Information Flow Control via Dependency Tracking

Benedikt Nordhoff

March 17, 2025

Abstract

We provide a characterisation of how information is propagated by program executions based on the tracking data and control dependencies within executions themselves. The characterisation might be used for deriving approximative safety properties to be targeted by static analyses or checked at runtime. We utilise a simple yet versatile control flow graph model as a program representation. As our model is not assumed to be finite it can be instantiated for a broad class of programs. The targeted security property is indistinguishable security where executions produce sequences of observations and only non-terminating executions are allowed to drop a tail of those.

A very crude approximation of our characterisation is slicing based on program dependence graphs, which we use as a minimal example and derive a corresponding soundness result.

For further details and applications refer to the authors upcoming dissertation.

Contents

1 Definitions

This section contains all necessary definitions of this development. Section 1.1 contains the structural definition of our program model which includes the security specification as well as abstractions of control flow and data. Executions of our program model are defined in section 1.1.1. Additional well-formedness properties are defined in section 1.1.2. Our security property is defined in section 1.2. Our characterisation of how information is propagated by executions of our program model is defined in section 1.3.5, for which the correctness result can be found in section 2.10. Section 1.3.6 contains an additional approximation of this characterisation whose correctness result can be found in section 2.11.

theory *IFC*
  imports *Main*
begin

1.1 Program Model

Our program model contains all necessary components for the remaining development and consists of:

record ('*n*, '*var*, '*val*, '*obs*) *ifc-problem* =
— A set of nodes representing program locations:
  nodes :: ‹'*n set*›
— An initial node where all executions start:
  entry :: ‹'*n*›
— A final node where executions can terminate:
  return :: ‹'*n*›
— An abstraction of control flow in the form of an edge relation:
  edges :: ‹('*n* × '*n*) *set*›
— An abstraction of variables written at program locations:
  writes :: ‹'*n* ⇒ '*var set*›
— An abstraction of variables read at program locations:
  reads :: ‹'*n* ⇒ '*var set*›
— A set of variables containing the confidential information in the initial state:
  hvars :: ‹'*var set*›
— A step function on location state pairs:
  step :: ‹('*n* × ('*var* ⇒ '*val*)) ⇒ ('*n* × ('*var* ⇒ '*val*))›
— An attacker model producing observations based on the reached state at certain locations:
  att :: ‹'*n* ⇀ (('*var* ⇒ '*val*) ⇒ '*obs*)›

We fix a program in the following in order to define the central concepts. The necessary well-formedness assumptions will be made in section 1.1.2.

locale *IFC-def* =
fixes *prob* :: ‹('*n*, '*var*, '*val*, '*obs*) *ifc-problem*›
begin

Some short hands to the components of the program which we will utilise exclusively in the following.

definition *nodes* **where** ‹*nodes* = *ifc-problem.nodes prob*›
definition *entry* **where** ‹*entry* = *ifc-problem.entry prob*›
definition *return* **where** ‹*return* = *ifc-problem.return prob*›
definition *edges* **where** ‹*edges* = *ifc-problem.edges prob*›
definition *writes* **where** ‹*writes* = *ifc-problem.writes prob*›
definition *reads* **where** ‹*reads* = *ifc-problem.reads prob*›
definition *hvars* **where** ‹*hvars* = *ifc-problem.hvars prob*›
definition *step* **where** ‹*step* = *ifc-problem.step prob*›
definition *att* **where** ‹*att* = *ifc-problem.att prob*›

The components of the step function for convenience.

definition *suc* **where** ‹$suc\ n\ \sigma = fst\ (step\ (n,\ \sigma))$›
definition *sem* **where** ‹$sem\ n\ \sigma = snd\ (step\ (n,\ \sigma))$›

lemma *step-suc-sem*: ‹$step\ (n,\sigma) = (suc\ n\ \sigma,\ sem\ n\ \sigma)$› **unfolding** *suc-def sem-def* **by** *auto*

1.1.1 Executions

In order to define what it means for a program to be well-formed, we first require concepts of executions and program paths.

The sequence of nodes visited by the execution corresponding to an input state.

definition *path* **where**
‹$path\ \sigma\ k = fst\ ((step\widehat{\ }\widehat{\ }k)\ (entry,\sigma))$›

The sequence of states visited by the execution corresponding to an input state.

definition *kth-state* (‹-$^{-}$› [*111*,*111*] *110*) **where**
‹$\sigma^k = snd\ ((step\widehat{\ }\widehat{\ }k)\ (entry,\sigma))$›

A predicate asserting that a sequence of nodes is a valid program path according to the control flow graph.

definition *is-path* **where**
‹$is\text{-}path\ \pi = (\forall\ n.\ (\pi\ n,\ \pi\ (Suc\ n)) \in edges)$›
end

1.1.2 Well-formed Programs

The following assumptions define our notion of valid programs.

locale *IFC* = *IFC-def* ‹*prob*› **for** *prob*:: ‹('n, 'var, 'val, 'out) *ifc-problem*› +
assumes *ret-is-node*[*simp*,*intro*]: ‹$return \in nodes$›
and *entry-is-node*[*simp*,*intro*]: ‹$entry \in nodes$›
and *writes*: ‹$\bigwedge\ v\ n.\ (\exists\sigma.\ \sigma\ v \neq sem\ n\ \sigma\ v) \Longrightarrow v \in writes\ n$›
and *writes-return*: ‹$writes\ return = \{\}$›
and *uses-writes*: ‹$\bigwedge\ n\ \sigma\ \sigma'.\ (\forall\ v \in reads\ n.\ \sigma\ v = \sigma'\ v) \Longrightarrow \forall\ v \in writes\ n.\ sem\ n\ \sigma\ v = sem\ n\ \sigma'\ v$›
and *uses-suc*: ‹$\bigwedge\ n\ \sigma\ \sigma'.\ (\forall\ v \in reads\ n.\ \sigma\ v = \sigma'\ v) \Longrightarrow suc\ n\ \sigma = suc\ n\ \sigma'$›
and *uses-att*: ‹$\bigwedge\ n\ f\ \sigma\ \sigma'.\ att\ n = Some\ f \Longrightarrow (\forall\ v \in reads\ n.\ \sigma\ v = \sigma'\ v) \Longrightarrow f\ \sigma = f\ \sigma'$›
and *edges-complete*[*intro*,*simp*]: ‹$\bigwedge m\ \sigma.\ m \in nodes \Longrightarrow (m,suc\ m\ \sigma) \in edges$›
and *edges-return* : ‹$\bigwedge x.\ (return,x) \in edges \Longrightarrow x = return$ ›
and *edges-nodes*: ‹$edges \subseteq nodes \times nodes$›
and *reaching-ret*: ‹$\bigwedge\ x.\ x \in nodes \Longrightarrow \exists\ \pi\ n.\ is\text{-}path\ \pi \wedge \pi\ 0 = x \wedge \pi\ n = return$›

1.2 Security

We define our notion of security, which corresponds to what Bohannon et al. [1] refer to as indistinguishable security. In order to do so we require notions of observations made by the attacker, termination and equivalence of input states.

context *IFC-def*
begin

1.2.1 Observations

The observation made at a given index within an execution.

definition *obsp* **where**

$\langle obsp\ \sigma\ k = (case\ att(path\ \sigma\ k)\ of\ Some\ f \Rightarrow Some\ (f\ (\sigma^k))\ |\ None \Rightarrow None)\rangle$

The indices within a path where an observation is made.

definition *obs-ids* :: $\langle (nat \Rightarrow 'n) \Rightarrow nat\ set\rangle$ **where**
$\langle obs\text{-}ids\ \pi = \{k.\ att\ (\pi\ k) \neq None\}\rangle$

A predicate relating an observable index to the number of observations made before.

definition *is-kth-obs* :: $\langle (nat \Rightarrow 'n) \Rightarrow nat \Rightarrow nat \Rightarrow bool\rangle$**where**
$\langle is\text{-}kth\text{-}obs\ \pi\ k\ i = (card\ (obs\text{-}ids\ \pi \cap \{..<i\}) = k \wedge att\ (\pi\ i) \neq\ None)\rangle$

The final sequence of observations made for an execution.

definition *obs* **where**
$\langle obs\ \sigma\ k = (if\ (\exists i.\ is\text{-}kth\text{-}obs\ (path\ \sigma)\ k\ i)\ then\ obsp\ \sigma\ (THE\ i.\ is\text{-}kth\text{-}obs\ (path\ \sigma)\ k\ i)\ else\ None)\rangle$

Comparability of observations.

definition *obs-prefix* :: $\langle (nat \Rightarrow 'obs\ option) \Rightarrow (nat \Rightarrow 'obs\ option) \Rightarrow bool\rangle$ (**infix** $\langle \lesssim \rangle$ *50*) **where**
$\langle a \lesssim b \equiv \forall\ i.\ a\ i \neq None \longrightarrow a\ i = b\ i\rangle$

definition *obs-comp* (**infix** $\langle \approx \rangle$ *50*) **where**
$\langle a \approx b \equiv a \lesssim b \vee b \lesssim a\rangle$

1.2.2 Low equivalence of input states

definition *restrict* (**infix** $\langle \upharpoonright \rangle$ *100*) **where**
$\langle f \upharpoonright U = (\lambda\ n.\ if\ n \in\ U\ then\ f\ n\ else\ undefined)\rangle$

Two input states are low equivalent if they coincide on the non high variables.

definition *loweq* (**infix** $\langle =_L \rangle$ *50*)
where $\langle \sigma =_L \sigma' = (\sigma \upharpoonright (-hvars) = \sigma' \upharpoonright (-hvars))\rangle$

1.2.3 Termination

An execution terminates iff it reaches the terminal node at any point.

definition *terminates* **where**
$\langle terminates\ \sigma \equiv \exists\ i.\ path\ \sigma\ i = return\rangle$

1.2.4 Security Property

The fixed program is secure if and only if for all pairs of low equivalent inputs the observation sequences are comparable and if the execution for an input state terminates then the observation sequence is not missing any observations.

definition *secure* **where**
$\langle secure \equiv \forall\ \sigma\ \sigma'.\ \sigma =_L \sigma' \longrightarrow (obs\ \sigma \approx obs\ \sigma' \wedge (terminates\ \sigma \longrightarrow obs\ \sigma' \lesssim obs\ \sigma))\rangle$

1.3 Characterisation of Information Flows

We now define our characterisation of information flows which tracks data and control dependencies within executions. To do so we first require some additional concepts.

1.3.1 Post Dominance

We utilise the post dominance relation in order to define control dependence.

The basic post dominance relation.

definition *is-pd* (**infix** ‹*pd*→› *50*) **where**
‹$y\ pd\!\rightarrow x \longleftrightarrow x \in nodes \land (\forall\ \pi\ n.\ is\text{-}path\ \pi \land \pi\ (0::nat) = x \land \pi\ n = return \longrightarrow (\exists k{\le}n.\ \pi\ k = y))$›

The immediate post dominance relation.

definition *is-ipd* (**infix** ‹*ipd*→› *50*)**where**
‹$y\ ipd\!\rightarrow x \longleftrightarrow x \neq y \land y\ pd\!\rightarrow x \land (\forall\ z.\ z{\neq}x \land z\ pd\!\rightarrow x \longrightarrow z\ pd\!\rightarrow y)$›

definition *ipd* **where**
‹$ipd\ x = (THE\ y.\ y\ ipd\!\rightarrow x)$›

The post dominance tree.

definition *pdt* **where**
‹$pdt = \{(x,y).\ x{\neq}y \land y\ pd\!\rightarrow x\}$›

1.3.2 Control Dependence

An index on an execution path is control dependent upon another if the path does not visit the immediate post domiator of the node reached by the smaller index.

definition *is-cdi* (‹- *cd*-→ -› [*51*,*51*,*51*]*50*) **where**
‹$i\ cd^{\pi}\!\rightarrow k \longleftrightarrow is\text{-}path\ \pi \land k < i \land \pi\ i \neq return \land (\forall\ j \in \{k..i\}.\ \pi\ j \neq ipd\ (\pi\ k))$›

The largest control dependency of an index is the immediate control dependency.

definition *is-icdi* (‹- *icd*-→ -› [*51*,*51*,*51*]*50*) **where**
‹$n\ icd^{\pi}\!\rightarrow n' \longleftrightarrow is\text{-}path\ \pi \land n\ cd^{\pi}\!\rightarrow n' \land (\forall\ m \in \{n'{<}..{<}n\}.\neg\ n\ cd^{\pi}\!\rightarrow m)$›

For the definition of the control slice, which we will define next, we require the uniqueness of the immediate control dependency.

lemma *icd-uniq*: **assumes** ‹$m\ icd^{\pi}\!\rightarrow n$› ‹ $m\ icd^{\pi}\!\rightarrow n'$› **shows** ‹$n = n'$›
proof −
  {
    fix $n\ n'$ **assume** ∗: ‹$m\ icd^{\pi}\!\rightarrow n$› ‹ $m\ icd^{\pi}\!\rightarrow n'$› ‹$n < n'$›
    have ‹$n'{<}m$› **using** ∗ **unfolding** *is-icdi-def is-cdi-def* **by** *auto*
    hence ‹$\neg\ m\ cd^{\pi}\!\rightarrow n'$› **using** ∗ **unfolding** *is-icdi-def* **by** *auto*
    with ∗(*2*) **have** ‹*False*› **unfolding** *is-icdi-def* **by** *auto*
  }
  thus *?thesis* **using** *assms* **by** (*metis linorder-neqE-nat*)
qed

1.3.3 Control Slice

We utilise the control slice, that is the sequence of nodes visited by the control dependencies of an index, to match indices between executions.

function *cs*:: ‹$(nat \Rightarrow 'n) \Rightarrow nat \Rightarrow 'n\ list$› (‹*cs*- -› [*51*,*70*] *71*) **where**
‹$cs^{\pi}\ n = (if\ (\exists\ m.\ n\ icd^{\pi}\!\rightarrow m)\ then\ (cs\ \pi\ (THE\ m.\ n\ icd^{\pi}\!\rightarrow m))@[\pi\ n]\ else\ [\pi\ n])$›
by *pat-completeness auto*
termination ‹*cs*› **proof**
  show ‹*wf* (*measure snd*)› **by** *simp*
  fix $\pi\ n$

define m **where** ‹$m == (The\ (is\text{-}icdi\ n\ \pi))$›
assume ‹$Ex\ (is\text{-}icdi\ n\ \pi)$›
hence ‹$n\ icd^{\pi}\!\rightarrow\ m$› **unfolding** *m-def* **by** (*metis* (*full-types*) *icd-uniq theI'*)
hence ‹$m < n$› **unfolding** *is-icdi-def is-cdi-def* **by** *simp*
thus ‹$((\pi,\ The\ (is\text{-}icdi\ n\ \pi)),\ \pi,\ n) \in measure\ snd$› **by** (*metis in-measure m-def snd-conv*)
qed

inductive *cs-less* (**infix** ‹$\prec$› *50*) **where**
‹$length\ xs < length\ ys \Longrightarrow take\ (length\ xs)\ ys = xs \ \Longrightarrow xs \prec ys$›

definition *cs-select* (**infix** ‹¡› *50*) **where**
‹$\pi¡xs = (THE\ k.\ cs^{\pi}\ k = xs)$›

1.3.4 Data Dependence

Data dependence is defined straight forward. An index is data dependent upon another, if the index reads a variable written by the earlier index and the variable in question has not been written by any index in between.

definition *is-ddi* (‹- $dd^{-,-}\!\rightarrow$ -› [*51*,*51*,*51*,*51*] *50*) **where**
‹$n\ dd^{\pi,v}\!\rightarrow\ m \longleftrightarrow is\text{-}path\ \pi \land m < n \land v \in reads\ (\pi\ n) \cap (writes\ (\pi\ m)) \land (\forall\ l \in \{m{<}..{<}n\}.\ v \notin writes\ (\pi\ l))$›

1.3.5 Characterisation via Critical Paths

With the above we define the set of critical paths which as we will prove characterise the matching points in executions where diverging data is read.

inductive-set *cp* **where**

— Any pair of low equivalent input states and indices where a diverging high variable is first read is critical.

‹$[\![\sigma =_L \sigma';$
$cs^{path\ \sigma}\ n = cs^{path\ \sigma'}\ n';$
$h \in reads(path\ \sigma\ n);$
$(\sigma^{n})\ h \neq (\sigma'^{m'})\ h;$
$\forall\ k{<}n.\ h{\notin}writes(path\ \sigma\ k);$
$\forall\ k'{<}n'.\ h{\notin}writes(path\ \sigma'\ k')$
$]\!] \Longrightarrow ((\sigma,n),(\sigma',n')) \in cp$› |

— If from a pair of critical indices in two executions there exist data dependencies from both indices to a pair of matching indices where the variable diverges, the later pair of indices is critical.

‹$[\![((\sigma,k),(\sigma',k')) \in cp;$
$n\ dd^{path\ \sigma,v}\!\rightarrow\ k;$
$n'\ dd^{path\ \sigma',v}\!\rightarrow\ k';$
$cs^{path\ \sigma}\ n = cs^{path\ \sigma'}\ n';$
$(\sigma^{n})\ v \neq (\sigma'^{m'})\ v$
$]\!] \Longrightarrow ((\sigma,n),(\sigma',n')) \in cp$› |

— If from a pair of critical indices the executions take different branches and one of the critical indices is a control dependency of an index that is data dependency of a matched index where diverging data is read and the variable in question is not written by the other execution after the executions first reached matching indices again, then the later matching pair of indices is critical.

‹⟦$((\sigma,k),(\sigma',k')) \in cp$;
$n\ dd^{path\ \sigma,v}\rightarrow l$;
$l\ cd^{path\ \sigma}\rightarrow k$;
$cs^{path\ \sigma}\ n = cs^{path\ \sigma'}\ n'$;
$path\ \sigma\ (Suc\ k) \neq path\ \sigma'\ (Suc\ k')$;
$(\sigma^n)\ v \neq (\sigma'^{m'})\ v$;
$\forall j' \in \{(LEAST\ i'.\ k' < i' \wedge (\exists i.\ cs^{path\ \sigma}\ i = cs^{path\ \sigma'}\ i'))..<n'\}.\ v \notin writes\ (path\ \sigma'\ j')$
⟧ $\Longrightarrow ((\sigma,n),(\sigma',n')) \in cp$› |

— The relation is symmetric.

‹⟦$((\sigma,k),(\sigma',k')) \in cp$⟧ $\Longrightarrow ((\sigma',k'),(\sigma,k)) \in cp$›

Based on the set of critical paths, the critical observable paths are those that either directly reach observable nodes or are diverging control dependencies of an observable index.

inductive-set *cop* **where**
‹⟦$((\sigma,n),(\sigma',n')) \in cp$;
$path\ \sigma\ n \in dom\ att$
⟧ $\Longrightarrow ((\sigma,n),(\sigma',n')) \in cop$› |

‹⟦$((\sigma,k),(\sigma',k')) \in cp$;
$n\ cd^{path\ \sigma}\rightarrow k$;
$path\ \sigma\ (Suc\ k) \neq path\ \sigma'\ (Suc\ k')$;
$path\ \sigma\ n \in dom\ att$
⟧ $\Longrightarrow ((\sigma,n),(\sigma',k')) \in cop$›

1.3.6 Approximation via Single Critical Paths

For applications we also define a single execution approximation.

definition *is-dcdi-via* (‹- $dcd^{-,-}\rightarrow$ - via - -› [51,51,51,51,51,51] 50) **where**
‹$n\ dcd^{\pi,v}\rightarrow m\ via\ \pi'\ m' = (is\text{-}path\ \pi \wedge m < n \wedge (\exists\ l'\ n'.\ cs^{\pi}\ m = cs^{\pi'}\ m' \wedge cs^{\pi}\ n = cs^{\pi'}\ n' \wedge n'\ dd^{\pi',v}\rightarrow l' \wedge l'\ cd^{\pi'}\rightarrow m') \wedge (\forall\ l \in \{m..<n\}.\ v \notin writes(\pi\ l)))$›

inductive-set *scp* **where**
‹⟦$h \in hvars$; $h \in reads\ (path\ \sigma\ n)$; $(\forall\ k<n.\ h \notin writes(path\ \sigma\ k))$⟧ $\Longrightarrow (path\ \sigma,n) \in scp$› |
‹⟦$(\pi,m) \in scp$; $n\ cd^{\pi}\rightarrow m$⟧ $\Longrightarrow (\pi,n) \in scp$›|
‹⟦$(\pi,m) \in scp$; $n\ dd^{\pi,v}\rightarrow m$⟧ $\Longrightarrow (\pi,n) \in scp$›|
‹⟦$(\pi,m) \in scp$; $(\pi',m') \in scp$; $n\ dcd^{\pi,v}\rightarrow m\ via\ \pi'\ m'$⟧ $\Longrightarrow (\pi,n) \in scp$›

inductive-set *scop* **where**
‹⟦$(\pi,n) \in scp$; $\pi\ n \in dom\ att$⟧ $\Longrightarrow (\pi,n) \in scop$›

1.3.7 Further Definitions

The following concepts are utilised by the proofs.

inductive *contradicts* (**infix** ‹𝔠› 50) **where**
‹⟦$cs^{\pi'}\ k' \prec cs^{\pi}\ k$; $\pi = path\ \sigma$; $\pi' = path\ \sigma'$; $\pi\ (Suc\ (\pi¡cs^{\pi'}\ k')) \neq \pi'\ (Suc\ k')$⟧ $\Longrightarrow (\sigma',\ k')$ 𝔠 $(\sigma,\ k)$›|
‹⟦$cs^{\pi'}\ k' = cs^{\pi}\ k$; $\pi = path\ \sigma$; $\pi' = path\ \sigma'$; $\sigma^k \upharpoonright (reads\ (\pi\ k)) \neq \sigma'^{k'} \upharpoonright (reads\ (\pi\ k))$⟧ $\Longrightarrow (\sigma',k')$ 𝔠 (σ,k)›

definition *path-shift* (**infixl** ‹«› 51) **where**
[*simp*]: ‹$\pi«m = (\lambda\ n.\ \pi\ (m+n))$›

definition *path-append* :: ‹$(nat \Rightarrow 'n) \Rightarrow nat \Rightarrow (nat \Rightarrow 'n) \Rightarrow (nat \Rightarrow 'n)$› (‹- @$^-$ -› [0,0,999] 51) **where**

[*simp*]: ‹π @^{m} π' = (λn.(*if* $n \le m$ *then* π n *else* π' ($n-m$)))›

definition *eq-up-to* :: ‹(*nat* ⇒ ′*n*) ⇒ *nat* ⇒ (*nat* ⇒ ′*n*) ⇒ *bool*› (‹- =_ -› [*55*,*55*,*55*] *50*) **where**
‹$\pi =_k \pi'$ = ($\forall$ $i \le k$. π i = π' i)›

end

2 Proofs

2.1 Miscellaneous Facts

lemma *option-neq-cases*: **assumes** ‹$x \ne y$› **obtains** (*none1*) *a* **where** ‹x = *None*› ‹y = *Some a*› | (*none2*) *a* **where** ‹x = *Some a*› ‹y = *None*› | (*some*) *a b* **where** ‹x = *Some a*› ‹y = *Some b*› ‹$a \ne b$› **using** *assms* **by** *fastforce*

lemmas *nat-sym-cases*[*case-names less sym eq*] = *linorder-less-wlog*

lemma *mod-bound-instance*: **assumes** ‹j < (i::*nat*)› **obtains** j' **where** ‹$k < j'$› **and** ‹j' *mod* i = j› **proof** −
  have ‹k < *Suc* $k * i + j$› **using** *assms less-imp-Suc-add* **by** *fastforce*
  moreover
  have ‹(*Suc* $k * i + j$) *mod* i = j› **by** (*metis assms mod-less mod-mult-self3*)
  ultimately show ‹*thesis*› **using** *that* **by** *auto*
qed

lemma *list-neq-prefix-cases*: **assumes** ‹*ls* ≠ *ls*′› **and** ‹*ls* ≠ *Nil*› **and** ‹*ls*′ ≠ *Nil*›
  obtains (*diverge*) *xs x x*′ *ys ys*′ **where** ‹*ls* = *xs*@[*x*]@*ys*› ‹*ls*′ = *xs*@[*x*′]@*ys*′› ‹*x* ≠ *x*′› |
   (*prefix1*) *xs* **where** ‹*ls* = *ls*′@*xs*› **and** ‹*xs* ≠ *Nil*› |
   (*prefix2*) *xs* **where** ‹*ls*@*xs* = *ls*′› **and** ‹*xs* ≠ *Nil*›
using *assms* **proof** (*induct* ‹*length ls*› *arbitrary*: ‹*ls*› ‹*ls*′› *rule*: *less-induct*)
  case (*less ls ls*′)
  obtain *z zs z*′ *zs*′ **where**
  lz: ‹*ls* = *z*#*zs*› ‹*ls*′ = *z*′#*zs*′› **by** (*metis list.exhaust less*(*6*,*7*))
  show ‹*?case*› **proof** *cases*
    assume *zz*: ‹*z* = *z*′›
    hence *zsz*: ‹*zs* ≠ *zs*′› **using** *less*(*5*) *lz* **by** *auto*
    have *lenz*: ‹*length zs* < *length ls*› **using** *lz* **by** *auto*
    show ‹*?case*› **proof**(*cases* ‹*zs* = *Nil*›)
      assume *zs*: ‹*zs* = *Nil*›
      hence ‹*zs*′ ≠ *Nil*› **using** *zsz* **by** *auto*
      moreover
      have ‹*ls*@*zs*′ = *ls*′› **using** *zs lz zz* **by** *auto*
      ultimately
      show ‹*thesis*› **using** *less*(*4*) **by** *blast*
    next
      assume *zs*: ‹*zs* ≠ *Nil*›
      show ‹*thesis*› **proof** (*cases* ‹*zs*′ = *Nil*›)
        assume ‹*zs*′ = *Nil*›
        hence ‹*ls* = *ls*′@*zs*› **using** *lz zz* **by** *auto*
        thus ‹*thesis*› **using** *zs less*(*3*) **by** *blast*
      next
        assume *zs*′: ‹*zs*′ ≠ *Nil*›
        { fix *xs x ys x*′ *ys*′
          assume ‹*zs* = *xs* @ [*x*] @ *ys*› ‹*zs*′ = *xs* @ [*x*′] @ *ys*′› **and** *xx*: ‹*x* ≠ *x*′›
          hence ‹*ls* = (*z*#*xs*) @ [*x*] @ *ys*› ‹*ls*′ = (*z*#*xs*) @ [*x*′] @ *ys*′› **using** *lz zz* **by** *auto*
          hence ‹*thesis*› **using** *less*(*2*) *xx* **by** *blast*
        } note ∗ = *this*

