {*0*..<*iedge-cnt G*}
and *tail-absI*[*simp*]: *tail* (*abs-IGraph G*) *e* = *fst* (*iarcs G e*)
and *head-absI*[*simp*]: *head* (*abs-IGraph G*) *e* = *snd* (*iarcs G e*)
by (*auto simp*: *abs-IGraph-def*)

lemma *is-wellformed-inv-step*:
is-wellformed-inv G (*Suc i*) ⟷ *is-wellformed-inv G i*
∧ *fst* (*iarcs G i*) < *ivertex-cnt G* ∧ *snd* (*iarcs G i*) < *ivertex-cnt G*
by (*auto simp add*: *is-wellformed-inv-def less-Suc-eq*)

lemma (**in** *is-wellformed-impl*) *is-wellformed-spec*:
∀ *G*. Γ $\vdash_t$ {|´*G* = *G*|} ´*R* :== *PROC is-wellformed*(´*G*) {|´*R* = *is-wellformed-inv*
G (*iedge-cnt G*)|}
apply *vcg*
apply (*auto simp*: *is-wellformed-inv-step*)
apply (*auto simp*: *is-wellformed-inv-def*)
done

lemma *trian-inv-step*:
trian-inv G d c (*Suc i*) ⟷ *trian-inv G d c i*
∧ *d* (*snd* (*iarcs G i*)) ≤ *d* (*fst* (*iarcs G i*)) + *c i*

```
  by (auto simp: trian-inv-def less-Suc-eq)

lemma (in trian-impl) trian-spec:
  ∀ G d c. Γ ⊢t ⦃ ´G = G ∧ ´dist = d ∧ ´c = c⦄
    ´R :== PROC trian(´G, ´dist, ´c)
    ⦃ ´R = trian-inv G d c (iedge-cnt G)⦄
  apply vcg
  apply (auto simp add: trian-inv-step)
  apply (auto simp: trian-inv-def)
done

lemma just-inv-step:
  just-inv G d c s n p (Suc v) ⟷ just-inv G d c s n p v
    ∧ (v ≠ s ∧ n v ≠ ∞ ⟶
      (∃ e. e = the (p v) ∧ e < iedge-cnt G ∧
        v = snd (iarcs G e) ∧
        d v = d (fst (iarcs G e)) + ereal (c e) ∧
        n v = n (fst (iarcs G e)) + (enat 1)))
  by (auto simp: just-inv-def less-Suc-eq)

lemma just-inv-le:
  assumes j ≤ i just-inv G d c s n p i
  shows just-inv G d c s n p j
  using assms by (induct rule: dec-induct) (auto simp: just-inv-step)

lemma not-just-verts:
  fixes G R c d n p s v
  assumes v < ivertex-cnt G
  assumes v ≠ s ∧ n v ≠ ∞ ∧
        (iedge-cnt G ≤ the (p v) ∨
        snd (iarcs G (the (p v))) ≠ v ∨
        d v ≠
          d (fst (iarcs G (the (p v)))) + ereal (c (the (p v))) ∨
        n v ≠ n (fst (iarcs G (the (p v)))) + enat 1)
  shows ¬ just-inv G d c s n p (ivertex-cnt G)
proof (rule notI)
  assume jv: just-inv G d c s n p (ivertex-cnt G)
  have just-inv G d c s n p (Suc v)
    using just-inv-le[OF - jv] assms(1) by simp
  then have (v ≠ s ∧ n v ≠ ∞ ⟶
      (∃ e. e = the (p v) ∧ e < iedge-cnt G ∧
        v = snd (iarcs G e) ∧
        d v = d (fst (iarcs G e)) + ereal (c e) ∧
        n v = n (fst (iarcs G e)) + (enat 1)))
        by (auto simp: just-inv-step)
  with assms show False by force
qed

lemma (in just-impl) just-spec:
```

$\forall G\ d\ c\ s\ n\ p.$
$\Gamma\ \vdash_t \{\!|\,\acute{}G = G \wedge \acute{}dist = d \wedge$
$\acute{}c = c \wedge \acute{}s = s \wedge \acute{}enum = n \wedge \acute{}pred = p|\!\}$
$\acute{}R :== PROC\ just(\acute{}G, \acute{}dist, \acute{}c, \acute{}s, \acute{}enum, \acute{}pred)$
$\{\!|\ \acute{}R = just\text{-}inv\ \ G\ d\ c\ s\ n\ p\ (ivertex\text{-}cnt\ G)|\!\}$
apply *vcg*
apply (*auto simp*: *not-just-verts just-inv-step*)
apply (*simp add*: *just-inv-def*)
done

lemma *no-path-inv-step*:
$no\text{-}path\text{-}inv\ G\ d\ n\ (Suc\ v) \longleftrightarrow no\text{-}path\text{-}inv\ G\ d\ n\ v$
$\wedge\ (d\ v = \infty \longleftrightarrow n\ v = \infty)$
by (*auto simp add*: *no-path-inv-def less-Suc-eq*)

lemma (**in** *no-path-impl*) *no-path-spec*:
$\forall G\ d\ n.\ \Gamma \vdash_t \{\!|\ \acute{}G = G \wedge \acute{}dist = d \wedge \acute{}enum = n|\!\}$
$\acute{}R :== PROC\ no\text{-}path(\acute{}G, \acute{}dist, \acute{}enum)$
$\{\!|\ \acute{}R = no\text{-}path\text{-}inv\ G\ d\ n\ (ivertex\text{-}cnt\ G)|\!\}$
apply *vcg*
apply (*simp-all add*: *no-path-inv-step*)
apply (*auto simp*: *no-path-inv-def*)
done

lemma *non-neg-cost-inv-step*:
$non\text{-}neg\text{-}cost\text{-}inv\ G\ c\ (Suc\ i) \longleftrightarrow non\text{-}neg\text{-}cost\text{-}inv\ G\ c\ i$
$\wedge\ c\ i \geq 0$
by (*auto simp add*: *non-neg-cost-inv-def less-Suc-eq*)

lemma (**in** *non-neg-cost-impl*) *non-neg-cost-spec*:
$\forall G\ c.\ \Gamma \vdash_t \{\!|\ \acute{}G = G \wedge \acute{}c = c|\!\}$
$\acute{}R :== PROC\ non\text{-}neg\text{-}cost(\acute{}G, \acute{}c)$
$\{\!|\ \acute{}R = non\text{-}neg\text{-}cost\text{-}inv\ G\ c\ (iedge\text{-}cnt\ G)|\!\}$
apply *vcg*
apply (*simp-all add*: *non-neg-cost-inv-step*)
apply (*auto simp*: *non-neg-cost-inv-def*)
done

lemma *basic-just-sp-eq-invariants*:
$\bigwedge G\ dist\ c\ s\ enum\ pred.$
$basic\text{-}just\text{-}sp\text{-}pred\ (abs\text{-}IGraph\ G)\ dist\ c\ s\ enum\ pred \longleftrightarrow$
$(is\text{-}wellformed\text{-}inv\ G\ (iedge\text{-}cnt\ G) \wedge$
$dist\ s \leq 0 \wedge$
$trian\text{-}inv\ G\ dist\ c\ (iedge\text{-}cnt\ G) \wedge$
$just\text{-}inv\ G\ dist\ c\ s\ enum\ pred\ (ivertex\text{-}cnt\ G))$
proof $-$
fix *G d c s n p*
let *?aG* = *abs-IGraph G*
have $fin\text{-}digraph\ (abs\text{-}IGraph\ G) \longleftrightarrow is\text{-}wellformed\text{-}inv\ G\ (iedge\text{-}cnt\ G)$

unfolding *is-wellformed-inv-def fin-digraph-def fin-digraph-axioms-def wf-digraph-def no-loops-def*
by *auto*
moreover
have *trian-inv G d c (iedge-cnt G)* =
$(\forall e.\ e \in arcs\ (abs\text{-}IGraph\ G) \longrightarrow$
$(d\ (head\ ?aG\ e) \leq d\ (tail\ ?aG\ e) + ereal\ (c\ e)))$
by (*simp add*: *trian-inv-def*)
moreover
have *just-inv G d c s n p (ivertex-cnt G)* =
$(\forall v.\ v \in verts\ ?aG \longrightarrow$
$v \neq s \longrightarrow n\ v \neq \infty \longrightarrow$
$(\exists e \in arcs\ ?aG.\ e = the\ (p\ v) \wedge$
$v = head\ ?aG\ e \wedge$
$d\ v = d\ (tail\ ?aG\ e) + ereal\ (c\ e) \wedge$
$n\ v = n\ (tail\ ?aG\ e) + enat\ 1))$
unfolding *just-inv-def* **by** *fastforce*
ultimately
show *?thesis G d c s n p*
unfolding
basic-just-sp-pred-def
basic-just-sp-pred-axioms-def
basic-sp-def basic-sp-axioms-def
by *presburger*
qed