{ **fix** *xs*
assume ‹$zs = zs'$ @ xs› **and** *xs*: ‹$xs \neq []$›
hence ‹$ls = ls'$ @ xs› **using** *lz zz* **by** *auto*
hence ‹*thesis*› **using** *xs less*(*3*) **by** *blast*
} **note** $** = this$
{ **fix** *xs*
assume ‹$zs@xs = zs'$› **and** *xs*: ‹$xs \neq []$›
hence ‹$ls@xs = ls'$› **using** *lz zz* **by** *auto*
hence ‹*thesis*› **using** *xs less*(*4*) **by** *blast*
} **note** $*** = this$
have ‹$(\bigwedge xs\ x\ ys\ x'\ ys'.\ zs = xs$ @ $[x]$ @ $ys \Longrightarrow zs' = xs$ @ $[x']$ @ $ys' \Longrightarrow x \neq x' \Longrightarrow thesis) \Longrightarrow$
$(\bigwedge xs.\ zs = zs'$ @ $xs \Longrightarrow xs \neq [] \Longrightarrow thesis) \Longrightarrow$
$(\bigwedge xs.\ zs$ @ $xs = zs' \Longrightarrow xs \neq [] \Longrightarrow thesis) \Longrightarrow thesis$›
using *less*(*1*)[*OF lenz - - - zsz zs zs'*] **.**
thus ‹*thesis*› **using** $*$ $**$ $***$ **by** *blast*
qed
qed
next
assume ‹$z \neq z'$›
moreover
have ‹$ls = []@[z]@zs$› ‹$ls' = []@[z']@zs'$› **using** *lz* **by** *auto*
ultimately show ‹*thesis*› **using** *less*(*2*) **by** *blast*
qed
qed

lemma *three-cases*: **assumes** ‹$A \vee B \vee C$› **obtains** ‹A› | ‹B› | ‹C› **using** *assms* **by** *auto*

lemma *insort-greater*: ‹$\forall\ x \in set\ ls.\ x < y \Longrightarrow insort\ y\ ls = ls@[y]$› **by** (*induction* ‹*ls*›,*auto*)

lemma *insort-append-first*: **assumes** ‹$\forall\ y \in set\ ys.\ x \leq y$› **shows** ‹*insort x* (*xs*@*ys*) = *insort x xs* @ *ys*› **using** *assms* **by** (*induction* ‹*xs*›,*auto*,*metis insort-is-Cons*)

lemma *sorted-list-of-set-append*: **assumes** ‹*finite xs*› ‹*finite ys*› ‹$\forall\ x \in xs.\ \forall\ y \in ys.\ x < y$› **shows** ‹*sorted-list-of-set* ($xs \cup ys$) = *sorted-list-of-set xs* @ (*sorted-list-of-set ys*)›
using *assms*(*1*,*3*) **proof** (*induction* ‹*xs*›)
case *empty* **thus** ‹*?case*› **by** *simp*
next
case (*insert x xs*)
hence *iv*: ‹*sorted-list-of-set* ($xs \cup ys$) = *sorted-list-of-set xs* @ *sorted-list-of-set ys*› **by** *blast*
have *le*: ‹$\forall\ y \in set$ (*sorted-list-of-set ys*). $x < y$› **using** *insert*(*4*) *assms*(*2*) *sorted-list-of-set* **by** *auto*
have ‹*sorted-list-of-set* (*insert x xs* $\cup$ *ys*) = *sorted-list-of-set* (*insert x* ($xs \cup ys$))› **by** *auto*
also
have ‹... = *insort x* (*sorted-list-of-set* ($xs \cup ys$))› **by** (*metis Un-iff assms*(*2*) *finite-Un insert.hyps*(*1*) *insert.hyps*(*2*) *insert.prems insertI1 less-irrefl sorted-list-of-set-insert*)
also
have ‹... = *insort x* (*sorted-list-of-set xs* @ *sorted-list-of-set ys*)› **using** *iv* **by** *simp*
also
have ‹... = *insort x* (*sorted-list-of-set xs*) @ *sorted-list-of-set ys*› **by** (*metis le insort-append-first less-le-not-le*)
also
have ‹... = *sorted-list-of-set* (*insert x xs*) @ *sorted-list-of-set ys*› **using** *sorted-list-of-set-insert*[*OF insert*(*1*),*of* ‹*x*›] *insert*(*2*) **by** *auto*
finally
show ‹*?case*› **.**
qed

lemma *filter-insort*: ‹*sorted xs* $\Longrightarrow$ *filter P* (*insort x xs*) = (*if P x then insort x* (*filter P xs*) *else filter P xs*)›

by (*induction* ‹*xs*›, *simp*) (*metis filter-insort filter-insort-triv map-ident*)

lemma *filter-sorted-list-of-set*: **assumes** ‹*finite xs*› **shows** ‹*filter P* (*sorted-list-of-set xs*) = *sorted-list-of-set* {*x* ∈ *xs*. *P x*}› **using** *assms* **proof**(*induction* ‹*xs*›)
  case *empty* **thus** ‹*?case*› **by** *simp*
next
  case (*insert x xs*)
  have ∗: ‹*set* (*sorted-list-of-set xs*) = *xs*› ‹*sorted* (*sorted-list-of-set xs*)› ‹*distinct* (*sorted-list-of-set xs*)› **by** (*auto simp add*: *insert.hyps*(*1*))
  have ∗∗: ‹*P x* ⟹ {*y* ∈ *insert x xs*. *P y*} = *insert x* {*y* ∈ *xs*. *P y*}› **by** *auto*
  have ∗∗∗: ‹¬ *P x* ⟹ {*y* ∈ *insert x xs*. *P y*} = {*y* ∈ *xs*. *P y*}› **by** *auto*
  note *filter-insort*[*OF* ∗(*2*),*of* ‹*P*› ‹*x*›] *sorted-list-of-set-insert*[*OF insert*(*1*), *of* ‹*x*›] *insert*(*2*,*3*) ∗∗ ∗∗∗
  thus ‹*?case*› **by** (*metis* (*mono-tags*) ∗(*1*) *List.finite-set distinct-filter distinct-insort distinct-sorted-list-of-set set-filter sorted-list-of-set-insert*)
qed

lemma *unbounded-nat-set-infinite*: **assumes** ‹∀ (*i*::*nat*). ∃ *j*≥*i*. *j* ∈ *A*› **shows** ‹¬ *finite A*› **using** *assms*
by (*metis finite-nat-set-iff-bounded-le not-less-eq-eq*)

lemma *infinite-ascending*: **assumes** *nf*: ‹¬ *finite* (*A*::*nat set*)› **obtains** *f* **where** ‹*range f* = *A*› ‹∀ *i*. *f i* < *f* (*Suc i*)› **proof**
  let ‹*?f*› = ‹λ *i*. (*LEAST a*. *a* ∈ *A* ∧ *card* (*A* ∩ {..<*a*}) = *i*)›
  { fix *i*
    obtain *a* **where** ‹*a* ∈ *A*› ‹*card* (*A* ∩ {..<*a*}) = *i*›
    proof (*induction* ‹*i*› *arbitrary*: ‹*thesis*›)
      case *0*
      let ‹*?a0*› = ‹(*LEAST a*. *a* ∈ *A*)›
      have ‹*?a0* ∈ *A*› **by** (*metis LeastI empty-iff finite.emptyI nf set-eq-iff*)
      moreover
      have ‹⋀*b*. *b* ∈ *A* ⟹ *?a0* ≤ *b*› **by** (*metis Least-le*)
      hence ‹*card* (*A* ∩ {..<*?a*}) = *0*› **by** *force*
      ultimately
      show ‹*?case*› **using** *0* **by** *blast*
    next
      case (*Suc i*)
      obtain *a* **where** *aa*: ‹*a* ∈ *A*› **and** *card*: ‹*card* (*A* ∩ {..<*a*}) = *i*› **using** *Suc.IH* **by** *metis*
      have *nf'*: ‹~ *finite* (*A* − {..*a*})› **using** *nf* **by** *auto*
      let ‹*?b*› = ‹*LEAST b*. *b* ∈ *A* − {..*a*}›
      have *bin*: ‹*?b* ∈ *A*−{..*a*}› **by** (*metis LeastI empty-iff finite.emptyI nf' set-eq-iff*)
      have *le*: ‹⋀*c*. *c* ∈ *A*−{..*a*} ⟹ *?b* ≤ *c*› **by** (*metis Least-le*)
      have *ab*: ‹*a* < *?b*› **using** *bin* **by** *auto*
      have ‹⋀ *c*. *c* ∈ *A* ⟹ *c* < *?b* ⟹ *c* ≤ *a*› **using** *le* **by** *force*
      hence ‹*A* ∩ {..<*?b*} = *insert a* (*A* ∩ {..<*a*})› **using** *bin ab aa* **by** *force*
      hence ‹*card* (*A* ∩{..<*?b*}) = *Suc i*› **using** *card* **by** *auto*
      thus ‹*?case*› **using** *Suc.prems bin* **by** *auto*
    qed
    note ‹⋀ *thesis*. ((⋀*a*. *a* ∈ *A* ⟹ *card* (*A* ∩ {..<*a*}) = *i* ⟹ *thesis*) ⟹ *thesis*)›
  }
  note *ex* = *this*

  {
    fix *i*
    obtain *a* **where** *a*: ‹*a* ∈ *A* ∧ *card* (*A* ∩{..<*a*}) = *i*› **using** *ex* **by** *blast*
    have *ina*: ‹*?f i* ∈ *A*› **and** *card*: ‹*card* (*A* ∩{..<*?f i*}) = *i*› **using** *LeastI*[*of* ‹λ *a*. *a* ∈ *A* ∧ *card* (*A* ∩{..<*a*}) = *i*› ‹*a*›, *OF a*] **by** *auto*
    obtain *b* **where** *b*: ‹*b* ∈ *A* ∧ *card* (*A* ∩{..<*b*}) = *Suc i*› **using** *ex* **by** *blast*

have *inab*: ‹?f (Suc i) ∈ A› **and** *cardb*: ‹card (A ∩{..<?f (Suc i)}) = Suc i› **using** *LeastI*[*of* ‹λ a. a ∈ A ∧ card (A ∩{..<a}) = Suc i› ‹b›, *OF b*] **by** *auto*
have ‹?f i < ?f (Suc i)› **proof** (*rule ccontr*)
assume ‹¬ ?f i < ?f (Suc i)›
hence ‹A ∩{..<?f (Suc i)} ⊆ A ∩{..<?f i}› **by** *auto*
moreover have ‹finite (A ∩{..<?f i})› **by** *auto*
ultimately have ‹card(A ∩{..<?f (Suc i)}) ≤ card (A ∩{..<?f i})› **by** (*metis* (*erased*, *lifting*) *card-mono*)
thus ‹False› **using** *card cardb* **by** *auto*
qed
note *this ina*
}
note *b* = *this*
thus ‹∀ i. ?f i < ?f (Suc i)› **by** *auto*
have ∗: ‹range ?f ⊆ A› **using** *b* **by** *auto*
moreover
{
fix *a* **assume** *ina*: ‹a ∈ A›
let ‹?i› = ‹card (A ∩ {..<a})›
obtain *b* **where** *b*: ‹b ∈ A ∧ card (A ∩{..<b}) = ?i› **using** *ex* **by** *blast*
have *inab*: ‹?f ?i ∈ A› **and** *cardb*: ‹card (A ∩{..<?f ?i}) = ?i› **using** *LeastI*[*of* ‹λ a. a ∈ A ∧ card (A ∩{..<a}) = ?i› ‹b›, *OF b*] **by** *auto*
have *le*: ‹?f ?i ≤ a› **using** *Least-le*[*of* ‹λ a. a ∈ A ∧ card (A ∩{..<a}) = ?i› ‹a›] *ina* **by** *auto*
have ‹a = ?f ?i› **proof** (*rule ccontr*)
have *fin*: ‹finite (A ∩ {..<a})› **by** *auto*
assume ‹a ≠ ?f ?i›
hence ‹?f ?i < a› **using** *le* **by** *simp*
hence ‹?f ?i ∈ A ∩ {..<a}› **using** *inab* **by** *auto*
moreover
have ‹A ∩ {..<?f ?i} ⊆ A ∩ {..<a}› **using** *le* **by** *auto*
hence ‹A ∩ {..<?f ?i} = A ∩ {..<a}› **using** *cardb card-subset-eq*[*OF fin*] **by** *auto*
ultimately
show ‹False› **by** *auto*
qed
hence ‹a ∈ range ?f› **by** *auto*
}
hence ‹A ⊆ range ?f› **by** *auto*
ultimately show ‹range ?f = A› **by** *auto*
qed

lemma *mono-ge-id*: ‹∀ i. f i < f (Suc i) ⟹ i ≤ f i›
apply (*induction* ‹i›,*auto*)
by (*metis not-le not-less-eq-eq order-trans*)

lemma *insort-map-mono*: **assumes** *mono*: ‹∀ n m. n < m ⟶ f n < f m› **shows** ‹map f (insort n ns) = insort (f n) (map f ns)›
apply (*induction* ‹ns›)
apply *auto*
apply (*metis not-less not-less-iff-gr-or-eq mono*)
apply (*metis antisym-conv1 less-imp-le mono*)
apply (*metis mono not-less*)
by (*metis mono not-less*)

lemma *sorted-list-of-set-map-mono*: **assumes** *mono*: ‹∀ n m. n < m ⟶ f n < f m› **and** *fin*: ‹finite A›
shows ‹map f (sorted-list-of-set A) = sorted-list-of-set (f‘A)›
using *fin* **proof** (*induction*)
case *empty* **thus** ‹?case› **by** *simp*

next
case (*insert x A*)
have [*simp*]:‹*sorted-list-of-set* (*insert x A*) = *insort x* (*sorted-list-of-set A*)› **using** *insert sorted-list-of-set-insert* **by** *simp*
have ‹*f* ‘ *insert x A* = *insert* (*f x*) (*f* ‘ *A*)› **by** *auto*
moreover
have ‹*f x* ∉ *f*‘*A*› **apply** (*rule ccontr*) **using** *insert*(*2*) *mono* **apply** *auto* **by** (*metis insert.hyps*(*2*) *mono neq-iff*)
ultimately
have ‹*sorted-list-of-set* (*f* ‘ *insert x A*) = *insort* (*f x*) (*sorted-list-of-set* (*f*‘*A*))› **using** *insert*(*1*) *sorted-list-of-set-insert* **by** *simp*
also
have ‹. . . = *insort* (*f x*) (*map f* (*sorted-list-of-set A*))› **using** *insert.IH* **by** *auto*
also have ‹. . . = *map f* (*insort x* (*sorted-list-of-set A*))› **using** *insort-map-mono*[*OF mono*] **by** *auto*
finally
show ‹*map f* (*sorted-list-of-set* (*insert x A*)) = *sorted-list-of-set* (*f* ‘ *insert x A*)› **by** *simp*
qed

lemma *GreatestIB*:
fixes *n* :: ‹*nat*› **and** *P*
assumes *a*:‹∃ *k*≤*n*. *P k*›
shows *GreatestBI*: ‹*P* (*GREATEST k*. *k*≤*n* ∧ *P k*)› **and** *GreatestB*: ‹(*GREATEST k*. *k*≤*n* ∧ *P k*) ≤ *n*›
proof −
show ‹*P* (*GREATEST k*. *k*≤*n* ∧ *P k*)› **using** *GreatestI-ex-nat*[*OF assms*] **by** *auto*
show ‹(*GREATEST k*. *k*≤*n* ∧ *P k*) ≤ *n*› **using** *GreatestI-ex-nat*[*OF assms*] **by** *auto*
qed

lemma *GreatestB-le*:
fixes *n* :: ‹*nat*›
assumes ‹*x*≤*n*› **and** ‹*P x*›
shows ‹*x* ≤ (*GREATEST k*. *k*≤*n* ∧ *P k*)›
proof −
have ∗: ‹∀ *y*. *y*≤*n* ∧ *P y* ⟶ *y*<*Suc n*› **by** *auto*
then show ‹*x* ≤ (*GREATEST k*. *k*≤*n* ∧ *P k*)› **using** *assms* **by** (*blast intro*: *Greatest-le-nat*)
qed

lemma *LeastBI-ex*: **assumes** ‹∃ *k* ≤ *n*. *P k*› **shows** ‹*P* (*LEAST k*::′*c*::*wellorder*. *P k*)› **and** ‹(*LEAST k*. *P k*) ≤ *n*›
proof −
from *assms* **obtain** *k* **where** *k*: *k* ≤ *n P k* **by** *blast*
thus ‹*P* (*LEAST k*. *P k*)› **using** *LeastI*[*of* ‹*P*› ‹*k*›] **by** *simp*
show ‹(*LEAST k*. *P k*) ≤ *n*› **using** *Least-le*[*of* ‹*P*› ‹*k*›] *k* **by** *auto*
qed

lemma *allB-atLeastLessThan-lower*: **assumes** ‹(*i*::*nat*) ≤ *j*› ‹∀ *x*∈{*i*..<*n*}. *P x*› **shows** ‹∀ *x*∈{*j*..<*n*}. *P x*›
proof
fix *x* **assume** ‹*x*∈{*j*..<*n*}› **hence** ‹*x*∈{*i*..<*n*}› **using** *assms*(*1*) **by** *simp*
thus ‹*P x*› **using** *assms*(*2*) **by** *auto*
qed

2.2 Facts about Paths

context *IFC*
begin

lemma *path0*: ‹*path σ 0* = *entry*› **unfolding** *path-def* **by** *auto*

```
lemma path-in-nodes[intro]: ‹path σ k ∈ nodes› proof (induction ‹k›)
  case (Suc k)
  hence ‹⋀ σ'. (path σ k, suc (path σ k) σ') ∈ edges› by auto
  hence ‹(path σ k, path σ (Suc k)) ∈ edges› unfolding path-def
    by (metis suc-def comp-apply funpow.simps(2) prod.collapse)
  thus ‹?case› using edges-nodes by force
qed (auto simp add: path-def)

lemma path-is-path[simp]: ‹is-path (path σ)› unfolding is-path-def path-def using step-suc-sem apply auto
by (metis path-def suc-def edges-complete path-in-nodes prod.collapse)

lemma term-path-stable: assumes ‹is-path π› ‹π i = return› and le: ‹i ≤ j› shows ‹π j = return›
using le proof (induction ‹j›)
  case (Suc j)
  show ‹?case› proof cases
    assume ‹i≤j›
    hence ‹π j = return› using Suc by simp
    hence ‹(return, π (Suc j)) ∈ edges› using assms(1) unfolding is-path-def by metis
    thus ‹π (Suc j) = return› using edges-return by auto
  next
    assume ‹¬ i ≤ j›
    hence ‹Suc j = i› using Suc by auto
    thus ‹?thesis› using assms(2) by auto
  qed
next
  case 0 thus ‹?case› using assms by simp
qed

lemma path-path-shift: assumes ‹is-path π› shows ‹is-path (π«m)›
using assms unfolding is-path-def by simp

lemma path-cons: assumes ‹is-path π› ‹is-path π'› ‹π m = π' 0› shows ‹is-path (π @^m π')›
unfolding is-path-def proof(rule,cases)
  fix n assume ‹m < n› thus ‹((π @^m π') n, (π @^m π') (Suc n)) ∈ edges›
    using assms(2) unfolding is-path-def path-append-def
    by (auto,metis Suc-diff-Suc diff-Suc-Suc less-SucI)
next
  fix n assume ∗: ‹¬ m < n› thus ‹((π @^m π') n, (π @^m π') (Suc n)) ∈ edges› proof cases
    assume [simp]: ‹n = m›
    thus ‹?thesis› using assms unfolding is-path-def path-append-def by force
  next
    assume ‹n ≠ m›
    hence ‹Suc n ≤ m› ‹n≤ m› using ∗ by auto
    with assms(1) show ‹?thesis› unfolding is-path-def by auto
  qed
qed

lemma is-path-loop: assumes ‹is-path π› ‹0 < i› ‹π i = π 0› shows ‹is-path (λ n. π (n mod i))› unfolding
is-path-def proof (rule,cases)
  fix n
  assume ‹0 < Suc n mod i›
  hence ‹Suc n mod i = Suc (n mod i)› by (metis mod-Suc neq0-conv)
  moreover
  have ‹(π (n mod i), π (Suc (n mod i))) ∈ edges› using assms(1) unfolding is-path-def by auto
  ultimately
```

show ‹(π (*n mod i*), π (*Suc n mod i*)) $\in$ *edges*› **by** *simp*
next
fix *n*
assume ‹$\neg$ *0* < *Suc n mod i*›
hence ‹*Suc n mod i* = *0*› **by** *auto*
moreover
hence ‹*n mod i* = *i* − *1*› **using** *assms*(*2*) **by** (*metis Zero-neq-Suc diff-Suc-1 mod-Suc*)
ultimately
show ‹(π(*n mod i*), π (*Suc n mod i*)) $\in$ *edges*› **using** *assms*(*1*) **unfolding** *is-path-def* **by** (*metis assms*(*3*) *mod-Suc*)
qed

lemma *path-nodes*: ‹*is-path* $\pi \Longrightarrow \pi$ *k* $\in$ *nodes*› **unfolding** *is-path-def* **using** *edges-nodes* **by** *force*

lemma *direct-path-return'*: **assumes** ‹*is-path* π › ‹π *0* = *x*› ‹*x* $\neq$ *return*› ‹π *n* = *return*›
obtains π' *n'* **where** ‹*is-path* π'› ‹π' *0* = *x*› ‹π' *n'* = *return*› ‹$\forall$ *i*> *0*. π' *i* $\neq$ *x*›
using *assms* **proof** (*induction* ‹*n*› *arbitrary*: ‹π› *rule*: *less-induct*)
case (*less n* π)
hence *ih*: ‹$\bigwedge$ *n'* π'. *n'* < *n* $\Longrightarrow$ *is-path* $\pi' \Longrightarrow \pi'$ *0* = *x* $\Longrightarrow \pi'$ *n'* = *return* $\Longrightarrow$ *thesis*› **using** *assms* **by** *auto*
show ‹*thesis*› **proof** *cases*
assume ‹$\forall$ *i*>*0*. π *i* $\neq$ *x*› **thus** ‹*thesis*› **using** *less* **by** *auto*
next
assume ‹$\neg$ ($\forall$ *i*>*0*. π *i* $\neq$ *x*)›
then obtain *i* **where** ‹*0*<*i*› ‹π *i* = *x*› **by** *auto*
hence ‹(π«*i*) *0* = *x*› **by** *auto*
moreover
have ‹*i* < *n*› **using** *less*(*3*,*5*,*6*) ‹π *i* = *x*› **by** (*metis linorder-neqE-nat term-path-stable less-imp-le*)
hence ‹(π«*i*) (*n*−*i*) = *return*› **using** *less*(*6*) **by** *auto*
moreover
have ‹*is-path* (π«*i*)› **using** *less*(*3*) **by** (*metis path-path-shift*)
moreover
have ‹*n* − *i* < *n*› **using** ‹*0*<*i*› ‹*i* < *n*› **by** *auto*
ultimately show ‹*thesis*› **using** *ih* **by** *auto*
qed
qed

lemma *direct-path-return*: **assumes** ‹*x* $\in$ *nodes*› ‹*x* $\neq$ *return*›
obtains π *n* **where** ‹*is-path* π› ‹π *0* = *x*› ‹π *n* = *return*› ‹$\forall$ *i*> *0*. π *i* $\neq$ *x*›
using *direct-path-return'*[*of* - ‹*x*›] *reaching-ret*[*OF assms*(*1*)] *assms*(*2*) **by** *blast*

lemma *path-append-eq-up-to*: ‹(π @^{k} π') $=_k$ π› **unfolding** *eq-up-to-def* **by** *auto*

lemma *eq-up-to-le*: **assumes** ‹*k* $\leq$ *n*› ‹π $=_n$ π'› **shows** ‹π $=_k$ π'› **using** *assms* **unfolding** *eq-up-to-def* **by** *auto*

lemma *eq-up-to-refl*: **shows** ‹π $=_k$ π› **unfolding** *eq-up-to-def* **by** *auto*

lemma *eq-up-to-sym*: **assumes** ‹π $=_k$ π'› **shows** ‹π' $=_k$ π› **using** *assms* **unfolding** *eq-up-to-def* **by** *auto*

lemma *eq-up-to-apply*: **assumes** ‹π $=_k$ π'› ‹*j* $\leq$ *k*› **shows** ‹π *j* = π' *j*› **using** *assms* **unfolding** *eq-up-to-def* **by** *auto*

lemma *path-swap-ret*: **assumes** ‹*is-path* π› **obtains** π' *n* **where** ‹*is-path* π'› ‹π $=_k$ π'› ‹π' *n* = *return*›
proof −
have *nd*: ‹π *k* $\in$ *nodes*› **using** *assms path-nodes* **by** *simp*
obtain π' *n* **where** $*$: ‹*is-path* π'› ‹π' *0* = π *k*› ‹π' *n* = *return*› **using** *reaching-ret*[*OF nd*] **by** *blast*

have ‹$\pi =_k (\pi @^k \pi')$› **by** (*metis eq-up-to-sym path-append-eq-up-to*)
moreover
have ‹*is-path* $(\pi @^k \pi')$› **using** *assms* $*$ *path-cons* **by** *metis*
moreover
have ‹$(\pi @^k \pi')\ (k + n) =$ *return*› **using** $*$ **by** *auto*
ultimately
show ‹*thesis*› **using** *that* **by** *blast*
qed

lemma *path-suc*: ‹*path* σ (*Suc* k) = *fst* (*step* (*path* σ k, σ^k))› **by** (*induction* ‹k›, *auto simp*: *path-def kth-state-def*)

lemma *kth-state-suc*: ‹$\sigma^{Suc\ k}$ = *snd* (*step* (*path* σ k, σ^k))› **by** (*induction* ‹k›, *auto simp*: *path-def kth-state-def*)