lemma (**in** *check-basic-just-sp-impl*) *check-basic-just-sp-imp-locale*:
$\forall\ G\ d\ c\ s\ n\ p\ .\ \Gamma \vdash_t \{\!|\ \acute{}G = G \wedge \acute{}dist = d \wedge \acute{}c = c \wedge \acute{}s = s \wedge \acute{}enum = n \wedge \acute{}pred = p\ |\!\}$
$\acute{}R$:== *PROC check-basic-just-sp* (´*G*, ´*dist*, ´*c*, ´*s*, ´*enum*, ´*pred*)
$\{\!|\ \acute{}R =$ *basic-just-sp-pred* (*abs-IGraph G*) *d c s n p* $|\!\}$
by *vcg* (*simp add*: *basic-just-sp-eq-invariants*)

lemma *shortest-path-non-neg-cost-eq-invariants*:
$\bigwedge G\ d\ c\ s\ n\ p\ .$
shortest-path-non-neg-cost-pred (*abs-IGraph G*) *d c s n p* $\longleftrightarrow$
(*is-wellformed-inv G* (*iedge-cnt G*) $\wedge$
$d\ s \leq 0 \wedge$
trian-inv G d c (*iedge-cnt G*) $\wedge$
just-inv G d c s n p (*ivertex-cnt G*) $\wedge$
$s <$ *ivertex-cnt G* $\wedge\ d\ s = 0 \wedge$
no-path-inv G d n (*ivertex-cnt G*) $\wedge$
non-neg-cost-inv G c (*iedge-cnt G*))
proof −
fix *G d c s n p*
let *?aG* = *abs-IGraph G*
have *no-path-inv G d n* (*ivertex-cnt G*) $\longleftrightarrow$
$(\forall v.\ v \in verts\ ?aG \longrightarrow (d\ v = \infty) = (n\ v = \infty))$

by (*simp add*: *no-path-inv-def*)
moreover
have *non-neg-cost-inv G c* (*iedge-cnt G*) $\longleftrightarrow$
($\forall$ *e*. *e* $\in$ *arcs ?aG* $\longrightarrow$ *0* $\leq$ *c e*)
by (*simp add*: *non-neg-cost-inv-def*)
ultimately
show *?thesis G d c s n p*
unfolding *shortest-path-non-neg-cost-pred-def*
shortest-path-non-neg-cost-pred-axioms-def
using *basic-just-sp-eq-invariants* **by** *simp*
qed

theorem (**in** *check-sp-impl*) *check-sp-eq-locale*:
$\forall$ *G d c s n p* . $\Gamma \vdash_t$ $\{\!|$ ´*G* = *G* $\wedge$ ´*dist* = *d* $\wedge$ ´*c* = *c* $\wedge$ ´*s* = *s* $\wedge$ ´*enum* = *n* $\wedge$ ´*pred* = *p* $|\!\}$
´*R* :== *PROC check-sp*(´*G*, ´*dist*, ´*c*, ´*s*, ´*enum*, ´*pred*)
$\{\!|$ ´*R* = *shortest-path-non-neg-cost-pred* (*abs-IGraph G*) *d c s n p*$|\!\}$
by *vcg* (*auto simp add*: *shortest-path-non-neg-cost-eq-invariants*)

lemma *shortest-path-non-neg-cost-imp-correct*:
$\bigwedge$*G d c s n p* .
shortest-path-non-neg-cost-pred (*abs-IGraph G*) *d c s n p* $\longrightarrow$
($\forall$ *v* $\in$ *verts* (*abs-IGraph G*).
d v = *wf-digraph*.μ (*abs-IGraph G*) *c s v*)
using *shortest-path-non-neg-cost-pred.correct-shortest-path-pred* **by** *fast*

theorem (**in** *check-sp-impl*) *check-sp-spec*:
$\forall$ *G d c s n p* . $\Gamma \vdash_t$ $\{\!|$ ´*G* = *G* $\wedge$ ´*dist* = *d* $\wedge$ ´*c* = *c* $\wedge$ ´*s* = *s* $\wedge$ ´*enum* = *n* $\wedge$ ´*pred* = *p* $|\!\}$
´*R* :== *PROC check-sp*(´*G*, ´*dist*, ´*c*, ´*s*, ´*enum*, ´*pred*)
$\{\!|$ ´*R* $\longrightarrow$ ($\forall$ *v* $\in$ *verts* (*abs-IGraph G*). *d v* = *wf-digraph*.μ (*abs-IGraph G*) *c s v*)$|\!\}$
using *shortest-path-non-neg-cost-eq-invariants*
shortest-path-non-neg-cost-imp-correct
by *vcg blast*

end

theory *Graph-Checker-Verification-Simpl*
imports
Check-Connected-Impl
Check-Connected-Verification
Check-Shortest-Path-Impl
Check-Shortest-Path-Verification

begin

end