2.3 Facts about Post Dominators

lemma *pd-trans*: **assumes** *1*: ‹y *pd*$\rightarrow$ x› **and** *2*: ‹z *pd*$\rightarrow$$y$› **shows** ‹$z$ *pd*$\rightarrow$$x$›
proof −
{
fix π n
assume *3*[*simp*]: ‹*is-path* π› ‹π *0* $= x$› ‹π n = *return*›
then obtain k **where** ‹$\pi\ k = y$› **and** *7*: ‹$k \leq n$› **using** *1* **unfolding** *is-pd-def* **by** *blast*
then have ‹$(\pi«k)$ *0* $= y$› **and** ‹$(\pi«k)\ (n-k)$ = *return*› **by** *auto*
moreover have ‹*is-path* $(\pi«k)$› **by**(*metis 3*(*1*) *path-path-shift*)
ultimately obtain k' **where** *8*: ‹$(\pi«k)\ k' = z$› **and** ‹$k' \leq n-k$› **using** *2* **unfolding** *is-pd-def* **by** *blast*
hence ‹$k+k' \leq n$› **and** ‹$\pi\ (k+ k') = z$› **using** *7* **by** *auto*
hence ‹$\exists\, k \leq n.\ \pi\ k = z$› **using** *path-nodes* **by** *auto*
}
thus ‹*?thesis*› **using** *1* **unfolding** *is-pd-def* **by** *blast*
qed

lemma *pd-path*: **assumes** ‹y *pd*$\rightarrow$ x›
obtains π n k **where** ‹*is-path* π› **and** ‹π *0* $= x$› **and** ‹π n = *return*› **and** ‹$\pi\ k = y$› **and** ‹$k \leq n$›
using *assms* **unfolding** *is-pd-def* **using** *reaching-ret*[*of* ‹x›] **by** *blast*

lemma *pd-antisym*: **assumes** *xpdy*: ‹x *pd*$\rightarrow$ y› **and** *ypdx*: ‹y *pd*$\rightarrow$ x› **shows** ‹$x = y$›
proof −
obtain π n **where** *path*: ‹*is-path* π› **and** π*0*: ‹π *0* $= x$› **and** πn: ‹π n = *return*› **using** *pd-path*[*OF ypdx*] **by** *metis*
hence *kex*: ‹$\exists\, k \leq n.\ \pi\ k = y$› **using** *ypdx* **unfolding** *is-pd-def* **by** *auto*
obtain k **where** k: ‹k = (*GREATEST* k. $k \leq n \wedge \pi\ k = y$)› **by** *simp*
have πk: ‹$\pi\ k = y$› **and** *kn*: ‹$k \leq n$› **using** k *kex* **by** (*auto intro*: *GreatestIB*)

have *kpath*: ‹*is-path* $(\pi«k)$› **by** (*metis path-path-shift path*)
moreover have *k0*: ‹$(\pi«k)$ *0* $= y$› **using** πk **by** *simp*
moreover have *kreturn*: ‹$(\pi«k)\ (n-k)$ = *return*› **using** *kn* πn **by** *simp*
ultimately have ky': ‹$\exists\, k' \leq (n-k).(\pi«k)\ k' = x$› **using** *xpdy* **unfolding** *is-pd-def* **by** *simp*

obtain k' **where** k': ‹k' = (*GREATEST* k'. $k' \leq (n-k) \wedge (\pi«k)\ k' = x$)› **by** *simp*

with ky' **have** $\pi k'$: ‹$(\pi«k)\ k' = x$› **and** kn': ‹$k' \leq (n-k)$› **by** (*auto intro*: *GreatestIB*)
have k'*path*: ‹*is-path* $(\pi«k«k')$› **using** *kpath* **by**(*metis path-path-shift*)
moreover have k'*0*: ‹$(\pi«k«k')$ *0* $= x$› **using** $\pi k'$ **by** *simp*
moreover have k'*return*: ‹$(\pi«k«k')\ (n-k-k')$ = *return*› **using** kn' *kreturn* **by** (*metis path-shift-def le-add-diff-inverse*)
ultimately have ky'': ‹$\exists\, k'' \leq (n-k-k').(\pi«k«k')\ k'' = y$› **using** *ypdx* **unfolding** *is-pd-def* **by** *blast*

obtain k'' **where** k'': ‹k''= (*GREATEST* k''. $k'' \leq (n-k-k') \wedge (\pi«k«k')\ k'' = y$)› **by** *simp*

with *ky''* **have** *πk''*: ‹(π«k«k') k'' = y› **and** *kn''*: ‹k'' ≤ (n−k−k')› **by** (*auto intro*: *GreatestIB*)

from *this*(*1*) **have** ‹π (k + k' + k'') = y› **by** (*metis path-shift-def add.commute add.left-commute*)
moreover
have ‹k + k' +k'' ≤ n› **using** *kn'' kn' kn* **by** *simp*
ultimately have ‹k + k' + k''≤ k› **using** *k* **by**(*auto simp*: *GreatestB-le*)
hence ‹k' = 0› **by** *simp*
with *k0 πk'* **show** ‹x = y› **by** *simp*
qed

lemma *pd-refl*[*simp*]: ‹x ∈ nodes ⟹ x pd→ x› **unfolding** *is-pd-def* **by** *blast*

lemma *pdt-trans-in-pdt*: ‹(x,y) ∈ pdt⁺ ⟹ (x,y) ∈ pdt›
proof (*induction rule*: *trancl-induct*)
case *base* **thus** ‹?case› **by** *simp*
next
case (*step y z*) **show** ‹?case› **unfolding** *pdt-def* **proof** (*simp*)
have ∗: ‹y pd→ x› ‹z pd→ y› **using** *step* **unfolding** *pdt-def* **by** *auto*
hence [*simp*]: ‹z pd→ x› **using** *pd-trans*[**where** x=‹x› **and** y=‹y› **and** z=‹z›] **by** *simp*
have ‹x≠z› **proof**
assume ‹x = z›
hence ‹z pd→ y› ‹y pd→ z› **using** ∗ **by** *auto*
hence ‹z = y› **using** *pd-antisym* **by** *auto*
thus ‹False› **using** *step*(*2*) **unfolding** *pdt-def* **by** *simp*
qed
thus ‹x ≠ z ∧ z pd→ x› **by** *auto*
qed
qed

lemma *pdt-trancl-pdt*: ‹pdt⁺ = pdt› **using** *pdt-trans-in-pdt* **by** *fast*

lemma *trans-pdt*: ‹trans pdt› **by** (*metis pdt-trancl-pdt trans-trancl*)

definition [*simp*]: ‹pdt-inv = pdt⁻¹›

lemma *wf-pdt-inv*: ‹wf (pdt-inv)› **proof** (*rule ccontr*)
assume ‹¬ wf (pdt-inv)›
then obtain *f* **where** ‹∀ i. (f (Suc i), f i) ∈ pdt⁻¹› **using** *wf-iff-no-infinite-down-chain* **by** *force*
hence ∗: ‹∀ i. (f i, f (Suc i)) ∈ pdt› **by** *simp*
have ∗∗:‹∀ i. ∀ j>i. (f i, f j) ∈ pdt› **proof**(*rule*,*rule*,*rule*)
fix *i j* **assume** ‹i < (j::nat)› **thus** ‹(f i, f j) ∈ pdt› **proof** (*induction* ‹j› *rule*: *less-induct*)
case (*less k*)
show ‹?case› **proof** (*cases* ‹Suc i < k›)
case *True*
hence *k*:‹k−1 < k› ‹i < k−1› **and** *sk*: ‹Suc (k−1) = k› **by** *auto*
show ‹?thesis› **using** *less*(*1*)[*OF k*] ∗[*rule-format*,*of* ‹k−1›,*unfolded sk*] *trans-pdt*[*unfolded trans-def*] **by** *blast*
next
case *False*
hence ‹Suc i = k› **using** *less*(*2*) **by** *auto*
then show ‹?thesis› **using** ∗ **by** *auto*
qed
qed
qed
hence ∗∗∗:‹∀ i. ∀ j > i. f j pd→ f i› ‹∀ i. ∀ j > i. f i ≠ f j› **unfolding** *pdt-def* **by** *auto*
hence ∗∗∗∗:‹∀ i>0. f i pd→ f 0› **by** *simp*

hence ‹$f\ 0 \in nodes$› **using** $*$ *is-pd-def* **by** *fastforce*
then obtain $\pi\ n$ **where** π:‹*is-path* π› ‹$\pi\ 0 = f\ 0$› ‹$\pi\ n = return$› **using** *reaching-ret* **by** *blast*
hence ‹$\forall\ i>0.\ \exists\ k\leq n.\ \pi\ k = f\ i$› **using** $***(1)$ ‹$f\ 0 \in nodes$› **unfolding** *is-pd-def* **by** *blast*
hence πf:‹$\forall\ i.\ \exists\ k\leq n.\ \pi\ k = f\ i$› **using** $\pi(2)$ **by** (*metis le0 not-gr-zero*)
have ‹*range* $f \subseteq \pi$ ‘ $\{..n\}$› **proof**(*rule subsetI*)
fix x **assume** ‹$x \in$ *range* f›
then obtain i **where** ‹$x = f\ i$› **by** *auto*
then obtain k **where** ‹$x = \pi\ k$› ‹$k \leq n$› **using** πf **by** *metis*
thus ‹$x \in \pi$ ‘ $\{..n\}$› **by** *simp*
qed
hence f:‹*finite* (*range* f)› **using** *finite-surj* **by** *auto*
hence *fi*:‹$\exists\ i.$ *infinite* $\{j.\ f\ j = f\ i\}$› **using** *pigeonhole-infinite*[*OF* - f] **by** *auto*
obtain i **where** ‹*infinite* $\{j.\ f\ j = f\ i\}$› **using** *fi* **..**
thus ‹*False*›
by (*metis* (*mono-tags, lifting*) $***(2)$ *bounded-nat-set-is-finite gt-ex mem-Collect-eq nat-neq-iff*)
qed

lemma *return-pd*: **assumes** ‹$x \in nodes$› **shows** ‹*return pd*$\rightarrow$ x› **unfolding** *is-pd-def* **using** *assms* **by** *blast*

lemma *pd-total*: **assumes** *xz*: ‹x *pd*$\rightarrow$ z› **and** *yz*: ‹y *pd*$\rightarrow$ z› **shows** ‹x *pd*$\rightarrow$ $y \lor y$ *pd*$\rightarrow x$›
proof −
obtain $\pi\ n$ **where** *path*: ‹*is-path* π› **and** $\pi 0$: ‹$\pi\ 0 = z$› **and** πn: ‹$\pi\ n = return$› **using** *xz reaching-ret* **unfolding** *is-pd-def* **by** *force*
have $*$: ‹$\exists\ k\leq n.\ (\pi\ k = x \lor \pi\ k = y)$› (**is** ‹$\exists\ k\leq n.\ ?P\ k$›) **using** *path* $\pi 0$ πn *xz yz* **unfolding** *is-pd-def* **by** *auto*
obtain k **where** k: ‹$k = (LEAST\ k.\ \pi\ k = x \lor \pi\ k = y)$› **by** *simp*
hence *kn*: ‹$k\leq n$› **and** πk: ‹$\pi\ k = x \lor \pi\ k = y$› **using** *LeastBI-ex*[*OF* $*$] **by** *auto*
note *k-le* = *Least-le*[**where** P = ‹$?P$›]
show ‹*?thesis*› **proof** (*cases*)
assume *kx*: ‹$\pi\ k = x$›
have *k-min*: ‹$\bigwedge\ k'.\ \pi\ k' = y \Longrightarrow k \leq k'$› **using** *k-le* **unfolding** k **by** *auto*
{
fix π'
and n' :: ‹*nat*›
assume *path'*: ‹*is-path* π'› **and** $\pi' 0$: ‹$\pi'\ 0 = x$› **and** $\pi' n'$: ‹$\pi'\ n' = return$›
have *path''*: ‹*is-path* $(\pi\ @^k\ \pi')$› **using** *path-cons*[*OF path path'*] *kx* $\pi' 0$ **by** *auto*
have $\pi'' 0$: ‹$(\pi\ @^k\ \pi')\ 0 = z$› **using** $\pi 0$ **by** *simp*
have $\pi'' n$: ‹$(\pi\ @^k\ \pi')\ (k+n') = return$› **using** $\pi' n'$ *kx* $\pi' 0$ **by** *auto*
obtain k' **where** k': ‹$k' \leq k + n'$› ‹$(\pi\ @^k\ \pi')\ k' = y$› **using** *yz path''* $\pi'' 0$ $\pi'' n$ **unfolding** *is-pd-def* **by** *blast*
have $**$: ‹$k \leq k'$› **proof** (*rule ccontr*)
assume ‹$\neg\ k \leq k'$›
hence ‹$k' < k$› **by** *simp*
moreover
hence ‹$\pi\ k' = y$› **using** k' **by** *auto*
ultimately
show ‹*False*› **using** *k-min* **by** *force*
qed
hence ‹$\pi'\ (k' - k) = y$› **using** k' $\pi' 0$ *kx* **by** *auto*
moreover
have ‹$(k' - k) \leq n'$› **using** k' **by** *auto*
ultimately
have ‹$\exists\ k\leq n'.\ \pi'\ k = y$› **by** *auto*
}
hence ‹y *pd*$\rightarrow$ x› **using** *kx path-nodes path* **unfolding** *is-pd-def* **by** *auto*

```
    thus ‹?thesis› ..
  next — This is analogous argument
    assume kx: ‹π k ≠ x›
    hence ky: ‹π k = y› using πk by auto
    have k-min: ‹⋀ k'. π k' = x ⟹ k ≤ k'› using k-le unfolding k by auto
     {
       fix π'
       and n' :: ‹nat›
       assume path': ‹is-path π'› and π'0: ‹π' 0 = y› and π'n': ‹π' n' = return›
       have path'': ‹is-path (π @^k π')› using path-cons[OF path path'] ky π'0 by auto
       have π''0: ‹(π @^k π') 0 = z› using π0 by simp
       have π''n: ‹(π @^k π') (k+n') = return› using π'n' ky π'0 by auto
       obtain k' where k': ‹k' ≤ k + n'› ‹(π @^k π') k' = x› using xz path'' π''0 π''n unfolding is-pd-def by
blast
       have **: ‹k ≤ k'› proof (rule ccontr)
         assume ‹¬ k ≤ k'›
         hence ‹k' < k› by simp
         moreover
         hence ‹π k' = x› using k' by auto
         ultimately
         show ‹False› using k-min by force
       qed
       hence ‹π' (k' − k) = x› using k' π'0 ky  by auto
       moreover
       have ‹(k' − k) ≤ n'› using k' by auto
       ultimately
       have ‹∃ k≤ n'. π' k = x› by auto
     }
    hence ‹x pd→ y› using ky path-nodes path unfolding is-pd-def by auto
    thus ‹?thesis› ..
  qed
qed

lemma pds-finite: ‹finite {y . (x,y) ∈ pdt}› proof cases
  assume ‹x ∈ nodes›
  then obtain π n where π:‹is-path π› ‹π 0 = x› ‹π n = return› using reaching-ret by blast
  have *: ‹∀ y ∈ {y. (x,y)∈ pdt}. y pd→ x› using pdt-def by auto
  have ‹∀ y ∈ {y. (x,y)∈ pdt}. ∃ k ≤ n. π k = y›  using * π is-pd-def by blast
  hence ‹{y. (x,y)∈ pdt} ⊆ π ‘ {..n}›  by auto
  then show ‹?thesis› using finite-surj by blast
next
  assume ‹¬ x∈ nodes›
  hence ‹{y. (x,y)∈pdt} = {}› unfolding pdt-def is-pd-def using path-nodes reaching-ret by fastforce
  then show ‹?thesis› by simp
qed

lemma ipd-exists: assumes node: ‹x ∈ nodes› and not-ret: ‹x≠return› shows ‹∃ y. y ipd→ x›
proof −
  let ‹?Q› = ‹{y. x≠y ∧ y pd→ x}›
  have *: ‹return ∈ ?Q› using assms return-pd by simp
  hence **: ‹∃ x. x∈ ?Q› by auto
  have fin: ‹finite ?Q› using pds-finite unfolding pdt-def by auto
  have tot: ‹∀ y z. y∈?Q ∧ z ∈ ?Q ⟶ z pd→ y ∨ y pd→ z› using pd-total by auto
  obtain y where ymax: ‹y∈ ?Q› ‹∀ z∈?Q. z = y ∨ z pd→ y› using fin ** tot proof (induct)
    case empty
```

```
    then show ‹?case› by auto
  next
    case (insert x F) show ‹thesis› proof (cases ‹F = {}›)
      assume ‹F = {}›
      thus ‹thesis› using insert(4)[of ‹x›] by auto
    next
      assume ‹F ≠ {}›
      hence ‹∃ x. x∈ F› by auto
      have ‹⋀y. y ∈ F ⟹ ∀ z∈F. z = y ∨ z pd→ y ⟹ thesis› proof –
        fix y assume a: ‹y ∈ F› ‹∀ z∈F. z = y ∨ z pd→ y›
        have ‹x ≠ y› using insert a by auto
        have ‹x pd→ y ∨ y pd→ x› using insert(6) a(1) by auto
        thus ‹thesis› proof
          assume ‹x pd→ y›
          hence ‹∀ z∈insert x F. z = y ∨ z pd→ y› using a(2) by blast
          thus ‹thesis› using a(1) insert(4) by blast
        next
          assume ‹y pd→ x›
          have ‹∀ z∈insert x F. z = x ∨ z pd→ x› proof
            fix z assume ‹z∈ insert x F› thus ‹z = x ∨ z pd→ x› proof(rule,simp)
              assume ‹z∈F›
              hence ‹z = y ∨ z pd→ y› using a(2) by auto
              thus ‹z = x ∨ z pd→ x› proof(rule,simp add: ‹y pd→ x›)
                assume ‹z pd→ y›
                show ‹z = x ∨ z pd→ x› using ‹y pd→ x› ‹z pd→ y› pd-trans by blast
              qed
            qed
          qed
          then show ‹thesis› using insert by blast
        qed
      qed
      then show ‹thesis› using insert by blast
    qed
  qed
  hence ***: ‹y pd→ x› ‹x≠y› by auto
  have ‹∀ z. z ≠ x ∧ z pd→ x ⟶ z pd→ y› proof (rule,rule)
    fix z
    assume a: ‹ z ≠ x ∧ z pd→ x›
    hence b: ‹z ∈ ?Q› by auto
    have ‹y pd→ z ∨ z pd→ y› using pd-total ***(1) a by auto
    thus ‹z pd→ y› proof
      assume c: ‹y pd→ z›
      hence ‹y = z› using b ymax pdt-def pd-antisym by auto
      thus ‹z pd→ y› using c by simp
    qed simp
  qed
  with *** have ‹y ipd→ x› unfolding is-ipd-def by simp
  thus ‹?thesis› by blast
qed

lemma ipd-unique: assumes yipd: ‹y ipd→ x› and y'ipd: ‹y' ipd→ x› shows ‹y = y'›
proof –
  have 1: ‹y pd→ y'› and 2: ‹y' pd→ y› using yipd y'ipd unfolding is-ipd-def by auto
  show ‹?thesis› using pd-antisym[OF 1 2] .
qed
```

lemma *ipd-is-ipd*: **assumes** ‹$x \in$ *nodes*› **and** ‹$x{\neq}$*return*› **shows** ‹*ipd x ipd*$\rightarrow$ *x*› **proof** −
  from *assms* **obtain** *y* **where** ‹*y ipd*$\rightarrow$ *x*› **using** *ipd-exists* **by** *auto*
  moreover
  hence ‹$\bigwedge$ *z*. *z ipd*$\rightarrow$*x* $\Longrightarrow$ *z* = *y*› **using** *ipd-unique* **by** *simp*
  ultimately show ‹*?thesis*› **unfolding** *ipd-def* **by** (*auto intro*: *theI2*)
qed

lemma *is-ipd-in-pdt*: ‹*y ipd*$\rightarrow$ *x* $\Longrightarrow$ (*x*,*y*) $\in$ *pdt*› **unfolding** *is-ipd-def pdt-def* **by** *auto*

lemma *ipd-in-pdt*: ‹$x \in$ *nodes* $\Longrightarrow$ $x{\neq}$*return* $\Longrightarrow$ (*x*,*ipd x*) $\in$ *pdt*› **by** (*metis ipd-is-ipd is-ipd-in-pdt*)

lemma *no-pd-path*: **assumes** ‹$x \in$ *nodes*› **and** ‹$\neg$ *y pd*$\rightarrow$ *x*›
obtains π *n* **where** ‹*is-path* π› **and** ‹π *0* = *x*› **and** ‹π *n* = *return*› **and** ‹$\forall$ $k \leq n$. π $k \neq y$›
proof (*rule ccontr*)
  assume ‹$\neg$ *thesis*›
  hence ‹$\forall$ π *n*. *is-path* π $\wedge$ π *0* = *x* $\wedge$ π *n* = *return* $\longrightarrow$ ($\exists$ $k{\leq}n$. π *k* = *y*)› **using** *that* **by** *force*
  thus ‹*False*› **using** *assms* **unfolding** *is-pd-def* **by** *auto*
qed

lemma *pd-pd-ipd*: **assumes** ‹$x \in$ *nodes*› ‹$x{\neq}$*return*› ‹$y{\neq}x$› ‹*y pd*$\rightarrow$ *x*› **shows** ‹*y pd*$\rightarrow$ *ipd x*›
proof −
  have ‹*ipd x pd*$\rightarrow$ *x*› **by** (*metis assms*(*1*,*2*) *ipd-is-ipd is-ipd-def*)
  hence ‹*y pd*$\rightarrow$ *ipd x* $\vee$ *ipd x pd*$\rightarrow$ *y*› **by** (*metis assms*(*4*) *pd-total*)
  thus ‹*?thesis*› **proof**
    have *1*: ‹*ipd x ipd*$\rightarrow$ *x*› **by** (*metis assms*(*1*,*2*) *ipd-is-ipd*)
    moreover
    assume ‹*ipd x pd*$\rightarrow$ *y*›
    ultimately
    show ‹*y pd*$\rightarrow$ *ipd x*› **unfolding** *is-ipd-def* **using** *assms*(*3*,*4*) **by** *auto*
  qed *auto*
qed

lemma *pd-nodes*: **assumes** ‹*y pd*$\rightarrow$ *x*› **shows** *pd-node1*: ‹$y \in$ *nodes*› **and** *pd-node2*: ‹$x \in$ *nodes*›
proof −
  obtain π *k* **where** ‹*is-path* π› ‹π *k* = *y*› **using** *assms* **unfolding** *is-pd-def* **using** *reaching-ret* **by** *force*
  thus ‹$y \in$ *nodes*› **using** *path-nodes* **by** *auto*
  show ‹$x \in$ *nodes*› **using** *assms* **unfolding** *is-pd-def* **by** *simp*
qed

lemma *pd-ret-is-ret*: ‹*x pd*$\rightarrow$ *return* $\Longrightarrow$ *x* = *return*› **by** (*metis pd-antisym pd-node1 return-pd*)

lemma *ret-path-none-pd*: **assumes** ‹$x \in$ *nodes*› ‹$x{\neq}$*return*›
obtains π *n* **where** ‹*is-path* π› ‹π *0* = *x*› ‹π *n* = *return*› ‹$\forall$ $i{>}0$. $\neg$ *x pd*$\rightarrow$ π *i*›
proof(*rule ccontr*)
  assume ‹$\neg$*thesis*›
  hence $*$: ‹$\bigwedge$ π *n*. ⟦*is-path* π; π *0* = *x*; π *n* = *return*⟧ $\Longrightarrow$ $\exists$ $i{>}0$. *x pd*$\rightarrow$ π *i*› **using** *that* **by** *blast*
  obtain π *n* **where** $**$: ‹*is-path* π› ‹π *0* = *x*› ‹π *n* = *return*› ‹$\forall$ $i{>}0$. π $i \neq x$› **using** *direct-path-return*[*OF assms*] **by** *metis*
  then obtain *i* **where** $***$: ‹$i{>}0$› ‹*x pd*$\rightarrow$ π *i*› **using** $*$ **by** *blast*
  hence ‹π $i \neq$ *return*› **using** *pd-ret-is-ret assms*(*2*) **by** *auto*
  hence ‹$i < n$› **using** *assms*(*2*) *term-path-stable* $**$ **by** (*metis linorder-neqE-nat less-imp-le*)
  hence ‹(π«*i*)($n{-}i$) = *return*› **using** $**$(*3*) **by** *auto*
  moreover
  have ‹(π«*i*) (*0*) = π *i*› **by** *simp*
  moreover
  have ‹*is-path* (π«*i*)› **using** $**$(*1*) *path-path-shift* **by** *metis*

ultimately
obtain *k* **where** ‹$(\pi \ll i)\ k = x$› **using** ∗∗∗(*2*) **unfolding** *is-pd-def* **by** *metis*
hence ‹$\pi\ (i + k) = x$› **by** *auto*
thus ‹*False*› **using** ∗∗(*4*) ‹$i>0$› **by** *auto*
qed

lemma *path-pd-ipd0′*: **assumes** ‹*is-path* π› **and** ‹$\pi\ n \neq$ *return*› ‹$\pi\ n \neq \pi\ 0$› **and** ‹$\pi\ n$ *pd*$\rightarrow$ $\pi\ 0$›
obtains *k* **where** ‹$k \leq n$› **and** ‹$\pi\ k =$ *ipd*$(\pi\ 0)$›
proof(*rule ccontr*)
have ∗: ‹$\pi\ n$ *pd*$\rightarrow$ *ipd* $(\pi\ 0)$› **by** (*metis is-pd-def assms*(*3*,*4*) *pd-pd-ipd pd-ret-is-ret*)
obtain $\pi'\ n'$ **where** ∗∗: ‹*is-path* π'› ‹$\pi'\ 0 = \pi\ n$› ‹$\pi'\ n' =$ *return*› ‹$\forall\ i>0.\ \neg\ \pi\ n$ *pd*$\rightarrow$ $\pi'\ i$› **by** (*metis assms*(*2*) *assms*(*4*) *pd-node1 ret-path-none-pd*)
hence ‹$\forall\ i>0.\ \pi'\ i \neq$ *ipd* $(\pi\ 0)$› **using** ∗ **by** *metis*
moreover
assume ‹$\neg$ *thesis*›
hence ‹$\forall\ k\leq n.\ \pi\ k \neq$ *ipd* $(\pi\ 0)$› **using** *that* **by** *blast*
ultimately
have ‹$\forall\ i.\ (\pi @^n\ \pi')\ i \neq$ *ipd* $(\pi\ 0)$› **by** (*metis diff-is-0-eq neq0-conv path-append-def*)
moreover
have ‹$(\pi @^n\ \pi')\ (n + n') =$ *return*›
by (*metis* ‹$\pi'\ 0 = \pi\ n$› ‹$\pi'\ n' =$ *return*› *add-diff-cancel-left′ assms*(*2*) *diff-is-0-eq path-append-def*)
moreover
have ‹$(\pi @^n\ \pi')\ 0 = \pi\ 0$› **by** (*metis le0 path-append-def*)
moreover
have ‹*is-path* $(\pi @^n\ \pi')$› **by** (*metis* ‹$\pi'\ 0 = \pi\ n$› ‹*is-path* π'› *assms*(*1*) *path-cons*)
moreover
have ‹*ipd* $(\pi\ 0)$ *pd*$\rightarrow$ $\pi\ 0$› **by** (*metis* ∗∗(*2*,*3*,*4*) *assms*(*2*) *assms*(*4*) *ipd-is-ipd is-ipd-def neq0-conv pd-node2*)
moreover
have ‹$\pi\ 0 \in$ *nodes*› **by** (*metis assms*(*1*) *path-nodes*)
ultimately
show ‹*False*› **unfolding** *is-pd-def* **by** *blast*
qed

lemma *path-pd-ipd0*: **assumes** ‹*is-path* π› **and** ‹$\pi\ 0 \neq$ *return*› ‹$\pi\ n \neq \pi\ 0$› **and** ‹$\pi\ n$ *pd*$\rightarrow$ $\pi\ 0$›
obtains *k* **where** ‹$k \leq n$› **and** ‹$\pi\ k =$ *ipd*$(\pi\ 0)$›
proof *cases*
assume ∗: ‹$\pi\ n =$ *return*›
have ‹*ipd* $(\pi\ 0)$ *pd*$\rightarrow$ $(\pi\ 0)$› **by** (*metis is-ipd-def is-pd-def assms*(*2*,*4*) *ipd-is-ipd*)
with *assms*(*1*,*2*,*3*) ∗ **show** ‹*thesis*› **unfolding** *is-pd-def* **by** (*metis that*)
next
assume ‹$\pi\ n \neq$ *return*›
from *path-pd-ipd0′* [*OF assms*(*1*) *this assms*(*3*,*4*)] *that* **show** ‹*thesis*› **by** *auto*
qed

lemma *path-pd-ipd*: **assumes** ‹*is-path* π› **and** ‹$\pi\ k \neq$ *return*› ‹$\pi\ n \neq \pi\ k$› **and** ‹$\pi\ n$ *pd*$\rightarrow$ $\pi\ k$› **and** *kn*: ‹$k < n$›
obtains *l* **where** ‹$k < l$› **and** ‹$l \leq n$› **and** ‹$\pi\ l =$ *ipd*$(\pi\ k)$›
proof −
have ‹*is-path* $(\pi \ll k)$› ‹$(\pi \ll k)\ 0 \neq$ *return*› ‹$(\pi \ll k)\ (n - k) \neq (\pi \ll k)\ 0$› ‹$(\pi \ll k)\ (n - k)$ *pd*$\rightarrow$ $(\pi \ll k)\ 0$›
using *assms path-path-shift* **by** *auto*
with *path-pd-ipd0*[*of* ‹$\pi \ll k$› ‹$n-k$›]
obtain *ka* **where** ‹$ka \leq n - k$› ‹$(\pi \ll k)\ ka =$ *ipd* $((\pi \ll k)\ 0)$› **.**
hence ‹$k + ka \leq n$› ‹$\pi\ (k + ka) =$ *ipd* $(\pi\ k)$› **using** *kn* **by** *auto*
moreover
hence ‹$\pi\ (k + ka)$ *ipd*$\rightarrow$ $\pi\ k$› **by** (*metis assms*(*1*) *assms*(*2*) *ipd-is-ipd path-nodes*)
hence ‹$k < k + ka$› **unfolding** *is-ipd-def* **by** (*metis nat-neq-iff not-add-less1*)

ultimately
show ‹*thesis*› **using** *that*[*of* ‹*k*+*ka*›] **by** *auto*
qed

lemma *path-ret-ipd*: **assumes** ‹*is-path* π› **and** ‹π $k \neq$ *return*› ‹π $n =$ *return*›
obtains l **where** ‹$k < l$› **and** ‹$l \leq n$› **and** ‹π $l =$ *ipd*(π k)›
proof −
have ‹π $n \neq \pi$ k› **using** *assms* **by** *auto*
moreover
have ‹$k \leq n$› **apply** (*rule ccontr*) **using** *term-path-stable assms* **by** *auto*
hence ‹$k < n$› **by** (*metis assms*(*2*,*3*) *dual-order.order-iff-strict*)
moreover
have ‹π n *pd*→ π k› **by** (*metis assms*(*1*,*3*) *path-nodes return-pd*)
ultimately
obtain l **where** ‹$k < l$› ‹$l \leq n$› ‹π $l =$ *ipd* (π k)› **using** *assms path-pd-ipd* **by** *blast*
thus ‹*thesis*› **using** *that* **by** *auto*
qed

lemma *pd-intro*: **assumes** ‹l *pd*→ k› ‹*is-path* π› ‹π *0* $= k$› ‹π $n =$ *return*›
obtains i **where** ‹$i \leq n$› ‹π $i = l$› **using** *assms* **unfolding** *is-pd-def* **by** *metis*

lemma *path-pd-pd0*: **assumes** *path*: ‹*is-path* π› **and** *lpdn*: ‹π l *pd*→ n› **and** *npd0*: ‹n *pd*→ π *0*›
obtains k **where** ‹$k \leq l$› ‹π $k = n$›
proof (*rule ccontr*)
assume ‹$\neg$ *thesis*›
hence *notn*: ‹$\bigwedge$ k. $k \leq l \Longrightarrow \pi$ $k \neq n$› **using** *that* **by** *blast*
have *nret*: ‹π $l \neq$ *return*› **by** (*metis is-pd-def assms*(*1*,*3*) *notn*)

obtain π' n' **where** *path'*: ‹*is-path* π'› **and** π*0'*: ‹π' *0* $= \pi$ l› **and** $\pi n'$: ‹π' $n' =$ *return*› **and** *nonepd*: ‹$\forall$ i>*0*. $\neg$ π l *pd*→ π' i›
using *nret path path-nodes ret-path-none-pd* **by** *metis*

have ‹π $l \neq n$› **using** *notn* **by** *simp*
hence ‹$\forall$ i. π' $i \neq n$› **using** *nonepd* π*0'* *lpdn* **by** (*metis neq0-conv*)

hence *notn'*: ‹$\forall$ i. (π@^{l} π') $i \neq n$› **using** *notn* π*0'* **by** *auto*

have ‹*is-path* (π@^{l} π')› **using** *path path'* **by** (*metis* π*0'* *path-cons*)
moreover
have ‹(π@^{l} π') *0* $= \pi$ *0*› **by** *simp*
moreover
have ‹(π@^{l} π') ($n' + l$) $=$ *return*› **using** π*0'* $\pi n'$ **by** *auto*
ultimately
show ‹*False*› **using** *notn' npd0* **unfolding** *is-pd-def* **by** *blast*
qed

2.4 Facts about Control Dependencies

lemma *icd-imp-cd*: ‹n *icd*$^\pi$→ $k \Longrightarrow n$ *cd*$^\pi$→ k› **by** (*metis is-icdi-def*)

lemma *ipd-impl-not-cd*: **assumes** ‹$j \in \{k..i\}$› **and** ‹π $j =$ *ipd* (π k)› **shows** ‹$\neg$ i *cd*$^\pi$→ k›
by (*metis assms*(*1*) *assms*(*2*) *is-cdi-def*)

lemma *cd-not-ret*: **assumes** ‹i *cd*$^\pi$→ k › **shows** ‹π $k \neq$ *return*› **by** (*metis is-cdi-def assms nat-less-le term-path-stable*)

lemma *cd-path-shift*: **assumes** ‹$j \leq k$› ‹*is-path* π › **shows** ‹(i *cd*$^\pi$→ k) = ($i - j$ *cd*$^{\pi«j}$→ $k-j$)› **proof**

assume *a*: ‹$i\ cd^{\pi}\rightarrow\ k$›
hence *b*: ‹$k < i$› **by** (*metis is-cdi-def*)
hence ‹*is-path* (π « j)› ‹$k - j < i - j$› **using** *assms* **apply** (*metis path-path-shift*)
by (*metis assms*(*1*) *b diff-less-mono*)
moreover
have *c*: ‹$\forall\ j \in \{k..i\}.\ \pi\ j \neq ipd\ (\pi\ k)$› **by** (*metis a ipd-impl-not-cd*)
hence ‹$\forall\ ja \in \{k - j..i - j\}.\ (\pi$ « $j)\ ja \neq ipd\ ((\pi$ « $j)\ (k - j))$› **using** *b assms* **by** *auto fastforce*
moreover
have ‹$j < i$› **using** *assms*(*1*) *b* **by** *auto*
hence ‹$(\pi$«$j)\ (i - j) \neq return$› **using** *a* **unfolding** *is-cdi-def* **by** *auto*
ultimately
show ‹$i - j\ cd^{\pi«j}\rightarrow\ k-j$› **unfolding** *is-cdi-def* **by** *simp*
next
assume *a*: ‹$i - j\ cd^{\pi«j}\rightarrow\ k-j$›
hence *b*: ‹$k - j < i-j$› **by** (*metis is-cdi-def*)
moreover
have *c*: ‹$\forall\ ja \in \{k - j..i - j\}.\ (\pi$ « $j)\ ja \neq ipd\ ((\pi$ « $j)\ (k - j))$› **by** (*metis a ipd-impl-not-cd*)
have ‹$\forall\ j \in \{k..i\}.\ \pi\ j \neq ipd\ (\pi\ k)$› **proof** (*rule,goal-cases*) **case** (*1 n*)
hence ‹$n-j \in \{k-j..i-j\}$› **using** *assms* **by** *auto*
hence ‹$\pi\ (j + (n-j)) \neq ipd(\pi\ (j + (k-j)))$› **by** (*metis c path-shift-def*)
thus ‹*?case*› **using** *1 assms*(*1*) **by** *auto*
qed
moreover
have ‹$j < i$› **using** *assms*(*1*) *b* **by** *auto*
hence ‹$\pi\ i \neq return$› **using** *a* **unfolding** *is-cdi-def* **by** *auto*
ultimately
show ‹$i\ cd^{\pi}\rightarrow k$› **unfolding** *is-cdi-def* **by** (*metis assms*(*1*) *assms*(*2*) *diff-is-0-eq′ le-diff-iff nat-le-linear nat-less-le*)
qed

lemma *cd-path-shift0*: **assumes** ‹*is-path* π› **shows** ‹$(i\ cd^{\pi}\rightarrow\ k) = (i-k\ cd^{\pi«k}\rightarrow 0)$›
using *cd-path-shift*[*OF - assms*] **by** (*metis diff-self-eq-0 le-refl*)

lemma *icd-path-shift*: **assumes** ‹$l \leq k$› ‹*is-path* π› **shows** ‹$(i\ icd^{\pi}\rightarrow\ k) = (i - l\ icd^{\pi«l}\rightarrow\ k - l)$›
proof −
have ‹*is-path* (π«l)› **using** *path-path-shift assms*(*2*) **by** *auto*
moreover
have ‹$(i\ cd^{\pi}\rightarrow\ k) = (i - l\ cd^{\pi«l}\rightarrow\ k - l)$› **using** *assms cd-path-shift* **by** *auto*
moreover
have ‹$(\forall\ m \in \{k<..<i\}.\ \neg\ i\ cd^{\pi}\rightarrow\ m) = (\forall\ m \in \{k - l<..<i - l\}.\ \neg\ i - l\ cd^{\pi\ «\ l}\rightarrow\ m)$›
proof −
{**fix** *m* **assume** ∗: ‹$\forall\ m \in \{k - l<..<i - l\}.\ \neg\ i - l\ cd^{\pi\ «\ l}\rightarrow\ m$› ‹$m \in \{k<..<i\}$›
hence ‹$m-l \in \{k-l<..<i-l\}$› **using** *assms*(*1*) **by** *auto*
hence ‹$\neg\ i - l\ cd^{\pi«l}\rightarrow(m-l)$› **using** ∗ **by** *blast*
moreover
have ‹$l \leq m$› **using** ∗ *assms* **by** *auto*
ultimately have ‹$\neg\ i\ cd^{\pi}\rightarrow m$› **using** *assms*(*2*) *cd-path-shift* **by** *blast*
}
moreover
{**fix** *m* **assume** ∗: ‹$\forall\ m \in \{k<..<i\}.\ \neg\ i\ cd^{\pi}\rightarrow\ m$› ‹$m-l \in \{k-l<..<i-l\}$›
hence ‹$m \in \{k<..<i\}$› **using** *assms*(*1*) **by** *auto*
hence ‹$\neg\ i\ cd^{\pi}\rightarrow m$› **using** ∗ **by** *blast*
moreover
have ‹$l \leq m$› **using** ∗ *assms* **by** *auto*
ultimately have ‹$\neg\ i - l\ cd^{\pi«l}\rightarrow(m-l)$› **using** *assms*(*2*) *cd-path-shift* **by** *blast*
}

ultimately show ‹*?thesis*› **by** *auto* (*metis diff-add-inverse*)
qed
ultimately
show ‹*?thesis*› **unfolding** *is-icdi-def* **using** *assms* **by** *blast*
qed

lemma *icd-path-shift0*: **assumes** ‹*is-path* π› **shows** ‹(i $icd^{\pi}\rightarrow$ k) = ($i-k$ $icd^{\pi«k}\rightarrow 0$)›
using *icd-path-shift*[*OF - assms*] **by** (*metis diff-self-eq-0 le-refl*)

lemma *cdi-path-swap*: **assumes** ‹*is-path* π'› ‹j $cd^{\pi}\rightarrow k$› ‹$\pi =_j \pi'$› **shows** ‹j $cd^{\pi'}\rightarrow k$› **using** *assms* **unfolding** *eq-up-to-def is-cdi-def* **by** *auto*

lemma *cdi-path-swap-le*: **assumes** ‹*is-path* π'› ‹j $cd^{\pi}\rightarrow k$› ‹$\pi =_n \pi'$› ‹$j \leq n$› **shows** ‹j $cd^{\pi'}\rightarrow k$› **by** (*metis assms cdi-path-swap eq-up-to-le*)

lemma *not-cd-impl-ipd*: **assumes** ‹*is-path* π› **and** ‹$k < i$› **and** ‹¬ i $cd^{\pi}\rightarrow$ k› **and** ‹π i ≠ *return*› **obtains** j **where** ‹$j \in \{k..i\}$› **and** ‹π j = *ipd* (π k)›
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *is-cdi-def*)

lemma *icd-is-the-icd*: **assumes** ‹i $icd^{\pi}\rightarrow$ k› **shows** ‹k = (*THE* k. i $icd^{\pi}\rightarrow$ k)› **using** *assms icd-uniq*
by (*metis the1-equality*)

lemma *all-ipd-imp-ret*: **assumes** ‹*is-path* π› **and** ‹∀ i. π i ≠ *return* ⟶ (∃ j>i. π j = *ipd* (π i))› **shows** ‹∃j. π j = *return*›
proof −
{ **fix** x **assume** ∗: ‹π *0* = x›
have ‹*?thesis*› **using** *wf-pdt-inv* ∗ *assms*
proof(*induction* ‹x› *arbitrary*: ‹π› *rule*: *wf-induct-rule*)
case (*less* x π) **show** ‹*?case*› **proof** (*cases* ‹x = *return*›)
case *True* **thus** ‹*?thesis*› **using** *less*(*2*) **by** *auto*
next
assume *not-ret*: ‹x ≠ *return*›
moreover
then obtain k **where** *k-ipd*: ‹π k = *ipd* x› **using** *less*(*2*,*4*) **by** *auto*
moreover
have ‹x ∈ *nodes*› **using** *less*(*2*,*3*) **by** (*metis path-nodes*)
ultimately
have ‹(x, π k) ∈ *pdt*› **by** (*metis ipd-in-pdt*)
hence *a*: ‹(π k, x) ∈ *pdt-inv*› **unfolding** *pdt-inv-def* **by** *simp*
have *b*: ‹*is-path* (π « k)› **by** (*metis less.prems*(*2*) *path-path-shift*)
have *c*: ‹∀ i. (π«k) i ≠ *return* ⟶ (∃j>i. (π«k) j = *ipd* ((π«k) i))› **using** *less*(*4*) **apply** *auto*
by (*metis* (*full-types*) *ab-semigroup-add-class.add-ac*(*1*) *less-add-same-cancel1 less-imp-add-positive*)
from *less*(*1*)[*OF a - b c*]
have ‹∃j. (π«k) j = *return*› **by** *auto*
thus ‹∃j. π j = *return*› **by** *auto*
qed
qed
}
thus ‹*?thesis*› **by** *simp*
qed

lemma *loop-has-cd*: **assumes** ‹*is-path* π› ‹*0* < i› ‹π i = π *0*› ‹π *0* ≠ *return*› **shows** ‹∃ k < i. i $cd^{\pi}\rightarrow$ k›
proof (*rule ccontr*)
let ‹*?π*› = ‹(λ n. π (n *mod* i))›
assume ‹¬ (∃k<i. i $cd^{\pi}\rightarrow$ k)›
hence ‹∀ k <i. ¬ i $cd^{\pi}\rightarrow$ k› **by** *blast*

hence ∗: ‹∀ k<i. (∃j ∈ {k..i}. π j = ipd (π k))› **using** assms(1,3,4) not-cd-impl-ipd **by** metis
have ‹∀ k. (∃ j > k. ?π j = ipd (?π k))› **proof**
fix k
have ‹k mod i < i› **using** assms(2) **by** auto
with ∗ **obtain** j **where** ‹j ∈ {(k mod i)..i}› ‹π j = ipd (π (k mod i))› **by** auto
then obtain j′ **where** 1: ‹j′ < i› ‹π j′ = ipd (π (k mod i))›
by (cases ‹j = i›, auto ,metis assms(2) assms(3),metis le-neq-implies-less)
then obtain j′′ **where** 2: ‹j′′ > k› ‹j′′ mod i = j′› **by** (metis mod-bound-instance)
hence ‹?π j′′ = ipd (?π k)› **using** 1 **by** auto
with 2(1)
show ‹∃ j > k. ?π j = ipd (?π k)› **by** auto
qed
moreover
have ‹is-path ?π› **by** (metis assms(1) assms(2) assms(3) is-path-loop)
ultimately
obtain k **where** ‹?π k = return› **by** (metis (lifting) all-ipd-imp-ret)
moreover
have ‹k mod i < i› **by** (simp add: assms(2))
ultimately
have ‹π i = return› **by** (metis assms(1) term-path-stable less-imp-le)
thus ‹False› **by** (metis assms(3) assms(4))
qed

lemma loop-has-cd′: **assumes** ‹is-path π› ‹j < i› ‹π i = π j› ‹π j ≠ return› **shows** ‹∃ k ∈ {j..<i}. i cd$^{\pi}$→ k›
proof −
have ‹∃ k′< i−j. i−j cd$^{\pi«j}$→k′›
apply(rule loop-has-cd)
apply (metis assms(1) path-path-shift)
apply (auto simp add: assms less-imp-le)
done
then obtain k **where** k: ‹k<i−j› ‹i−j cd$^{\pi«j}$→k› **by** auto
hence k′: ‹(k+j) < i› ‹i−j cd$^{\pi«j}$→ (k+j)−j› **by** auto
note cd-path-shift[OF - assms(1)]
hence ‹i cd$^{\pi}$→ k+j› **using** k′(2) **by** (metis le-add1 add.commute)
with k′(1) **show** ‹?thesis› **by** force
qed

lemma claim′′: **assumes** pathπ: ‹is-path π› **and** pathπ′: ‹is-path π′›
and πi: ‹π i = π′ i′› **and** πj: ‹π j = π′ j′›
and not-cd: ‹∀ k. ¬ j cd$^{\pi}$→ k› ‹∀ k. ¬ i′ cd$^{\pi'}$→ k›
and nret: ‹π i ≠ return›
and ilj: ‹i < j›
shows ‹i′ < j′› **proof** (rule ccontr)
assume ‹¬ i′ < j′›
hence jlei: ‹j′ ≤ i′› **by** auto
show ‹False› **proof** (cases)
assume j′li′: ‹j′ < i′›
define π′′ **where** ‹π′′ ≡ (π@^{j}(π′«j′))«i›
note π′′-def[simp]
have ‹π j = (π′ « j′) 0› **by** (metis path-shift-def Nat.add-0-right πj)
hence ‹is-path π′′› **using** pathπ pathπ′ π′′-def path-path-shift path-cons **by** presburger
moreover
have ‹π′′ (j−i+(i′−j′)) = π′′ 0› **using** ilj jlei πi πj
by (auto, metis add-diff-cancel-left′ le-antisym le-diff-conv le-eq-less-or-eq)
moreover

have ‹π'' *0* $\neq$ *return*› **by** (*simp add*: *ilj less-or-eq-imp-le nret*)
moreover
have ‹$0 < j{-}i{+}(i'{-}j')$› **by** (*metis add-is-0 ilj neq0-conv zero-less-diff*)
ultimately obtain k **where** k: ‹$k < j{-}i{+}(i'{-}j')$› ‹$j{-}i{+}(i'{-}j')$ $cd^{\pi''}\!\rightarrow k$› **by** (*metis loop-has-cd*)
hence $*$: ‹$\forall\ l \in \{k..j{-}i{+}(i'{-}j')\}.\ \pi''\ l \neq ipd\ (\pi''\ k)$› **by** (*metis is-cdi-def*)
show ‹*False*› **proof** (*cases* ‹$k < j{-}i$›)
  assume a: ‹$k < j - i$›
  hence b: ‹$\pi''\ k = \pi\ (i + k)$› **by** *auto*
  have ‹$\forall\ l \in \{i{+}k..j\}.\ \pi\ l \neq ipd\ (\pi\ (i{+}k))$› **proof**
    fix l **assume** l: ‹$l \in \{i + k..j\}$›
    hence ‹$\pi\ l = \pi''\ (l - i)$› **by** *auto*
    moreover
    from a l **have** ‹$l{-}i \in \{k\ ..\ j{-}i + (i'{-}j')\}$› **by** *force*
    ultimately show ‹$\pi\ l \neq ipd\ (\pi\ (i + k))$› **using** $*$ b **by** *auto*
  qed
  moreover
  have ‹$i + k < j$› **using** a **by** *simp*
  moreover
  have ‹$\pi\ j \neq$ *return*› **by** (*metis* πi πj $j'li'$ *nret path*π' *term-path-stable less-imp-le*)
  ultimately
  have ‹j $cd^{\pi}\!\rightarrow i{+}k$› **by** (*metis not-cd-impl-ipd path*π)
  thus ‹*False*› **by** (*metis not-cd*(*1*))
next
  assume ‹$\neg\ k < j - i$›
  hence a: ‹$j - i \leq k$› **by** *simp*
  hence b: ‹$\pi''\ k = \pi'\ (j' + (i + k) - j)$› **unfolding** π''*-def path-shift-def path-append-def* **using** *ilj*
    by(*auto*,*metis* πj *add-diff-cancel-left*$'$ *le-antisym le-diff-conv add.commute*)
  have ‹$\forall\ l \in \{j' + (i{+}k) - j..i'\}.\ \pi'\ l \neq ipd\ (\pi'\ (j' + (i{+}k) - j))$› **proof**
    fix l **assume** l: ‹$l \in \{j' + (i{+}k) - j..i'\}$›
    hence ‹$\pi'\ l = \pi''\ (j + l - i - j')$› **unfolding** π''*-def path-shift-def path-append-def* **using** *ilj*
     by (*auto*, *metis Nat.diff-add-assoc* πj *a add.commute add-diff-cancel-left*$'$ *add-leD1 le-antisym le-diff-conv*)
    moreover
    from a l **have** ‹$j + l - i - j' \in \{k\ ..\ j{-}i + (i'{-}j')\}$› **by** *force*
    ultimately show ‹$\pi'\ l \neq ipd\ (\pi'\ (j' + (i + k) - j))$› **using** $*$ b **by** *auto*
  qed
  moreover
  have ‹$j' + (i{+}k) - j < i'$› **using** a $j'li'$ *ilj* k(*1*) **by** *linarith*
  moreover
  have ‹$\pi'\ i' \neq$ *return*› **by** (*metis* πi *nret*)
  ultimately
  have ‹i' $cd^{\pi'}\!\rightarrow j' + (i{+}k) - j$› **by** (*metis not-cd-impl-ipd path*π')
  thus ‹*False*› **by** (*metis not-cd*(*2*))
qed
next
assume ‹$\neg\ j' < i'$›
hence ‹$j' = i'$› **by** (*metis* ‹$\neg\ i' < j'$› *linorder-cases*)
hence ‹$\pi\ i = \pi\ j$› **by** (*metis* πi πj)
thus ‹*False*› **by** (*metis ilj loop-has-cd*$'$ *not-cd*(*1*) *nret path*π)
qed
qed

lemma *other-claim*$'$: **assumes** *path*: ‹*is-path* π› **and** *eq*: ‹$\pi\ i = \pi\ j$› **and** ‹$\pi\ i \neq$ *return*›
and *icd*: ‹$\forall\ k.\ \neg\ i$ $cd^{\pi}\!\rightarrow k$› **and** ‹$\forall\ k.\ \neg\ j$ $cd^{\pi}\!\rightarrow k$› **shows** ‹$i = j$›
proof (*rule ccontr*,*cases*)
  assume ‹$i < j$› **thus** ‹*False*› **using** *assms claim*$''$ **by** *blast*

next
  assume ‹¬ $i < j$› ‹$i \neq j$›
  hence ‹$j < i$› **by** *auto*
  thus ‹*False*› **using** *assms claim''* **by** (*metis loop-has-cd'*)
qed

lemma *icd-no-cd-path-shift*: **assumes** ‹i *icd*$^{\pi}\rightarrow$ *0*› **shows** ‹(∀ k. ¬ $i - 1$ *cd*$^{\pi «1}\rightarrow k$)›
proof (*rule*,*rule ccontr*,*goal-cases*)
  case (*1 k*)
  hence ∗: ‹$i - 1$ *cd*$^{\pi\ «\ 1}\rightarrow k$› **by** *simp*
  have ∗∗: ‹$1 \leq k + 1$› **by** *simp*
  have ∗∗∗: ‹*is-path* π› **by** (*metis assms is-icdi-def*)
  hence ‹i *cd*$^{\pi}\rightarrow k+1$› **using** *cd-path-shift*[*OF* ∗∗ ∗∗∗] ∗ **by** *auto*
  moreover
  hence ‹$k+1 < i$› **unfolding** *is-cdi-def* **by** *simp*
  moreover
  have ‹$0 < k + 1$› **by** *simp*
  ultimately show ‹*False*› **using** *assms*[*unfolded is-icdi-def*] **by** *auto*
qed

lemma *claim'*: **assumes** *pathπ*: ‹*is-path* π› **and** *pathπ'*: ‹*is-path* π'› **and**
  πi: ‹$\pi\ i = \pi'\ i'$› **and** *πj*: ‹$\pi\ j = \pi'\ j'$› **and** *not-cd*:
  ‹i *icd*$^{\pi}\rightarrow$ *0*› ‹j *icd*$^{\pi}\rightarrow$ *0*›
  ‹i' *icd*$^{\pi'}\rightarrow$ *0*› ‹j' *icd*$^{\pi'}\rightarrow$ *0*›
   and *ilj*: ‹$i < j$›
   and *nret*: ‹$\pi\ i \neq$ *return*›
  shows ‹$i' < j'$›
proof −
  have *g0*: ‹$0 < i$› ‹$0 < j$› ‹$0 < i'$› ‹$0 < j'$›**using** *not-cd*[*unfolded is-icdi-def is-cdi-def*] **by** *auto*
  have ‹$(\pi\ «\ 1)\ (i - 1) = (\pi'\ «\ 1)\ (i' - 1)$› ‹$(\pi\ «\ 1)\ (j - 1) = (\pi'\ «\ 1)\ (j' - 1)$› **using** *πi πj g0* **by** *auto*
  moreover
  have ‹∀ k. ¬ $(j - 1)$ *cd*$^{\pi «1}\rightarrow k$› ‹∀ k. ¬ $(i' - 1)$ *cd*$^{\pi' «1}\rightarrow k$›
    by (*metis icd-no-cd-path-shift not-cd*(*2*)) (*metis icd-no-cd-path-shift not-cd*(*3*))
  moreover
  have ‹*is-path* (π«*1*)› ‹*is-path* (π'«*1*)› **using** *pathπ pathπ' path-path-shift* **by** *blast+*
  moreover
  have ‹(π«*1*) $(i - 1) \neq$ *return*› **using** *g0 nret* **by** *auto*
  moreover
  have ‹$i - 1 < j - 1$› **using** *g0 ilj* **by** *auto*
  ultimately have ‹$i' - 1 < j' - 1$› **using** *claim''* **by** *blast*
  thus ‹$i'<j'$› **by** *auto*
qed

lemma *other-claim*: **assumes** *path*: ‹*is-path* π› **and** *eq*: ‹$\pi\ i = \pi\ j$› **and** ‹$\pi\ i \neq$ *return*›
and *icd*: ‹i *icd*$^{\pi}\rightarrow$ *0*› **and** ‹j *icd*$^{\pi}\rightarrow$ *0*› **shows** ‹$i = j$› **proof** (*rule ccontr*,*cases*)
  assume ‹$i < j$› **thus** ‹*False*› **using** *assms claim'* **by** *blast*
next
  assume ‹¬ $i < j$› ‹$i \neq j$›
  hence ‹$j < i$› **by** *auto*
  thus ‹*False*› **using** *assms claim'* **by** (*metis less-not-refl*)
qed

lemma *cd-trans0*: **assumes** ‹j *cd*$^{\pi}\rightarrow$ *0*› **and** ‹k *cd*$^{\pi}\rightarrow j$› **shows** ‹k *cd*$^{\pi}\rightarrow$ *0*› **proof** (*rule ccontr*)
  have *path*: ‹*is-path* π› **and** *ij*: ‹$0 < j$› **and** *jk*: ‹$j < k$›
  and *nret*: ‹$\pi\ j \neq$ *return*› ‹$\pi\ k \neq$ *return*›
  and *noipdi*: ‹∀ $l \in \{0..j\}$. $\pi\ l \neq$ *ipd* $(\pi\ 0)$›

and *noipdj*: ‹∀ *l* ∈ {*j*..*k*}. π *l* ≠ *ipd* (π *j*)›
using *assms* **unfolding** *is-cdi-def* **by** *auto*
assume ‹¬ *k* cd^{π}→ *0*›
hence ‹∃ *l* ∈ {*0*..*k*}. π *l* = *ipd* (π *0*)› **unfolding** *is-cdi-def* **using** *path ij jk nret* **by** *force*
then obtain *l* **where** ‹*l* ∈ {*0*..*k*}› **and** *l*: ‹π *l* = *ipd* (π *0*)› **by** *auto*
hence *jl*: ‹*j*<*l*› **and** *lk*: ‹*l*≤*k*› **using** *noipdi ij* **by** *auto*
have *pdj*: ‹*ipd* (π *0*) *pd*→ π *j*› **proof** (*rule ccontr*)
have ‹π *j* ∈ *nodes*› **using** *path* **by** (*metis path-nodes*)
moreover
assume ‹¬ *ipd* (π *0*) *pd*→ π *j*›
ultimately
obtain π′ *n* **where** ∗: ‹*is-path* π′› ‹π′ *0* = π *j*› ‹π′ *n* = *return*› ‹∀ *k*≤*n*. π′ *k* ≠ *ipd*(π *0*)› **using** *no-pd-path* **by** *metis*
hence *path′*: ‹*is-path* (π $@^{j}$ π′)› **by** (*metis path path-cons*)
moreover
have ‹∀ *k* ≤ *j* + *n*. (π$@^{j}$ π′) *k* ≠ *ipd* (π *0*)› **using** *noipdi* ∗(*4*) **by** *auto*
moreover
have ‹(π$@^{j}$ π′) *0* = π *0*› **by** *auto*
moreover
have ‹(π$@^{j}$ π′) (*j* + *n*) = *return*› **using** ∗(*2*,*3*) **by** *auto*
ultimately
have ‹¬ *ipd* (π *0*) *pd*→ π *0*› **unfolding** *is-pd-def* **by** *metis*
thus ‹*False*› **by** (*metis is-ipd-def ij ipd-is-ipd nret*(*1*) *path path-nodes term-path-stable less-imp-le*)
qed
hence ‹(π«*j*) (*l*−*j*) *pd*→ (π«*j*) *0*› **using** *jl l* **by** *auto*
moreover
have ‹*is-path* (π«*j*)› **by** (*metis path path-path-shift*)
moreover
have ‹π *l* ≠ *return*› **by** (*metis lk nret*(*2*) *path term-path-stable*)
hence ‹(π«*j*) (*l*−*j*) ≠ *return*› **using** *jl* **by** *auto*
moreover
have ‹π *j* ≠ *ipd* (π *0*)› **using** *noipdi* **by** *force*
hence ‹(π«*j*) (*l*−*j*) ≠ (π«*j*) *0*› **using** *jl l* **by** *auto*
ultimately
obtain *k′* **where** ‹*k′* ≤ *l*−*j*› **and** ‹(π«*j*) *k′* = *ipd* ((π«*j*) *0*)› **using** *path-pd-ipd0′* **by** *blast*
hence ‹*j* + *k′* ∈ {*j*..*k*}› ‹π (*j*+*k′*) = *ipd* (π *j*)› **using** *jl lk* **by** *auto*
thus ‹*False*› **using** *noipdj* **by** *auto*
qed

lemma *cd-trans*: **assumes** ‹*j* cd^{π}→ *i*› **and** ‹*k* cd^{π}→*j*› **shows** ‹*k* cd^{π}→ *i*› **proof** −
have *path*: ‹*is-path* π› **using** *assms is-cdi-def* **by** *auto*
have *ij*: ‹*i*<*j*› **using** *assms is-cdi-def* **by** *auto*
let ‹*?π*› = ‹π«*i*›
have ‹*j*−*i* $cd^{?\pi}$→ *0*› **using** *assms*(*1*) *cd-path-shift0 path* **by** *auto*
moreover
have ‹*k*−*i* $cd^{?\pi}$→*j*−*i*› **by** (*metis assms*(*2*) *cd-path-shift is-cdi-def ij less-imp-le-nat*)
ultimately
have ‹*k*−*i* $cd^{?\pi}$→ *0*› **using** *cd-trans0* **by** *auto*
thus ‹*k* cd^{π}→ *i*› **using** *path cd-path-shift0* **by** *auto*
qed

lemma *excd-impl-exicd*: **assumes** ‹∃ *k*. *i* cd^{π}→*k*› **shows** ‹∃ *k*. *i* icd^{π}→*k*›
using *assms* **proof**(*induction* ‹*i*› *arbitrary*: ‹π› *rule*: *less-induct*)
case (*less i*)
then obtain *k* **where** *k*: ‹*i* cd^{π}→*k*› **by** *auto*
hence *ip*: ‹*is-path* π› **unfolding** *is-cdi-def* **by** *auto*

show ‹*?case*› **proof** (*cases*)
assume ∗: ‹$\forall\ m \in \{k{<}..{<}i\}.\ \neg\ i\ cd^{\pi}\!\rightarrow m$›
hence ‹$i\ icd^{\pi}\!\rightarrow k$› **using** *k ip* **unfolding** *is-icdi-def* **by** *auto*
thus ‹*?case*› **by** *auto*
next
assume ‹$\neg\ (\forall\ m \in \{k{<}..{<}i\}.\ \neg\ i\ cd^{\pi}\!\rightarrow m)$›
then obtain *m* **where** *m*: ‹$m \in \{k{<}..{<}i\}$› ‹$i\ cd^{\pi}\!\rightarrow m$› **by** *blast*
hence ‹$i - m\ cd^{\pi \ll m}\!\rightarrow 0$› **by** (*metis cd-path-shift0 is-cdi-def*)
moreover
have ‹$i - m < i$› **using** *m* **by** *auto*
ultimately
obtain k' **where** k': ‹$i - m\ icd^{\pi \ll m}\!\rightarrow k'$› **using** *less*(*1*) **by** *blast*
hence ‹$i\ icd^{\pi}\!\rightarrow k' + m$› **using** *ip*
by (*metis add.commute add-diff-cancel-right′ icd-path-shift le-add1*)
thus ‹*?case*› **by** *auto*
qed
qed

lemma *cd-split*: **assumes** ‹$i\ cd^{\pi}\!\rightarrow k$› **and** ‹$\neg\ i\ icd^{\pi}\!\rightarrow k$› **obtains** *m* **where** ‹$i\ icd^{\pi}\!\rightarrow m$› **and** ‹$m\ cd^{\pi}\!\rightarrow k$›
proof −
have *ki*: ‹$k < i$› **using** *assms is-cdi-def* **by** *auto*
obtain *m* **where** *m*: ‹$i\ icd^{\pi}\!\rightarrow m$› **using** *assms*(*1*) **by** (*metis excd-impl-exicd*)
hence ‹$k \le m$› **unfolding** *is-icdi-def* **using** *ki assms*(*1*) **by** *force*
hence *km*: ‹$k < m$›**using** *m assms*(*2*) **by** (*metis le-eq-less-or-eq*)
moreover have ‹$\pi\ m \ne return$› **using** *m* **unfolding** *is-icdi-def is-cdi-def* **by** (*simp*, *metis term-path-stable less-imp-le*)
moreover have ‹$m{<}i$› **using** *m* **unfolding** *is-cdi-def is-icdi-def* **by** *auto*
ultimately
have ‹$m\ cd^{\pi}\!\rightarrow k$› **using** *assms*(*1*) **unfolding** *is-cdi-def* **by** *auto*
with *m that* **show** ‹*thesis*› **by** *auto*
qed

lemma *cd-induct*[*consumes 1*, *case-names base IS*]: **assumes** *prem*: ‹$i\ cd^{\pi}\!\rightarrow k$› **and** *base*: ‹$\bigwedge i.\ i\ icd^{\pi}\!\rightarrow k \Longrightarrow P\ i$›
and *IH*: ‹$\bigwedge k'\ i'.\ k'\ cd^{\pi}\!\rightarrow k \Longrightarrow P\ k' \Longrightarrow i'\ icd^{\pi}\!\rightarrow k' \Longrightarrow P\ i'$› **shows** ‹$P\ i$›
using *prem IH* **proof** (*induction* ‹*i*› *rule*: *less-induct*,*cases*)
case (*less i*)
assume ‹$i\ icd^{\pi}\!\rightarrow k$›
thus ‹$P\ i$› **using** *base* **by** *simp*
next
case (*less i′*)
assume ‹$\neg\ i'\ icd^{\pi}\!\rightarrow k$›
then obtain k' **where** k': ‹ $i'\ icd^{\pi}\!\rightarrow k'$› ‹$k'\ cd^{\pi}\!\rightarrow k$› **using** *less cd-split* **by** *blast*
hence *icdk*: ‹$i'\ cd^{\pi}\!\rightarrow k'$› **using** *is-icdi-def* **by** *auto*
note *ih=less*(*3*)[*OF k′*(*2*) - *k′*(*1*)]
have *ki*: ‹$k' < i'$› **using** *k′ is-icdi-def is-cdi-def* **by** *auto*
have ‹$P\ k'$› **using** *less*(*1*)[*OF ki k′*(*2*)] *less*(*3*) **by** *auto*
thus ‹$P\ i'$› **using** *ih* **by** *simp*
qed

lemma *cdi-prefix*: ‹$n\ cd^{\pi}\!\rightarrow m \Longrightarrow m < n' \Longrightarrow n' \le n \Longrightarrow n'\ cd^{\pi}\!\rightarrow m$› **unfolding** *is-cdi-def*
by (*simp*, *metis term-path-stable*)

lemma *cr-wn′*: **assumes** *1*: ‹$n\ cd^{\pi}\!\rightarrow m$› **and** *nc*: ‹$\neg\ m'\ cd^{\pi}\!\rightarrow m$› **and** *3*: ‹$m < m'$› **shows** ‹$n < m'$›
proof (*rule ccontr*)
assume ‹$\neg\ n < m'$›

hence ‹$m' \leq n$› **by** *simp*
hence ‹m' $cd^{\pi}\rightarrow$ m› **by** (*metis 1 3 cdi-prefix*)
thus ‹*False*› **using** *nc* **by** *simp*
qed

lemma *cr-wn''*: **assumes** ‹i $cd^{\pi}\rightarrow$ m› **and** ‹j $cd^{\pi}\rightarrow$ n› **and** ‹$\neg$ m $cd^{\pi}\rightarrow$ n› **and** ‹$i \leq j$› **shows** ‹$m \leq n$›
proof (*rule ccontr*)
assume ‹$\neg m \leq n$›
hence *nm*: ‹$n < m$› **by** *auto*
moreover
have ‹$m<j$› **using** *assms*(*1*) *assms*(*4*) **unfolding** *is-cdi-def* **by** *auto*
ultimately
have ‹m $cd^{\pi}\rightarrow$ n› **using** *assms*(*2*) *cdi-prefix* **by** *auto*
thus ‹*False*› **using** *assms*(*3*) **by** *auto*
qed

lemma *ret-no-cd*: **assumes** ‹π n = *return*› **shows** ‹$\neg$ n $cd^{\pi}\rightarrow$ k› **by** (*metis assms is-cdi-def*)

lemma *ipd-not-self*: **assumes** ‹$x \in$ *nodes*› ‹$x \neq$ *return*› **shows** ‹$x \neq$ *ipd* x› **by** (*metis is-ipd-def assms ipd-is-ipd*)

lemma *icd-cs*: **assumes** ‹l $icd^{\pi}\rightarrow k$› **shows** ‹cs^{π} l = cs^{π} k @ [π l]›
proof −
from *assms* **have** ‹k = (*THE* k. l $icd^{\pi}\rightarrow$ k)› **by** (*metis icd-is-the-icd*)
with *assms* **show** ‹*?thesis*› **by** *auto*
qed

lemma *cd-not-pd*: **assumes** ‹l $cd^{\pi}\rightarrow$ k› ‹π $l \neq \pi$ k› **shows** ‹$\neg$ π l $pd\rightarrow$ π k› **proof**
assume *pd*: ‹π l $pd\rightarrow$ π k›
have *nret*: ‹π $k \neq$ *return*› **by** (*metis assms*(*1*) *pd pd-ret-is-ret ret-no-cd*)
have *kl*: ‹$k < l$› **by** (*metis is-cdi-def assms*(*1*))
have *path*: ‹*is-path* π› **by** (*metis is-cdi-def assms*(*1*))
from *path-pd-ipd*[*OF path nret assms*(*2*) *pd kl*]
obtain n **where** ‹$k < n$› ‹$n \leq l$› ‹π n = *ipd* (π k)› **.**
thus ‹*False*› **using** *assms*(*1*) **unfolding** *is-cdi-def* **by** *auto*
qed

lemma *cd-ipd-is-cd*: **assumes** ‹$k<m$› ‹π m = *ipd* (π k)› ‹$\forall$ $n \in \{k..<m\}$. π $n \neq$ *ipd* (π k)› **and** *mcdj*: ‹m $cd^{\pi}\rightarrow$ j› **shows** ‹k $cd^{\pi}\rightarrow$ j› **proof** *cases*
assume ‹$j < k$› **thus** ‹k $cd^{\pi}\rightarrow$ j› **by** (*metis mcdj assms*(*1*) *cdi-prefix less-imp-le-nat*)
next
assume ‹$\neg$ $j < k$›
hence *kj*: ‹$k \leq j$› **by** *simp*
have ‹$k < j$› **apply** (*rule ccontr*) **using** *kj assms mcdj* **by** (*auto*, *metis is-cdi-def is-ipd-def cd-not-pd ipd-is-ipd path-nodes term-path-stable less-imp-le*)
moreover
have ‹$j < m$› **using** *mcdj is-cdi-def* **by** *auto*
hence ‹$\forall$ $n \in \{k..j\}$. π $n \neq$ *ipd*(π k)› **using** *assms*(*3*) **by** *force*
ultimately
have ‹j $cd^{\pi}\rightarrow$ k› **by** (*metis mcdj is-cdi-def term-path-stable less-imp-le*)
hence ‹m $cd^{\pi}\rightarrow$ k› **by** (*metis mcdj cd-trans*)
hence ‹*False*› **by** (*metis is-cdi-def is-ipd-def assms*(*2*) *cd-not-pd ipd-is-ipd path-nodes term-path-stable less-imp-le*)
thus ‹*?thesis*› **by** *simp*
qed

lemma *ipd-pd-cd0*: **assumes** *lcd*: ‹n $cd^{\pi}\rightarrow$ *0*› **shows** ‹*ipd* (π *0*) $pd\rightarrow$ (π n)›
proof −

obtain k l **where** $\pi 0$: ‹$\pi\ 0 = k$› **and** πn: ‹$\pi\ n = l$› **and** *cdi*: ‹$n\ cd^{\pi} \rightarrow\ 0$› **using** *lcd* **unfolding** *is-cdi-def* **by** *blast*
have *nret*: ‹$k \neq return$› **by** (*metis is-cdi-def* $\pi 0$ *cdi term-path-stable less-imp-le*)
have *path*: ‹*is-path* π› **and** *ipd*: ‹$\forall\ i{\le}n.\ \pi\ i \neq ipd\ k$› **using** *cdi* **unfolding** *is-cdi-def* $\pi 0$ **by** *auto*
{
fix π' n'
assume *path'*: ‹*is-path* π'›
and $\pi' 0$: ‹$\pi'\ 0 = l$›
and *ret*: ‹$\pi'\ n' = return$›
have ‹*is-path* $(\pi\ @^{n}\ \pi')$› **using** *path path'* πn $\pi' 0$ **by** (*metis path-cons*)
moreover
have ‹$(\pi\ @^{n}\ \pi')\ (n{+}n') = return$› **using** *ret* πn $\pi' 0$ **by** *auto*
moreover
have ‹$(\pi\ @^{n}\ \pi')\ 0 = k$› **using** $\pi 0$ **by** *auto*
moreover
have ‹$ipd\ k\ pd{\rightarrow}\ k$› **by** (*metis is-ipd-def path* $\pi 0$ *ipd-is-ipd nret path-nodes*)
ultimately
obtain k' **where** k': ‹$k' \le n{+}n'$› ‹$(\pi\ @^{n}\ \pi')\ k' = ipd\ k$› **by** (*metis pd-intro*)
have ‹$\neg\ k'{\le}\ n$› **proof**
assume ‹$k' \le n$›
hence ‹$(\pi\ @^{n}\ \pi')\ k' = \pi\ k'$› **by** *auto*
thus ‹*False*› **using** $k'(2)$ *ipd* **by** (*metis* ‹$k' \le n$›)
qed
hence ‹$(\pi\ @^{n}\ \pi')\ k' = \pi'\ (k' - n)$› **by** *auto*
moreover
have ‹$(k' - n) \le n'$› **using** k' **by** *simp*
ultimately
have ‹$\exists\ k'{\le}n'.\ \pi'\ k' = ipd\ k$› **unfolding** k' **by** *auto*
}
moreover
have ‹$l \in nodes$› **by** (*metis* πn *path path-nodes*)
ultimately show ‹$ipd\ (\pi\ 0)\ pd{\rightarrow}\ (\pi\ n)$› **unfolding** *is-pd-def* **by** (*simp add*: $\pi 0$ πn)
qed

lemma *ipd-pd-cd*: **assumes** *lcd*: ‹$l\ cd^{\pi} \rightarrow\ k$› **shows** ‹$ipd\ (\pi\ k)\ pd{\rightarrow}\ (\pi\ l)$›
proof −
have ‹$l{-}k\ cd^{\pi«k}{\rightarrow}0$› **using** *lcd cd-path-shift0 is-cdi-def* **by** *blast*
moreover
note *ipd-pd-cd0*[*OF this*]
moreover
have ‹$(\pi\ «\ k)\ 0 = \pi\ k$› **by** *auto*
moreover
have ‹$k < l$› **using** *lcd* **unfolding** *is-cdi-def* **by** *simp*
then have ‹$(\pi\ «\ k)\ (l - k) = \pi\ l$› **by** *simp*
ultimately show ‹*?thesis*› **by** *simp*
qed

lemma *cd-is-cd-ipd*: **assumes** *km*: ‹$k{<}m$› **and** *ipd*: ‹$\pi\ m = ipd\ (\pi\ k)$› ‹$\forall\ n \in \{k..{<}m\}.\ \pi\ n \neq ipd\ (\pi\ k)$› **and** *cdj*: ‹$k\ cd^{\pi} \rightarrow\ j$› **and** *nipdj*: ‹$ipd\ (\pi\ j) \neq \pi\ m$› **shows** ‹$m\ cd^{\pi} \rightarrow\ j$› **proof** −
have *path*: ‹*is-path* π›
and *jk*: ‹$j < k$›
and *nretj*: ‹$\pi\ k \neq return$›
and *nipd*: ‹$\forall\ l \in \{j..k\}.\ \pi\ l \neq ipd\ (\pi\ j)$› **using** *cdj is-cdi-def* **by** *auto*
have *pd*: ‹$ipd\ (\pi\ j)\ pd{\rightarrow}\ \pi\ m$› **by** (*metis atLeastAtMost-iff cdj ipd*(*1*) *ipd-pd-cd jk le-refl less-imp-le nipd nretj path path-nodes pd-pd-ipd*)
have *nretm*: ‹$\pi\ m \neq return$› **by** (*metis nipdj pd pd-ret-is-ret*)

have *jm*: ‹$j < m$› **using** *jk km* **by** *simp*
show ‹$m\ cd^{\pi}\!\!\rightarrow j$› **proof** (*rule ccontr*)
assume *ncdj*: ‹$\neg\ m\ cd^{\pi}\!\!\rightarrow j$›
hence ‹$\exists\ l \in \{j..m\}.\ \pi\ l = ipd\ (\pi\ j)$› **unfolding** *is-cdi-def* **by** (*metis jm nretm path*)
then obtain *l*
where *jl*: ‹$j \leq l$› **and** ‹$l \leq m$›
and *lipd*: ‹$\pi\ l = ipd\ (\pi\ j)$› **by** *force*
hence *lm*: ‹$l < m$› **using** *nipdj* **by** (*metis le-eq-less-or-eq*)
have *npd*: ‹$\neg\ ipd\ (\pi\ k)\ pd\!\!\rightarrow \pi\ l$› **by** (*metis ipd(1) lipd nipdj pd pd-antisym*)
have *nd*: ‹$\pi\ l \in nodes$› **using** *path path-nodes* **by** *simp*
from *no-pd-path*[*OF nd npd*]
obtain π' *n* **where** *path'*: ‹*is-path* π'› **and** $\pi'0$: ‹$\pi'\ 0 = \pi\ l$› **and** $\pi'n$: ‹$\pi'\ n = return$› **and** *nipd*: ‹$\forall\ ka \leq n.$ $\pi'\ ka \neq ipd\ (\pi\ k)$› **.**
let ‹$?\pi$› = ‹$(\pi @^{l}\ \pi')$ « k›
have *path''*: ‹*is-path* $?\pi$› **by** (*metis* $\pi'0$ *path path' path-cons path-path-shift*)
moreover
have *kl*: ‹$k < l$› **using** *lipd cdj jl* **unfolding** *is-cdi-def* **by** *fastforce*
have ‹$?\pi\ 0 = \pi\ k$› **using** *kl* **by** *auto*
moreover
have ‹$?\pi\ (l + n - k) = return$› **using** $\pi'n$ $\pi'0$ *kl* **by** *auto*
moreover
have ‹$ipd\ (\pi\ k)\ pd\!\!\rightarrow \pi\ k$› **by** (*metis is-ipd-def ipd-is-ipd nretj path path-nodes*)
ultimately
obtain l' **where** l': ‹$l' \leq (l + n - k)$› ‹$?\pi\ l' = ipd\ (\pi\ k)$› **unfolding** *is-pd-def* **by** *blast*
show ‹*False*› **proof** (*cases*)
assume $*$: ‹$k + l' \leq l$›
hence ‹$\pi\ (k + l') = ipd\ (\pi\ k)$› **using** l' **by** *auto*
moreover
have ‹$k + l' < m$› **by** (*metis* $*$ *dual-order.strict-trans2 lm*)
ultimately
show ‹*False*› **using** *ipd(2)* **by** *simp*
next
assume ‹$\neg\ k + l' \leq l$›
hence ‹$\pi'\ (k + l' - l) = ipd\ (\pi\ k)$› **using** l' **by** *auto*
moreover
have ‹$k + l' - l \leq n$› **using** l' *kl* **by** *linarith*
ultimately
show ‹*False*› **using** *nipd* **by** *auto*
qed
qed
qed

lemma *ipd-icd-greatest-cd-not-ipd*: **assumes** *ipd*: ‹$\pi\ m = ipd\ (\pi\ k)$› ‹$\forall\ n \in \{k..<m\}.\ \pi\ n \neq ipd\ (\pi\ k)$›
and *km*: ‹$k < m$› **and** *icdj*: ‹$m\ icd^{\pi}\!\!\rightarrow j$› **shows** ‹$j = (GREATEST\ j.\ k\ cd^{\pi}\!\!\rightarrow j \wedge ipd\ (\pi\ j) \neq \pi\ m)$›
proof –
let ‹$?j$› = ‹$GREATEST\ j.\ k\ cd^{\pi}\!\!\rightarrow j \wedge ipd\ (\pi\ j) \neq \pi\ m$›
have *kcdj*: ‹$k\ cd^{\pi}\!\!\rightarrow j$› **using** *assms cd-ipd-is-cd is-icdi-def* **by** *blast*
have *nipd*: ‹$ipd\ (\pi\ j) \neq \pi\ m$› **using** *icdj* **unfolding** *is-icdi-def is-cdi-def* **by** *auto*
have *bound*: ‹$\bigwedge\ j.\ k\ cd^{\pi}\!\!\rightarrow j \wedge ipd\ (\pi\ j) \neq \pi\ m \Longrightarrow j \leq k$› **unfolding** *is-cdi-def* **by** *simp*
have *exists*: ‹$k\ cd^{\pi}\!\!\rightarrow j \wedge ipd\ (\pi\ j) \neq \pi\ m$› (**is** ‹$?P\ j$›) **using** *kcdj nipd* **by** *auto*
note *GreatestI-nat*[*of* ‹$?P$› - ‹k›, *OF exists*] *Greatest-le-nat*[*of* ‹$?P$› ‹j› ‹k›, *OF exists*]
hence *kcdj'*: ‹$k\ cd^{\pi}\!\!\rightarrow\ ?j$› **and** *ipd'*: ‹$ipd\ (\pi\ ?j) \neq \pi\ m$› **and** *jj*: ‹$j \leq\ ?j$› **using** *bound* **by** *auto*
hence *mcdj'*: ‹$m\ cd^{\pi}\!\!\rightarrow\ ?j$› **using** *ipd km cd-is-cd-ipd* **by** *auto*
show ‹$j =\ ?j$› **proof** (*rule ccontr*)
assume ‹$j \neq\ ?j$›
hence *jlj*: ‹$j <\ ?j$› **using** *jj* **by** *simp*

moreover
have ‹?j < m› using kcdj′ km unfolding is-cdi-def by auto
ultimately
show ‹False› using icdj mcdj′ unfolding is-icdi-def by auto
qed
qed

lemma *cd-impl-icd-cd*: **assumes** ‹$i\ cd^{\pi}\rightarrow l$› **and** ‹$i\ icd^{\pi}\rightarrow k$› **and** ‹$\neg\ i\ icd^{\pi}\rightarrow l$› **shows** ‹$k\ cd^{\pi}\rightarrow l$›
using *assms cd-split icd-uniq* **by** *metis*

lemma *cdi-is-cd-icdi*: **assumes** ‹$k\ icd^{\pi}\rightarrow j$› **shows** ‹$k\ cd^{\pi}\rightarrow i \longleftrightarrow j\ cd^{\pi}\rightarrow i \lor i = j$›
by (*metis assms cd-impl-icd-cd cd-trans icd-imp-cd icd-uniq*)

lemma *same-ipd-stable*: **assumes** ‹$k\ cd^{\pi}\rightarrow i$› ‹$k\ cd^{\pi}\rightarrow j$› ‹$i<j$› ‹$ipd\ (\pi\ i) = ipd\ (\pi\ k)$› **shows** ‹$ipd\ (\pi\ j) = ipd\ (\pi\ k)$›
proof −
have *jcdi*: ‹$j\ cd^{\pi}\rightarrow i$› **by** (*metis is-cdi-def assms(1,2,3) cr-wn′ le-antisym less-imp-le-nat*)
have *1*: ‹$ipd\ (\pi\ j)\ pd\rightarrow \pi\ k$ › **by** (*metis assms(2) ipd-pd-cd*)
have *2*: ‹$ipd\ (\pi\ k)\ pd\rightarrow \pi\ j$ › **by** (*metis assms(4) ipd-pd-cd jcdi*)
have *3*: ‹$ipd\ (\pi\ k)\ pd\rightarrow (ipd\ (\pi\ j))$› **by** (*metis 2 IFC-def.is-cdi-def assms(1,2,4) atLeastAtMost-iff jcdi less-imp-le pd-node2 pd-pd-ipd*)
have *4*: ‹$ipd\ (\pi\ j)\ pd\rightarrow (ipd\ (\pi\ k))$› **by** (*metis 1 2 IFC-def.is-ipd-def assms(2) cd-not-pd ipd-is-ipd jcdi pd-node2 ret-no-cd*)
show ‹*?thesis*› **using** *3 4 pd-antisym* **by** *simp*
qed

lemma *icd-pd-intermediate′*: **assumes** *icd*: ‹$i\ icd^{\pi}\rightarrow k$› **and** *j*: ‹$k < j$› ‹$j < i$› **shows** ‹$\pi\ i\ pd\rightarrow (\pi\ j)$›
using *j* **proof** (*induction* ‹$i - j$› *arbitrary*: ‹*j*› *rule*: *less-induct*)
case (*less j*)
have ‹$\neg\ i\ cd^{\pi}\rightarrow j$› **using** *less.prems icd* **unfolding** *is-icdi-def* **by** *force*
moreover
have ‹*is-path* π› **using** *icd* **by** (*metis is-icdi-def*)
moreover
have ‹$\pi\ i \neq return$› **using** *icd* **by** (*metis is-icdi-def ret-no-cd*)
ultimately
have ‹$\exists\ l.\ j \le l \land l \le i \land \pi\ l = ipd\ (\pi\ j)$› **unfolding** *is-cdi-def* **using** *less.prems* **by** *auto*
then obtain *l* **where** *l*: ‹$j \le l$› ‹$l \le i$› ‹$\pi\ l = ipd\ (\pi\ j)$› **by** *blast*
hence *lpd*: ‹$\pi\ l\ pd\rightarrow (\pi\ j)$› **by** (*metis is-ipd-def* ‹$\pi\ i \neq return$› ‹*is-path* π› *ipd-is-ipd path-nodes term-path-stable*)
show ‹*?case*› **proof** (*cases*)
assume ‹$l = i$›
thus ‹*?case*› **using** *lpd* **by** *auto*
next
assume ‹$l \neq i$›
hence ‹$l < i$› **using** *l* **by** *simp*
moreover
have ‹$j \neq l$› **using** *l* **by** (*metis is-ipd-def* ‹$\pi\ i \neq return$› ‹*is-path* π› *ipd-is-ipd path-nodes term-path-stable*)
hence ‹$j < l$› **using** *l* **by** *simp*
moreover
hence ‹$i - l < i - j$› **by** (*metis diff-less-mono2 less.prems(2)*)
moreover
have ‹$k < l$› **by** (*metis l(1) less.prems(1) linorder-neqE-nat not-le order.strict-trans*)
ultimately
have ‹$\pi\ i\ pd\rightarrow (\pi\ l)$› **using** *less.hyps* **by** *auto*
thus ‹*?case*› **using** *lpd* **by** (*metis pd-trans*)
qed
qed

lemma *icd-pd-intermediate*: **assumes** *icd*: ‹*i* $icd^{\pi}\to$ *k*› **and** *j*: ‹*k* < *j*› ‹*j* ≤ *i*› **shows** ‹π *i pd*→ (π *j*)›
using *assms icd-pd-intermediate'*[*OF assms*(*1*,*2*)] **apply** (*cases* ‹*j* < *i*›,*metis*) **by** (*metis is-icdi-def le-neq-trans path-nodes pd-refl*)

lemma *no-icd-pd*: **assumes** *path*: ‹*is-path* π› **and** *noicd*: ‹∀ *l*≥*n*. ¬ *k* $icd^{\pi}\to$ *l*› **and** *nk*: ‹*n* ≤ *k*› **shows** ‹π *k pd*→ π *n*›
proof *cases*
  assume ‹π *k* = *return*› **thus** ‹*?thesis*› **by** (*metis path path-nodes return-pd*)
next
  assume *nret*: ‹π *k* ≠ *return*›
  have *nocd*: ‹⋀ *l*. *n*≤*l* ⟹ ¬ *k* $cd^{\pi}\to$ *l*› **proof**
    fix *l* **assume** *kcd*: ‹*k* $cd^{\pi}\to$ *l*› **and** *nl*: ‹*n* ≤ *l*›
    hence ‹(*k* − *n*) $cd^{\pi«n}\to$ (*l* − *n*)› **using** *cd-path-shift*[*OF nl path*] **by** *simp*
    hence ‹∃ *l*. (*k* − *n*) $icd^{\pi«n}\to$ *l*› **using** *excd-impl-exicd* **by** *blast*
    then obtain *l'* **where** *k* − *n* $icd^{\pi\ «\ n}\to$ *l'* **..**
    hence ‹*k* $icd^{\pi}\to$ (*l'* + *n*)› **using** *icd-path-shift*[*of* ‹*n*› ‹*l'* + *n*› ‹π› ‹*k*›] *path* **by** *auto*
    thus ‹*False*› **using** *noicd* **by** *auto*
  qed
  hence ‹⋀*l*. *n* ≤ *l* ⟹ *l*<*k* ⟹ ∃ *j* ∈ {*l*..*k*}. π *j* = *ipd* (π *l*)› **using** *path nret* **unfolding** *is-cdi-def* **by** *auto*
  thus ‹*?thesis*› **using** *nk* **proof** (*induction* ‹*k* − *n*› *arbitrary*: ‹*n*› *rule*: *less-induct*,*cases*)
    case (*less n*)
    assume ‹*n* = *k*›
    thus ‹*?case*› **using** *pd-refl path path-nodes* **by** *auto*
  next
    case (*less n*)
    assume ‹*n* ≠ *k*›
    hence *nk*: ‹*n* < *k*› **using** *less*(*3*) **by** *auto*
    with *less*(*2*) **obtain** *j* **where** *jnk*: ‹*j* ∈ {*n*..*k*}› **and** *ipdj*: ‹π *j* = *ipd* (π *n*)› **by** *blast*
    have *nretn*: ‹π *n* ≠ *return*› **using** *nk nret term-path-stable path* **by** *auto*
    with *ipd-is-ipd path path-nodes is-ipd-def ipdj*
    have *jpdn*: ‹π *j pd*→ π *n*› **by** *auto*
    show ‹*?case*› **proof** *cases*
      assume ‹*j* = *k*› **thus** ‹*?case*› **using** *jpdn* **by** *simp*
    next
      assume ‹*j* ≠ *k*›
      hence *jk*: ‹*j* < *k*› **using** *jnk* **by** *auto*
      have ‹*j* ≠ *n*› **using** *ipdj* **by** (*metis ipd-not-self nretn path path-nodes*)
      hence *nj*: ‹*n* < *j*› **using** *jnk* **by** *auto*
      have ∗: ‹*k* − *j* < *k* − *n*› **using** *jk nj* **by** *auto*

      with *less*(*1*)[*OF* ∗] *less*(*2*) *jk nj*
      have ‹π *k pd*→ π *j*› **by** *auto*

      thus ‹*?thesis*› **using** *jpdn pd-trans* **by** *metis*
    qed
  qed
qed

lemma *first-pd-no-cd*: **assumes** *path*: ‹*is-path* π› **and** *pd*: ‹π *n pd*→ π *0*› **and** *first*: ‹∀ *l* < *n*. π *l* ≠ π *n*›
shows ‹∀ *l*. ¬ *n* $cd^{\pi}\to$ *l*›
proof (*rule ccontr*, *goal-cases*)
  case *1*
  then obtain *l* **where** *ncdl*: ‹*n* $cd^{\pi}\to$ *l*› **by** *blast*
  hence *ln*: ‹*l* < *n*› **using** *is-cdi-def* **by** *auto*

have ‹¬ π *n pd*→ π *l*› **using** *ncdl cd-not-pd* **by** (*metis ln first*)
then obtain π' n' **where** *path'*: ‹*is-path* π'› **and** $\pi 0$: ‹π' *0* = π *l*› **and** πn: ‹π' n' = *return*› **and** *not*πn: ‹∀ *j*≤ n'. π' *j* ≠ π *n*› **unfolding** *is-pd-def* **using** *path path-nodes* **by** *auto*
let ‹*?*π› = ‹π@^{l} π'›

have ‹*is-path ?*π› **by** (*metis* $\pi 0$ *path path' path-cons*)
moreover
have ‹*?*π *0* = π *0*› **by** *auto*
moreover
have ‹*?*π (n' + *l*) = *return*› **using** $\pi 0$ πn **by** *auto*
ultimately
obtain *j* **where** *j*: ‹*j* ≤ n' + *l*› **and** *jn*: ‹*?*π *j* = π *n*› **using** *pd* **unfolding** *is-pd-def* **by** *blast*
show ‹*False*› **proof** *cases*
  assume ‹*j* ≤ *l*› **thus** ‹*False*› **using** *jn first ln* **by** *auto*
next
  assume ‹¬ *j* ≤ *l*› **thus** ‹*False*› **using** *j jn not*πn **by** *auto*
qed
qed

lemma *first-pd-no-icd*: **assumes** *path*: ‹*is-path* π› **and** *pd*: ‹π *n pd*→ π *0*› **and** *first*: ‹∀ *l* < *n*. π *l* ≠ π *n*› **shows** ‹∀ *l*. ¬ *n icd*$^\pi$→ *l*›
by (*metis first first-pd-no-cd icd-imp-cd path pd*)

lemma *path-nret-ex-nipd*: **assumes** ‹*is-path* π› ‹∀ *i*. π *i* ≠ *return*› **shows** ‹∀ *i*. (∃ *j*≥*i*. (∀ *k*>*j*. π *k* ≠ *ipd* (π *j*)))› **proof**(*rule*, *rule ccontr*)
fix *i*
assume ‹¬ (∃ *j*≥*i*. ∀ *k*>*j*. π *k* ≠ *ipd* (π *j*))›
hence ∗: ‹∀ *j*≥*i*. (∃ *k*>*j*. π *k* = *ipd* (π *j*))› **by** *blast*
have ‹∀ *j*. (∃ *k*>*j*. (π«*i*) *k* = *ipd* ((π«*i*) *j*))› **proof**
  fix *j*
  have ‹*i* + *j* ≥ *i*› **by** *auto*
  then obtain *k* **where** *k*: ‹*k*>*i*+*j*› ‹π *k* = *ipd* (π (*i*+*j*))› **using** ∗ **by** *blast*
  hence ‹(π«*i*) (*k* − *i*) = *ipd* ((π«*i*) *j*)› **by** *auto*
  moreover
  have ‹*k* − *i* > *j*› **using** *k* **by** *auto*
  ultimately
  show ‹∃ *k*>*j*. (π«*i*) *k* = *ipd* ((π«*i*) *j*)› **by** *auto*
qed
moreover
have ‹*is-path* (π«*i*)› **using** *assms*(*1*) *path-path-shift* **by** *simp*
ultimately
obtain *k* **where** ‹(π«*i*) *k* = *return*› **using** *all-ipd-imp-ret* **by** *blast*
thus ‹*False*› **using** *assms*(*2*) **by** *auto*
qed

lemma *path-nret-ex-all-cd*: **assumes** ‹*is-path* π› ‹∀ *i*. π *i* ≠ *return*› **shows** ‹∀ *i*. (∃ *j*≥*i*. (∀ *k*>*j*. *k cd*$^\pi$→ *j*))›
unfolding *is-cdi-def* **using** *assms path-nret-ex-nipd*[*OF assms*] **by** (*metis atLeastAtMost-iff ipd-not-self linorder-neqE-nat not-le path-nodes*)

lemma *path-nret-inf-all-cd*: **assumes** ‹*is-path* π› ‹∀ *i*. π *i* ≠ *return*› **shows** ‹¬ *finite* {*j*. ∀ *k*>*j*. *k cd*$^\pi$→ *j*}›
using *unbounded-nat-set-infinite path-nret-ex-all-cd*[*OF assms*] **by** *auto*

lemma *path-nret-inf-icd-seq*: **assumes** *path*: ‹*is-path* π› **and** *nret*: ‹∀ *i*. π *i* ≠ *return*›
obtains *f* **where** ‹∀ *i*. *f* (*Suc i*) *icd*$^\pi$→ *f i*› ‹*range f* = {*i*. ∀ *j*>*i*. *j cd*$^\pi$→ *i*}› ‹¬ (∃ *i*. *f 0 cd*$^\pi$→ *i*)›
proof −

note *path-nret-inf-all-cd*[*OF assms*]
then obtain *f* **where** *ran*: ‹*range f* = {*j*. ∀ *k*>*j*. *k* $cd^{\pi}\rightarrow$ *j*}› **and** *asc*: ‹∀ *i*. *f i* < *f* (*Suc i*)› **using** *infinite-ascending* **by** *blast*
have *mono*: ‹∀ *i j*. *i* < *j* ⟶ *f i* < *f j*› **using** *asc* **by** (*metis lift-Suc-mono-less*)
{
fix *i*
have *cd*: ‹*f* (*Suc i*) $cd^{\pi}\rightarrow$ *f i*› **using** *ran asc* **by** *auto*
have ‹*f* (*Suc i*) $icd^{\pi}\rightarrow$ *f i*› **proof** (*rule ccontr*)
assume ‹¬ *f* (*Suc i*) $icd^{\pi}\rightarrow$ *f i*›
then obtain *m* **where** *im*: ‹*f i* < *m*› **and** *mi*: ‹ *m* < *f* (*Suc i*)› **and** *cdm*: ‹*f* (*Suc i*) $cd^{\pi}\rightarrow$ *m*› **unfolding** *is-icdi-def* **using** *assms*(*1*) *cd* **by** *auto*
have ‹∀ *k*>*m*. *k* $cd^{\pi}\rightarrow$*m*› **proof** (*rule*,*rule*,*cases*)
fix *k* **assume** ‹*f* (*Suc i*) < *k*›
hence ‹*k* $cd^{\pi}\rightarrow$ *f* (*Suc i*)› **using** *ran* **by** *auto*
thus ‹*k* $cd^{\pi}\rightarrow$ *m*› **using** *cdm cd-trans* **by** *metis*
next
fix *k* **assume** *mk*: ‹*m* < *k*› **and** ‹¬ *f* (*Suc i*) < *k*›
hence *ik*: ‹*k* ≤ *f* (*Suc i*)› **by** *simp*
thus ‹*k* $cd^{\pi}\rightarrow$ *m*› **using** *cdm* **by** (*metis cdi-prefix mk*)
qed
hence ‹*m* ∈ *range f*› **using** *ran* **by** *blast*
then obtain *j* **where** *m*: ‹*m* = *f j*› **by** *blast*
show ‹*False*› **using** *im mi mono* **unfolding** *m* **by** (*metis Suc-lessI le-less not-le*)
qed
}
moreover
{
fix *m*
assume *cdm*: ‹*f 0* $cd^{\pi}\rightarrow$ *m*›
have ‹∀ *k*>*m*. *k* $cd^{\pi}\rightarrow$*m*› **proof** (*rule*,*rule*,*cases*)
fix *k* **assume** ‹*f 0* < *k*›
hence ‹*k* $cd^{\pi}\rightarrow$ *f 0*› **using** *ran* **by** *auto*
thus ‹*k* $cd^{\pi}\rightarrow$ *m*› **using** *cdm cd-trans* **by** *metis*
next
fix *k* **assume** *mk*: ‹*m* < *k*› **and** ‹¬ *f 0* < *k*›
hence *ik*: ‹*k* ≤ *f 0*› **by** *simp*
thus ‹*k* $cd^{\pi}\rightarrow$ *m*› **using** *cdm* **by** (*metis cdi-prefix mk*)
qed
hence ‹*m* ∈ *range f*› **using** *ran* **by** *blast*
then obtain *j* **where** *m*: ‹*m* = *f j*› **by** *blast*
hence *fj0*: ‹*f j* < *f 0*› **using** *cdm m is-cdi-def* **by** *auto*
hence ‹*0* < *j*› **by** (*metis less-irrefl neq0-conv*)
hence ‹*False*› **using** *fj0 mono* **by** *fastforce*
}
ultimately show ‹*thesis*› **using** *that ran* **by** *blast*
qed

lemma *cdi-iff-no-strict-pd*: ‹*i* $cd^{\pi}\rightarrow$ *k* ⟷ *is-path* π ∧ *k* < *i* ∧ π *i* ≠ *return* ∧ (∀ *j* ∈ {*k*..*i*}. ¬ (π *k*, π *j*) ∈ *pdt*)›
proof
assume *cd*:‹*i* $cd^{\pi}\rightarrow$ *k*›
have *1*: ‹*is-path* π ∧ *k* < *i* ∧ π *i* ≠ *return*› **using** *cd* **unfolding** *is-cdi-def* **by** *auto*
have *2*: ‹∀ *j* ∈ {*k*..*i*}. ¬ (π *k*, π *j*) ∈ *pdt*› **proof** (*rule ccontr*)
assume ‹ ¬ (∀ *j*∈{*k*..*i*}. (π *k*, π *j*) ∉ *pdt*)›
then obtain *j* **where** ‹*j* ∈ {*k*..*i*}› **and** ‹(π *k*, π *j*) ∈ *pdt*› **by** *auto*
hence ‹π *j* ≠ π *k*› **and** ‹π *j* *pd*→ π *k*› **unfolding** *pdt-def* **by** *auto*

thus ‹*False*› **using** *path-pd-ipd* **by** (*metis* ‹$j \in \{k..i\}$› *atLeastAtMost-iff cd cd-not-pd cdi-prefix le-eq-less-or-eq*)

qed
show ‹$\textit{is-path}\ \pi \wedge k < i \wedge \pi\ i \neq \textit{return} \wedge (\forall\ j \in \{k..i\}.\ \neg\ (\pi\ k,\ \pi\ j) \in \textit{pdt})$› **using** *1 2* **by** *simp*
next
assume ‹$\textit{is-path}\ \pi \wedge k < i \wedge \pi\ i \neq \textit{return} \wedge (\forall\ j \in \{k..i\}.\ \neg\ (\pi\ k,\ \pi\ j) \in \textit{pdt})$›
thus ‹$i\ cd^{\pi}\!\!\rightarrow k$› **by** (*metis ipd-in-pdt term-path-stable less-or-eq-imp-le not-cd-impl-ipd path-nodes*)
qed

2.5 Facts about Control Slices

lemma *last-cs*: ‹$\textit{last}\ (cs^{\pi}\ i) = \pi\ i$› **by** *auto*

lemma *cs-not-nil*: ‹$cs^{\pi}\ n \neq []$› **by** (*auto*)

lemma *cs-return*: **assumes** ‹$\pi\ n = \textit{return}$› **shows** ‹$cs^{\pi}\ n = [\pi\ n]$› **by** (*metis assms cs.elims icd-imp-cd ret-no-cd*)

lemma *cs-0*[*simp*]: ‹$cs^{\pi}\ 0 = [\pi\ 0]$› **using** *is-icdi-def is-cdi-def* **by** *auto*

lemma *cs-inj*: **assumes** ‹$\textit{is-path}\ \pi$› ‹$\pi\ n \neq \textit{return}$› ‹$cs^{\pi}\ n = cs^{\pi}\ n'$› **shows** ‹$n = n'$›
using *assms* **proof** (*induction* ‹$cs^{\pi}\ n$› *arbitrary*: ‹π› ‹n› ‹n'› *rule*:*rev-induct*)
case *Nil* **hence** ‹*False*› **using** *cs-not-nil* **by** *metis* **thus** ‹*?case*› **by** *simp*
next
case (*snoc x xs* π *n n'*) **show** ‹*?case*› **proof** (*cases* ‹*xs*›)
case *Nil*
hence $*$: ‹$\neg\ (\exists\ k.\ n\ icd^{\pi}\!\!\rightarrow k)$› **using** *snoc*(*2*) *cs-not-nil*
by (*auto*,*metis append1-eq-conv append-Nil cs-not-nil*)
moreover
have ‹$[x] = cs^{\pi}\ n'$› **using** *Nil snoc* **by** *auto*
hence $**$: ‹$\neg\ (\exists\ k.\ n'\ icd^{\pi}\!\!\rightarrow k)$› **using** *cs-not-nil*
by (*auto*,*metis append1-eq-conv append-Nil cs-not-nil*)
ultimately
have ‹$\forall\ k.\ \neg\ n\ cd^{\pi}\!\!\rightarrow k$› ‹$\forall\ k.\ \neg\ n'\ cd^{\pi}\!\!\rightarrow k$› **using** *excd-impl-exicd* **by** *auto blast+*
moreover
hence ‹$\pi\ n = \pi\ n'$› **using** *snoc*(*5*,*2*) **by** *auto* (*metis* $*$ $**$ *list.inject*)
ultimately
show ‹$n = n'$› **using** *other-claim' snoc* **by** *blast*
next
case (*Cons y ys*)
hence $*$: ‹$\exists\ k.\ n\ icd^{\pi}\!\!\rightarrow k$› **using** *snoc*(*2*) **by** *auto* (*metis append-is-Nil-conv list.distinct*(*1*) *list.inject*)
then obtain *k* **where** *k*: ‹$n\ icd^{\pi}\!\!\rightarrow k$› **by** *auto*
have ‹$k = (\textit{THE}\ k\ .\ n\ icd^{\pi}\!\!\rightarrow k)$› **using** *k* **by** (*metis icd-is-the-icd*)
hence *xsk*: ‹$xs = cs^{\pi}\ k$› **using** $*$ *k snoc*(*2*) **unfolding** *cs.simps*[*of* ‹π› ‹n›] **by** *auto*
have $**$: ‹$\exists\ k.\ n'\ icd^{\pi}\!\!\rightarrow k$› **using** *snoc*(*2*)[*unfolded snoc*(*5*)] **by** *auto* (*metis Cons append1-eq-conv append-Nil list.distinct*(*1*))
then obtain *k'* **where** *k'*: ‹$n'\ icd^{\pi}\!\!\rightarrow k'$› **by** *auto*
hence ‹$k' = (\textit{THE}\ k\ .\ n'\ icd^{\pi}\!\!\rightarrow k)$› **using** *k'* **by** (*metis icd-is-the-icd*)
hence *xsk'*: ‹$xs = cs^{\pi}\ k'$› **using** $**$ *k' snoc*(*5*,*2*) **unfolding** *cs.simps*[*of* ‹π› ‹n'›] **by** *auto*
hence ‹$cs^{\pi}\ k = cs^{\pi}\ k'$› **using** *xsk* **by** *simp*
moreover
have *kn*: ‹$k < n$› **using** *k* **by** (*metis is-icdi-def is-cdi-def*)
hence ‹$\pi\ k \neq \textit{return}$› **using** *snoc* **by** (*metis term-path-stable less-imp-le*)
ultimately
have *kk'*[*simp*]: ‹$k' = k$› **using** *snoc*(*1*) *xsk snoc*(*3*) **by** *metis*
have *nk0*: ‹$n - k\ icd^{\pi«k}\!\!\rightarrow 0$› ‹$n' - k\ icd^{\pi«k}\!\!\rightarrow 0$› **using** *k k' icd-path-shift0 snoc*(*3*) **by** *auto*

moreover
have *nkr*: ‹$(\pi«k)(n{-}k) \neq$ *return*› **using** *snoc*(*4*) *kn* **by** *auto*
moreover
have ‹*is-path* $(\pi«k)$› **by** (*metis path-path-shift snoc.prems*(*1*))
moreover
have *kn'*: ‹$k < n'$› **using** *k' kk'* **by** (*metis is-icdi-def is-cdi-def*)
have ‹$\pi\ n = \pi\ n'$› **using** *snoc*(*5*) $*$ $**$ **by** *auto*
hence ‹$(\pi«k)(n{-}k) = (\pi«k)(n'{-}k)$› **using** *kn kn'* **by** *auto*
ultimately
have ‹$n - k = n' - k$› **using** *other-claim* **by** *auto*
thus ‹$n = n'$› **using** *kn kn'* **by** *auto*
qed
qed

lemma *cs-cases*: **fixes** π *i*
obtains (*base*) ‹$cs^{\pi}\ i = [\pi\ i]$› **and** ‹$\forall\ k.\ \neg\ i\ cd^{\pi}\rightarrow k$› |
(*depend*) *k* **where** ‹$cs^{\pi}\ i = (cs^{\pi}\ k)@[\pi\ i]$› **and** ‹$i\ icd^{\pi}\rightarrow k$›
proof *cases*
assume $*$: ‹$\exists\ k.\ i\ icd^{\pi}\rightarrow k$›
then obtain *k* **where** *k*: ‹$i\ icd^{\pi}\rightarrow k$› **..**
hence ‹$k = (THE\ k.\ i\ icd^{\pi}\rightarrow k)$› **by** (*metis icd-is-the-icd*)
hence ‹$cs^{\pi}\ i = (cs^{\pi}\ k)@[\pi\ i]$› **using** $*$ **by** *auto*
with *k that* **show** ‹*thesis*› **by** *simp*
next
assume $*$: ‹$\neg\ (\exists\ k.\ i\ icd^{\pi}\rightarrow k)$›
hence ‹$\forall\ k.\ \neg\ i\ cd^{\pi}\rightarrow k$› **by** (*metis excd-impl-exicd*)
moreover
have ‹$cs^{\pi}\ i = [\pi\ i]$› **using** $*$ **by** *auto*
ultimately
show ‹*thesis*› **using** *that* **by** *simp*
qed

lemma *cs-length-one*: **assumes** ‹*length* $(cs^{\pi}\ i) = 1$› **shows** ‹$cs^{\pi}\ i = [\pi\ i]$› **and** ‹$\forall\ k.\ \neg\ i\ cd^{\pi}\rightarrow k$›
apply (*cases* ‹*i*› ‹π› *rule*: *cs-cases*)
using *assms cs-not-nil*
apply *auto*
apply (*cases* ‹*i*› ‹π› *rule*: *cs-cases*)
using *assms cs-not-nil*
by *auto*

lemma *cs-length-g-one*: **assumes** ‹*length* $(cs^{\pi}\ i) \neq 1$› **obtains** *k* **where** ‹$cs^{\pi}\ i = (cs^{\pi}\ k)@[\pi\ i]$› **and** ‹$i$ $icd^{\pi}\rightarrow k$›
apply (*cases* ‹*i*› ‹π› *rule*: *cs-cases*)
using *assms cs-not-nil* **by** *auto*

lemma *claim*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π'› **and** *ii*: ‹$cs^{\pi}\ i = cs^{\pi'}\ i'$› **and** *jj*: ‹$cs^{\pi}\ j = cs^{\pi'}\ j'$›
and *bl*: ‹*butlast* $(cs^{\pi}\ i) =$ *butlast* $(cs^{\pi}\ j)$› **and** *nret*: ‹$\pi\ i \neq$ *return*› **and** *ilj*: ‹$i < j$›
shows ‹$i' < j'$›
proof (*cases*)
assume $*$: ‹*length* $(cs^{\pi}\ i) = 1$›
hence $**$: ‹*length* $(cs^{\pi}\ i) = 1$› ‹*length* $(cs^{\pi}\ j) = 1$› ‹*length* $(cs^{\pi'}\ i') = 1$› ‹*length* $(cs^{\pi'}\ j') = 1$›
apply *metis*
apply (*metis* $*$ *bl butlast.simps*(*2*) *butlast-snoc cs-length-g-one cs-length-one*(*1*) *cs-not-nil*)
apply (*metis* $*$ *ii*)
by (*metis* $*$ *bl butlast.simps*(*2*) *butlast-snoc cs-length-g-one cs-length-one*(*1*) *cs-not-nil jj*)

then obtain ‹*cs*$^{\pi}$ *i* = [π *i*]› ‹*cs*$^{\pi}$ *j* = [π *j*]› ‹*cs*$^{\pi'}$ *j′* = [π′ *j′*]› ‹*cs*$^{\pi'}$ *i′*= [π′ *i′*]›
‹∀ *k*. ¬ *j cd*$^{\pi}$→ *k*› ‹∀ *k*. ¬ *i′ cd*$^{\pi'}$→ *k*› ‹∀ *k*. ¬ *j′ cd*$^{\pi'}$→ *k*›
by (*metis cs-length-one* ∗∗)
moreover
hence ‹π *i* = π′ *i′*› ‹π *j* = π′ *j′*› **using** *assms* **by** *auto*
ultimately
show ‹*i′* < *j′*› **using** *nret ilj path claim″* **by** *blast*
next
assume ∗: ‹*length* (*cs*$^{\pi}$ *i*) ≠ *1*›
hence ∗∗: ‹*length* (*cs*$^{\pi}$ *i*) ≠ *1*› ‹*length* (*cs*$^{\pi}$ *j*) ≠ *1*› ‹*length* (*cs*$^{\pi'}$ *i′*) ≠ *1*› ‹*length* (*cs*$^{\pi'}$ *j′*) ≠ *1*›
apply *metis*
apply (*metis* ∗ *bl butlast.simps*(*2*) *butlast-snoc cs-length-g-one cs-length-one*(*1*) *cs-not-nil*)
apply (*metis* ∗ *ii*)
by (*metis* ∗ *bl butlast.simps*(*2*) *butlast-snoc cs-length-g-one cs-length-one*(*1*) *cs-not-nil jj*)
obtain *k l k′ l′* **where** ∗∗∗:
‹*cs*$^{\pi}$ *i* = (*cs*$^{\pi}$ *k*)@[π *i*]› ‹*cs*$^{\pi}$ *j* = (*cs*$^{\pi}$ *l*)@[π *j*]› ‹*cs*$^{\pi'}$ *i′* = (*cs*$^{\pi'}$ *k′*)@[π′ *i′*]› ‹*cs*$^{\pi'}$ *j′* = (*cs*$^{\pi'}$ *l′*)@[π′ *j′*]›
and
icds: ‹*i icd*$^{\pi}$→ *k*› ‹*j icd*$^{\pi}$→ *l*› ‹*i′ icd*$^{\pi'}$→ *k′*› ‹*j′ icd*$^{\pi'}$→ *l′*›
by (*metis* ∗∗ *cs-length-g-one*)
hence ‹*cs*$^{\pi}$ *k* = *cs*$^{\pi}$ *l*› ‹*cs*$^{\pi'}$ *k′* = *cs*$^{\pi'}$ *l′*› **using** *assms* **by** *auto*
moreover
have ‹π *k* ≠ *return*› ‹π′ *k′* ≠ *return*› **using** *nret*
apply (*metis is-icdi-def icds*(*1*) *is-cdi-def term-path-stable less-imp-le*)
by (*metis is-cdi-def is-icdi-def icds*(*3*) *term-path-stable less-imp-le*)
ultimately
have *lk*[*simp*]: ‹*l* = *k*› ‹*l′* = *k′*› **using** *path cs-inj* **by** *auto*
let ‹*?π*› = ‹π « *k*›
let ‹*?π′*› = ‹π′«*k′*›
have ‹*i*−*k icd*$^{?\pi}$→ *0*› ‹*j*−*k icd*$^{?\pi}$→ *0*› ‹*i′*−*k′ icd*$^{?\pi'}$→ *0*› ‹*j′*−*k′ icd*$^{?\pi'}$→ *0*› **using** *icd-path-shift0 path icds* **by** *auto*
moreover
have *ki*: ‹*k* < *i*› **using** *icds* **by** (*metis is-icdi-def is-cdi-def*)
hence ‹*i*−*k* < *j*−*k*› **by** (*metis diff-is-0-eq diff-less-mono ilj nat-le-linear order.strict-trans*)
moreover
have π*i*: ‹π *i* = π′ *i′*› ‹π *j* = π′ *j′*› **using** *assms* ∗∗∗ **by** *auto*
have ‹*k′* < *i′*› ‹*k′* < *j′*› **using** *icds* **unfolding** *lk* **by** (*metis is-cdi-def is-icdi-def*)+
hence ‹*?π* (*i*−*k*) = *?π′* (*i′*−*k′*)› ‹*?π* (*j*−*k*) = *?π′* (*j′*−*k′*)› **using** π*i ki ilj* **by** *auto*
moreover
have ‹*?π* (*i*−*k*) ≠ *return*› **using** *nret ki* **by** *auto*
moreover
have ‹*is-path ?π*› ‹*is-path ?π′*› **using** *path path-path-shift* **by** *auto*
ultimately
have ‹*i′*−*k′* < *j′* − *k′*› **using** *claim′* **by** *blast*
thus ‹*i′* < *j′*› **by** (*metis diff-is-0-eq diff-less-mono less-nat-zero-code linorder-neqE-nat nat-le-linear*)
qed

lemma *cs-split′*: **assumes** ‹*cs*$^{\pi}$ *i* = *xs*@[*x*,*x′*]@*ys*› **shows** ‹∃ *m*. *cs*$^{\pi}$ *m* = *xs*@[*x*] ∧ *i cd*$^{\pi}$→ *m*›
using *assms* **proof** (*induction* ‹*ys*› *arbitrary*: ‹*i*› *rule*:*rev-induct*)
case (*snoc y ys*)
hence ‹*length* (*cs*$^{\pi}$ *i*) ≠ *1*› **by** *auto*
then obtain *i′* **where** ‹*cs*$^{\pi}$ *i* = (*cs*$^{\pi}$ *i′*) @ [π *i*]› **and** ∗: ‹*i icd*$^{\pi}$→ *i′*› **using** *cs-length-g-one*[*of* ‹π› ‹*i*›] **by** *metis*
hence ‹*cs*$^{\pi}$ *i′* = *xs*@[*x*,*x′*]@*ys*› **using** *snoc*(*2*) **by** (*metis append1-eq-conv append-assoc*)
then obtain *m* **where** ∗∗: ‹*cs*$^{\pi}$ *m* = *xs* @ [*x*]› **and** ‹*i′ cd*$^{\pi}$→ *m*› **using** *snoc*(*1*) **by** *blast*
hence ‹*i cd*$^{\pi}$→ *m*› **using** ∗ *cd-trans* **by** (*metis is-icdi-def*)

with $**$ **show** ‹*?case*› **by** *blast*
next
case *Nil*
hence ‹*length* $(cs^{\pi}\ i) \neq 1$› **by** *auto*
then obtain i' **where** *a*: ‹$cs^{\pi}\ i = (cs^{\pi}\ i')$ @ $[\pi\ i]$› **and** $*$: ‹$i\ icd^{\pi}\!\rightarrow i'$› **using** *cs-length-g-one*[*of* ‹π› ‹*i*›] **by** *metis*
have ‹$cs^{\pi}\ i = (xs@[x])@[x']$› **using** *Nil* **by** *auto*
hence ‹$cs^{\pi}\ i' = xs@[x]$› **using** *append1-eq-conv a* **by** *metis*
thus ‹*?case*› **using** $*$ **unfolding** *is-icdi-def* **by** *blast*
qed

lemma *cs-split*: **assumes** ‹$cs^{\pi}\ i = xs@[x]@ys@[\pi\ i]$› **shows** ‹$\exists\ m.\ cs^{\pi}\ m = xs@[x] \wedge i\ cd^{\pi}\!\rightarrow m$› **proof** −
obtain $x'\ ys'$ **where** ‹$ys@[\pi\ i] = [x']@ys'$› **by** (*metis append-Cons append-Nil neq-Nil-conv*)
thus ‹*?thesis*› **using** *cs-split'*[*of* ‹π› ‹*i*› ‹*xs*› ‹*x*› ‹x'› ‹ys'›] *assms* **by** *auto*
qed

lemma *cs-less-split*: **assumes** ‹$xs \prec ys$› **obtains** *a as* **where** ‹$ys = xs@a\#as$›
using *assms* **unfolding** *cs-less.simps* **apply** *auto*
by (*metis Cons-nth-drop-Suc append-take-drop-id*)

lemma *cs-select-is-cs*: **assumes** ‹*is-path* π› ‹$xs \neq Nil$› ‹$xs \prec cs^{\pi}\ k$› **shows** ‹$cs^{\pi}\ (\pi¡xs) = xs$› ‹$k\ cd^{\pi}\!\rightarrow (\pi¡xs)$›**proof** −
obtain *b bs* **where** *b*: ‹$cs^{\pi}\ k = xs@b\#bs$› **using** *assms cs-less-split* **by** *blast*
obtain *a as* **where** *a*: ‹$xs = as@[a]$› **using** *assms* **by** (*metis rev-exhaust*)
have ‹$cs^{\pi}\ k = as@[a,b]@bs$› **using** *a b* **by** *auto*
then obtain k' **where** *csk*: ‹$cs^{\pi}\ k' = xs$› **and** *is-cd*: ‹$k\ cd^{\pi}\!\rightarrow k'$› **using** *cs-split' a* **by** *blast*
hence *nret*: ‹$\pi\ k' \neq return$› **by** (*metis is-cdi-def term-path-stable less-imp-le*)
show *a*: ‹$cs^{\pi}\ (\pi¡xs) = xs$› **unfolding** *cs-select-def* **using** *cs-inj*[*OF assms*(*1*) *nret*] *csk the-equality*[*of* - ‹k'›]
by (*metis* (*mono-tags*))
show ‹$k\ cd^{\pi}\!\rightarrow (\pi¡xs)$› **unfolding** *cs-select-def* **by** (*metis a assms*(*1*) *cs-inj cs-select-def csk is-cd nret*)
qed

lemma *cd-in-cs*: **assumes** ‹$n\ cd^{\pi}\!\rightarrow m$› **shows** ‹$\exists\ ns.\ cs^{\pi}\ n = (cs^{\pi}\ m)$ @ *ns* @$[\pi\ n]$›
using *assms* **proof** (*induction rule*: *cd-induct*)
case (*base n*) **thus** ‹*?case*› **by** (*metis append-Nil cs.simps icd-is-the-icd*)
next
case (*IS k n*)
hence ‹$cs^{\pi}\ n = cs^{\pi}\ k$ @ $[\pi\ n]$› **by** (*metis cs.simps icd-is-the-icd*)
thus ‹*?case*› **using** *IS* **by** *force*
qed

lemma *butlast-cs-not-cd*: **assumes** ‹*butlast* $(cs^{\pi}\ m) =$ *butlast* $(cs^{\pi}\ n)$› **shows** ‹$\neg\ m\ cd^{\pi}\!\rightarrow n$›
by (*metis append-Cons append-Nil append-assoc assms cd-in-cs cs-not-nil list.distinct*(*1*) *self-append-conv snoc-eq-iff-butlast*)

lemma *wn-cs-butlast*: **assumes** ‹*butlast* $(cs^{\pi}\ m) =$ *butlast* $(cs^{\pi}\ n)$› ‹$i\ cd^{\pi}\!\rightarrow m$› ‹$j\ cd^{\pi}\!\rightarrow n$› ‹$m<n$› **shows** ‹$i<j$›
proof (*rule ccontr*)
assume ‹$\neg\ i < j$›
moreover
have ‹$\neg\ n\ cd^{\pi}\!\rightarrow m$› **by** (*metis assms*(*1*) *butlast-cs-not-cd*)
ultimately
have ‹$n \leq m$› **using** *assms*(*2*,*3*) *cr-wn''* **by** *auto*
thus ‹*False*› **using** *assms*(*4*) **by** *auto*
qed

This is the central theorem making the control slice suitable for matching indices between executions.

theorem *cs-order*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π'› **and** *csi*: ‹$cs^{\pi}\ i = cs^{\pi'}\ i'$›
and *csj*: ‹$cs^{\pi}\ j = cs^{\pi'}\ j'$› **and** *nret*: ‹$\pi\ i \neq return$› **and** *ilj*: ‹$i < j$›
shows ‹$i'<j'$›
proof −
  have ‹$cs^{\pi}\ i \neq cs^{\pi}\ j$› **using** *cs-inj*[*OF path*(*1*) *nret*] *ilj* **by** *blast*
  moreover
  have ‹$cs^{\pi}\ i \neq Nil$› ‹$cs^{\pi}\ j \neq Nil$› **by** (*metis cs-not-nil*)+
  ultimately show ‹*?thesis*› **proof** (*cases rule*: *list-neq-prefix-cases*)
    case (*diverge xs x x′ ys ys′*)
    note *csx* = ‹$cs^{\pi}\ i = xs\ @\ [x]\ @\ ys$›
    note *csx′* = ‹$cs^{\pi}\ j = xs\ @\ [x']\ @\ ys'$›
    note *xx* = ‹$x \neq x'$›
    show ‹$i' < j'$› **proof** (*cases* ‹*ys*›)
      assume *ys*: ‹$ys = Nil$›
      show ‹*?thesis*› **proof** (*cases* ‹*ys′*›)
        assume *ys′*: ‹$ys' = Nil$›
        have *cs*: ‹$cs^{\pi}\ i = xs\ @\ [x]$› ‹$cs^{\pi}\ j = xs\ @\ [x']$› **by** (*metis append-Nil2 csx ys*, *metis append-Nil2 csx′ ys′*)
        hence *bl*: ‹$butlast\ (cs^{\pi}\ i) = butlast\ (cs^{\pi}\ j)$› **by** *auto*
        show ‹$i' < j'$› **using** *claim*[*OF path csi csj bl nret ilj*] **.**
      next
        fix *y′ zs′*
        assume *ys′*: ‹$ys' = y'\#zs'$›
        have *cs*: ‹$cs^{\pi}\ i = xs\ @\ [x]$› ‹$cs^{\pi}\ j = xs\ @\ [x',y']@\ zs'$› **by** (*metis append-Nil2 csx ys*, *metis append-Cons append-Nil csx′ ys′*)
        obtain *n* **where** *n*: ‹$cs^{\pi}\ n = xs@[x']$› **and** *jn*: ‹$j\ cd^{\pi}\!\rightarrow n$› **using** *cs cs-split′* **by** *blast*
        obtain *n′* **where** *n′*: ‹$cs^{\pi'}\ n' = xs@[x']$› **and** *jn′*: ‹$j'\ cd^{\pi'}\!\rightarrow n'$› **using** *cs cs-split′* **unfolding** *csj* **by** *blast*
        have *csn* : ‹$cs^{\pi}\ n = cs^{\pi'}\ n'$› **and** *bl*: ‹$butlast\ (cs^{\pi}\ i) = butlast\ (cs^{\pi}\ n)$› **using** *n n′ cs* **by** *auto*
        hence *bl′*: ‹$butlast\ (cs^{\pi'}\ i') = butlast\ (cs^{\pi'}\ n')$› **using** *csi* **by** *auto*
        have *notcd*: ‹$\neg\ i\ cd^{\pi}\!\rightarrow n$› **by** (*metis butlast-cs-not-cd bl*)
        have *nin*: ‹$i \neq n$› **using** *cs n xx* **by** *auto*
        have *iln*: ‹$i < n$› **apply** (*rule ccontr*) **using** *cr-wn′*[*OF jn notcd*] *nin ilj* **by** *auto*
        note *claim*[*OF path csi csn bl nret iln*]
        hence ‹$i' < n'$› **.**
        thus ‹$i' < j'$› **using** *jn′* **unfolding** *is-cdi-def* **by** *auto*
      qed
    next
      fix *y zs*
      assume *ys*: ‹$ys = y\#zs$›
      show ‹*?thesis*› **proof** (*cases* ‹*ys′*›)
        assume *ys′* : ‹$ys' = Nil$›
        have *cs*: ‹$cs^{\pi}\ i = xs\ @\ [x,y]@zs$› ‹$cs^{\pi}\ j = xs\ @\ [x']$› **by** (*metis append-Cons append-Nil csx ys*, *metis append-Nil2 csx′ ys′*)
        obtain *n* **where** *n*: ‹$cs^{\pi}\ n = xs@[x]$› **and** *jn*: ‹$i\ cd^{\pi}\!\rightarrow n$› **using** *cs cs-split′* **by** *blast*
        obtain *n′* **where** *n′*: ‹$cs^{\pi'}\ n' = xs@[x]$› **and** *jn′*: ‹$i'\ cd^{\pi'}\!\rightarrow n'$› **using** *cs cs-split′* **unfolding** *csi* **by** *blast*
        have *csn* : ‹$cs^{\pi}\ n = cs^{\pi'}\ n'$› **and** *bl*: ‹$butlast\ (cs^{\pi}\ n) = butlast\ (cs^{\pi}\ j)$› **using** *n n′ cs* **by** *auto*
        hence *bl′*: ‹$butlast\ (cs^{\pi'}\ j') = butlast\ (cs^{\pi'}\ n')$› **using** *csj* **by** *auto*
        have *notcd*: ‹$\neg\ j'\ cd^{\pi'}\!\rightarrow n'$› **by** (*metis butlast-cs-not-cd bl′*)
        have *nin*: ‹$n < i$› **using** *jn* **unfolding** *is-cdi-def* **by** *auto*
        have *nlj*: ‹$n < j$› **using** *nin ilj* **by** *auto*
        note *claim*[*OF path csn csj bl - nlj*]
        hence *nj′*: ‹$n' < j'$› **using** *term-path-stable*[*OF path*(*1*) -] *less-imp-le nin nret* **by** *auto*
        show ‹$i' < j'$› **apply**(*rule ccontr*) **using** *cdi-prefix*[*OF jn′ nj′*] *notcd* **by** *auto*
      next

fix *y′ zs′*
assume *ys′* : ‹$ys' = y'\#zs'$›
have *cs*: ‹$cs^{\pi}\ i = xs@[x,y]@zs$› ‹$cs^{\pi}\ j = xs@[x',y']@zs'$› **by** (*metis append-Cons append-Nil csx ys,metis append-Cons append-Nil csx′ ys′*)
have *neq*: ‹$cs^{\pi}\ i \neq cs^{\pi}\ j$› **using** *cs-inj path nret ilj* **by** *blast*
obtain *m* **where** *m*: ‹$cs^{\pi}\ m = xs@[x]$› **and** *im*: ‹$i\ cd^{\pi}\!\rightarrow m$› **using** *cs cs-split′* **by** *blast*
obtain *n* **where** *n*: ‹$cs^{\pi}\ n = xs@[x']$› **and** *jn*: ‹$j\ cd^{\pi}\!\rightarrow n$› **using** *cs cs-split′* **by** *blast*
obtain *m′* **where** *m′*: ‹$cs^{\pi'}\ m' = xs@[x]$› **and** *im′*: ‹$i'\ cd^{\pi'}\!\rightarrow m'$› **using** *cs cs-split′* **unfolding** *csi* **by** *blast*
obtain *n′* **where** *n′*: ‹$cs^{\pi'}\ n' = xs@[x']$› **and** *jn′*: ‹$j'\ cd^{\pi'}\!\rightarrow n'$› **using** *cs cs-split′* **unfolding** *csj* **by** *blast*
have ‹$m \leq n$› **using** *ilj m n wn-cs-butlast*[*OF - jn im*] **by** *force*
moreover
have ‹$m \neq n$› **using** *m n xx* **by** (*metis last-snoc*)
ultimately
have *mn*: ‹$m < n$› **by** *auto*
moreover
have ‹$\pi\ m \neq return$› **by** (*metis last-cs last-snoc m mn n path*(*1*) *term-path-stable xx less-imp-le*)
moreover
have ‹$butlast\ (cs^{\pi}\ m) = butlast\ (cs^{\pi}\ n)$› ‹$cs^{\pi}\ m = cs^{\pi'}\ m'$› ‹$cs^{\pi}\ n = cs^{\pi'}\ n'$› **using** *m n n′ m′* **by** *auto*
ultimately
have ‹$m' < n'$› **using** *claim path* **by** *blast*
thus ‹$i' < j'$› **using** *m′ n′ im′ jn′ wn-cs-butlast* **by** (*metis butlast-snoc*)
qed
qed
next
case (*prefix1 xs*)
note *pfx* = ‹$cs^{\pi}\ i = cs^{\pi}\ j\ @\ xs$›
note *xs* = ‹$xs \neq []$›
obtain *a as* **where** ‹$xs = a\#as$› **using** *xs* **by** (*metis list.exhaust*)
moreover
obtain *bs b* **where** *bj*: ‹$cs^{\pi}\ j = bs@[b]$› **using** *cs-not-nil* **by** (*metis rev-exhaust*)
ultimately
have ‹$cs^{\pi}\ i = bs@[b,a]@as$› **using** *pfx* **by** *auto*
then obtain *m* **where** ‹$cs^{\pi}\ m = bs@[b]$› **and** *cdep*: ‹$i\ cd^{\pi}\!\rightarrow m$› **using** *cs-split′* **by** *blast*
hence *mi*: ‹$m = j$› **using** *bj cs-inj* **by** (*metis is-cdi-def term-path-stable less-imp-le*)
hence ‹$i\ cd^{\pi}\!\rightarrow j$› **using** *cdep* **by** *auto*
hence ‹*False*› **using** *ilj* **unfolding** *is-cdi-def* **by** *auto*
thus ‹$i' < j'$› **..**
next
case (*prefix2 xs*)
have *pfx* : ‹$cs^{\pi'}\ i'\ @\ xs = cs^{\pi'}\ j'$› **using** *prefix2 csi csj* **by** *auto*
note *xs* = ‹$xs \neq []$›
obtain *a as* **where** ‹$xs = a\#as$› **using** *xs* **by** (*metis list.exhaust*)
moreover
obtain *bs b* **where** *bj*: ‹$cs^{\pi'}\ i' = bs@[b]$› **using** *cs-not-nil* **by** (*metis rev-exhaust*)
ultimately
have ‹$cs^{\pi'}\ j' = bs@[b,a]@as$› **using** *pfx* **by** *auto*
then obtain *m* **where** ‹$cs^{\pi'}\ m = bs@[b]$› **and** *cdep*: ‹$j'\ cd^{\pi'}\!\rightarrow m$› **using** *cs-split′* **by** *blast*
hence *mi*: ‹$m = i'$› **using** *bj cs-inj* **by** (*metis is-cdi-def term-path-stable less-imp-le*)
hence ‹$j'\ cd^{\pi'}\!\rightarrow i'$› **using** *cdep* **by** *auto*
thus ‹$i' < j'$› **unfolding** *is-cdi-def* **by** *auto*
qed
qed

lemma *cs-order-le*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π'› **and** *csi*: ‹cs^{π} i = $cs^{\pi'}$ i'›
and *csj*: ‹cs^{π} j = $cs^{\pi'}$ j'› **and** *nret*: ‹π $i \neq$ *return*› **and** *ilj*: ‹$i \leq j$›
shows ‹$i' \leq j'$› **proof** *cases*
  assume ‹$i < j$› **with** *cs-order*[*OF assms*(*1*,*2*,*3*,*4*,*5*)] **show** ‹*?thesis*› **by** *simp*
next
  assume ‹$\neg$ $i < j$›
  hence ‹$i = j$› **using** *ilj* **by** *simp*
  hence *csij*: ‹$cs^{\pi'}$ i' = $cs^{\pi'}$ j'› **using** *csi csj* **by** *simp*
  have *nret'*: ‹π' $i' \neq$ *return*› **using** *nret last-cs csi* **by** *metis*
  show ‹*?thesis*› **using** *cs-inj*[*OF path*(*2*) *nret' csij*] **by** *simp*
qed

lemmas *cs-induct*[*case-names cs*] = *cs.induct*

lemma *icdi-path-swap*: **assumes** ‹*is-path* π'› ‹j $icd^{\pi}\rightarrow k$› ‹$\pi =_j \pi'$› **shows** ‹j $icd^{\pi'}\rightarrow k$› **using** *assms* **unfolding** *eq-up-to-def is-icdi-def is-cdi-def* **by** *auto*

lemma *icdi-path-swap-le*: **assumes** ‹*is-path* π'› ‹j $icd^{\pi}\rightarrow k$› ‹$\pi =_n \pi'$› ‹$j \leq n$› **shows** ‹j $icd^{\pi'}\rightarrow k$› **by** (*metis assms icdi-path-swap eq-up-to-le*)

lemma *cs-path-swap*: **assumes** ‹*is-path* π› ‹*is-path* π'› ‹$\pi =_k \pi'$› **shows** ‹cs^{π} k = $cs^{\pi'}$ k› **using** *assms*(*1*,*3*)
proof (*induction* ‹π› ‹k› *rule*:*cs-induct*,*cases*)
  case (*cs* π k)
  let ‹*?l*› = ‹(*THE* l. k $icd^{\pi}\rightarrow$ l)›
  assume $*$: ‹$\exists\, l$. k $icd^{\pi}\rightarrow$ l›
  have *kicd*: ‹k $icd^{\pi}\rightarrow$ *?l*› **by** (*metis* $*$ *icd-is-the-icd*)
  hence ‹*?l* $<$ k› **unfolding** *is-cdi-def*[*of* ‹k› ‹π› ‹*?l*›] *is-icdi-def*[*of* ‹k› ‹π› ‹*?l*›] **by** *auto*
  hence ‹$\forall$ $i \leq$*?l*. π i = π' i› **using** *cs*(*2*,*3*) **unfolding** *eq-up-to-def* **by** *auto*
  hence *csl*: ‹cs^{π} *?l* = $cs^{\pi'}$ *?l*› **using** *cs*(*1*,*2*) $*$ **unfolding** *eq-up-to-def* **by** *auto*
  have *kicd*: ‹k $icd^{\pi}\rightarrow$ *?l*› **by** (*metis* $*$ *icd-is-the-icd*)
  hence *csk*: ‹cs^{π} k = cs^{π} *?l* @ [π k]› **using** *kicd* **by** *auto*
  have *kicd'*: ‹k $icd^{\pi'}\rightarrow$ *?l*› **using** *kicd icdi-path-swap*[*OF assms*(*2*) - *cs*(*3*)] **by** *simp*
  hence ‹*?l* = (*THE* l. k $icd^{\pi'}\rightarrow$ l)› **by** (*metis icd-is-the-icd*)
  hence *csk'*: ‹$cs^{\pi'}$ k = $cs^{\pi'}$ *?l* @ [π' k]› **using** *kicd'* **by** *auto*
  have ‹π' k = π k› **using** *cs*(*3*) **unfolding** *eq-up-to-def* **by** *auto*
  with *csl csk csk'*
  show ‹*?case*› **by** *auto*
next
  case (*cs* π k)
  assume $*$: ‹$\neg$ ($\exists\, l$. k $icd^{\pi}\rightarrow$ l)›
  hence *csk*: ‹cs^{π} k = [π k]› **by** *auto*
  have ‹$\neg$ ($\exists\, l$. k $icd^{\pi'}\rightarrow$ l)› **apply** (*rule ccontr*) **using** $*$ *icdi-path-swap-le*[*OF cs*(*2*) -, *of* ‹k› ‹π'›] *cs*(*3*) **by** (*metis eq-up-to-sym le-refl*)
  hence *csk'*: ‹$cs^{\pi'}$ k = [π' k]› **by** *auto*
  with *csk* **show** ‹*?case*› **using** *cs*(*3*) *eq-up-to-apply* **by** *auto*
qed

lemma *cs-path-swap-le*: **assumes** ‹*is-path* π› ‹*is-path* π'› ‹$\pi =_n \pi'$› ‹$k \leq n$› **shows** ‹cs^{π} k = $cs^{\pi'}$ k› **by** (*metis assms cs-path-swap eq-up-to-le*)

lemma *cs-path-swap-cd*: **assumes** ‹*is-path* π› **and** ‹*is-path* π'› **and** ‹cs^{π} n = $cs^{\pi'}$ n'› **and** ‹n $cd^{\pi}\rightarrow$ k›
obtains k' **where** ‹n' $cd^{\pi'}\rightarrow$ k'› **and** ‹cs^{π} k = $cs^{\pi'}$ k'›
proof –

from *cd-in-cs*[*OF assms*(*4*)]
obtain *ns* **where** ∗: ‹cs^{π} *n* = cs^{π} *k* @ *ns* @ [π *n*]› **by** *blast*
obtain *xs x* **where** *csk*: ‹cs^{π} *k* = *xs* @ [*x*]› **by** (*metis cs-not-nil rev-exhaust*)
have ‹π *n* = π′ *n*′› **using** *assms*(*3*) *last-cs* **by** *metis*
hence ∗∗: ‹$cs^{\pi'}$ *n*′ = *xs*@[*x*]@*ns*@[π′ *n*′]› **using** ∗ *assms*(*3*) *csk* **by** *auto*
from *cs-split*[*OF* ∗∗]
obtain *k*′ **where** ‹$cs^{\pi'}$ *k*′ = *xs* @ [*x*]› ‹*n*′ $cd^{\pi'}\!\!\rightarrow$ *k*′› **by** *blast*
thus ‹*thesis*› **using** *that csk* **by** *auto*
qed

lemma *path-ipd-swap*: **assumes** ‹*is-path* π› ‹π *k* ≠ *return*› ‹*k* < *n*›
obtains π′ *m* **where** ‹*is-path* π′› ‹π $=_n$ π′› ‹*k* < *m*› ‹π′ *m* = *ipd* (π′ *k*)› ‹∀ *l* ∈ {*k*..<*m*}. π′ *l* ≠ *ipd* (π′ *k*)›
proof −
obtain π′ *r* **where** ∗: ‹π′ *0* = π *n*› ‹*is-path* π′› ‹π′ *r* = *return*› **by** (*metis assms*(*1*) *path-nodes reaching-ret*)
let ‹*?*π› = ‹π$@^n$ π′›
have *path*: ‹*is-path ?*π› **and** *ret*: ‹*?*π (*n* + *r*) = *return*› **and** *equpto*: ‹*?*π $=_n$ π› **using** *assms path-cons* ∗ *path-append-eq-up-to* **by** *auto*
have π*k*: ‹*?*π *k* = π *k*› **by** (*metis assms*(*3*) *less-imp-le-nat path-append-def*)
obtain *j* **where** *j*: ‹*k* < *j* ∧ *j* ≤ (*n* + *r*) ∧ *?*π *j* = *ipd* (π *k*)› (**is** ‹*?P j*›)**by** (*metis* π*k assms*(*2*) *path path-ret-ipd ret*)
define *m* **where** *m*: ‹*m* ≡ *LEAST m* . *?P m*›
have *Pm*: ‹*?P m*› **using** *LeastI*[*of* ‹*?P*› ‹*j*›] *j m* **by** *auto*
hence *km*: ‹*k* < *m*› ‹*m* ≤ (*n* + *r*)› ‹*?*π *m* = *ipd* (π *k*)› **by** *auto*
have *le*: ‹⋀*l*. *?P l* ⟹ *m* ≤ *l*› **using** *Least-le*[*of* ‹*?P*›] *m* **by** *blast*
have π*knipd*: ‹*?*π *k* ≠ *ipd* (π *k*)› **by** (*metis* π*k assms*(*1*) *assms*(*2*) *ipd-not-self path-nodes*)
have *nipd*′: ‹⋀*l*. *k* < *l* ⟹ *l* < *m* ⟹ *?*π *l* ≠ *ipd* (π *k*)› **apply** (*rule ccontr*) **using** *le km*(*2*) **by** *force*
have ‹∀ *l* ∈ {*k*..<*m*}. *?*π *l* ≠ *ipd*(π *k*)› **using** π*knipd nipd*′ **by**(*auto*, *metis le-eq-less-or-eq*,*metis le-eq-less-or-eq*)
thus ‹*thesis*› **using** *that* **by** (*metis* π*k eq-up-to-sym km*(*1*) *km*(*3*) *path path-append-eq-up-to*)
qed

lemma *cs-sorted-list-of-cd*′: ‹cs^{π} *k* = *map* π (*sorted-list-of-set* { *i* . *k* $cd^{\pi}\!\!\rightarrow$ *i*}) @ [π *k*]›
proof (*induction* ‹π› ‹*k*› *rule*: *cs.induct*, *cases*)
case (*1* π *k*)
assume ‹∃ *j*. *k* $icd^{\pi}\!\!\rightarrow$ *j*›
then obtain *j* **where** *j*: *k* $icd^{\pi}\!\!\rightarrow$ *j* **..**
hence *csj*: ‹cs^{π} *j* = *map* π (*sorted-list-of-set* {*i*. *j* $cd^{\pi}\!\!\rightarrow$ *i*}) @ [π *j*]› **by** (*metis 1.IH icd-is-the-icd*)
have ‹{*i*. *k* $cd^{\pi}\!\!\rightarrow$ *i*} = *insert j* {*i*. *j* $cd^{\pi}\!\!\rightarrow$ *i*}› **using** *cdi-is-cd-icdi*[*OF j*] **by** *auto*
moreover
have *f*: ‹*finite* {*i*. *j* $cd^{\pi}\!\!\rightarrow$ *i*}› **unfolding** *is-cdi-def* **by** *auto*
moreover
have ‹*j* ∉ {*i*. *j* $cd^{\pi}\!\!\rightarrow$ *i*}› **unfolding** *is-cdi-def* **by** *auto*
ultimately
have ‹*sorted-list-of-set* { *i* . *k* $cd^{\pi}\!\!\rightarrow$ *i*} = *insort j* (*sorted-list-of-set* { *i* . *j* $cd^{\pi}\!\!\rightarrow$ *i*})› **using** *sorted-list-of-set-insert* **by** *auto*
moreover
have ‹∀ *x* ∈ {*i*. *j* $cd^{\pi}\!\!\rightarrow$ *i*}. *x* < *j*› **unfolding** *is-cdi-def* **by** *auto*
hence ‹∀ *x* ∈ *set* (*sorted-list-of-set* {*i*. *j* $cd^{\pi}\!\!\rightarrow$ *i*}). *x* < *j*› **by** (*simp add*: *f*)
ultimately
have ‹*sorted-list-of-set* { *i* . *k* $cd^{\pi}\!\!\rightarrow$ *i*} = (*sorted-list-of-set* { *i* . *j* $cd^{\pi}\!\!\rightarrow$ *i*})@[*j*]› **using** *insort-greater* **by** *auto*
hence ‹cs^{π} *j* = *map* π (*sorted-list-of-set* { *i* . *k* $cd^{\pi}\!\!\rightarrow$ *i*})› **using** *csj* **by** *auto*
thus ‹*?case*› **by** (*metis icd-cs j*)
next
case (*1* π *k*)
assume ∗: ‹¬ (∃ *j*. *k* $icd^{\pi}\!\!\rightarrow$ *j*)›

hence ‹$cs^{\pi}\ k = [\pi\ k]$› **by** (*metis cs-cases*)
moreover
have ‹$\{\ i\ .\ k\ cd^{\pi}\!\to i\} = \{\}$› **by** (*auto, metis* $*$ *excd-impl-exicd*)
ultimately
show ‹*?case*› **by** (*metis append-Nil list.simps*(*8*) *sorted-list-of-set-empty*)
qed

lemma *cs-sorted-list-of-cd*: ‹$cs^{\pi}\ k = map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ (\{\ i\ .\ k\ cd^{\pi}\!\to i\} \cup \{k\}))$› **proof** −
have *le*: ‹$\forall\ x \in \{i.\ k\ cd^{\pi}\!\to i\}.\forall\ y \in \{k\}.\ x < y$› **unfolding** *is-cdi-def* **by** *auto*
have *fin*: ‹*finite* $\{i.\ k\ cd^{\pi}\!\to i\}$› ‹*finite* $\{k\}$› **unfolding** *is-cdi-def* **by** *auto*
show ‹*?thesis*› **unfolding** *cs-sorted-list-of-cd*$'$[*of* ‹π› ‹k›] *sorted-list-of-set-append*[*OF fin le*] **by** *auto*
qed

lemma *cs-not-ipd*: **assumes** ‹$k\ cd^{\pi}\!\to j \wedge ipd\ (\pi\ j) \neq ipd\ (\pi\ k)$› (**is** ‹*?Q j*›)
shows ‹$cs^{\pi}\ (GREATEST\ j.\ k\ cd^{\pi}\!\to j \wedge ipd\ (\pi\ j) \neq ipd\ (\pi\ k)) = [n{\leftarrow}cs^{\pi}\ k\ .\ ipd\ n \neq ipd\ (\pi\ k)]$›
(**is** ‹cs^{π} *?j* = *filter ?P -*›)
proof −
have *csk*: ‹$cs^{\pi}\ k = map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ (\{\ i\ .\ k\ cd^{\pi}\!\to i\ \} \cup \{k\}))$› **by** (*metis cs-sorted-list-of-cd*)
have *csj*: ‹cs^{π} *?j* $= map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ (\{i.\ ?j\ cd^{\pi}\!\to i\ \} \cup \{?j\}))$› **by** (*metis cs-sorted-list-of-cd*)

have *bound*: ‹$\forall\ j.\ k\ cd^{\pi}\!\to j \wedge ipd\ (\pi\ j) \neq ipd(\pi\ k) \longrightarrow j \leq k$› **unfolding** *is-cdi-def* **by** *simp*

have *kcdj*: ‹$k\ cd^{\pi}\!\to\ ?j$› **and** *ipd*$'$: ‹$ipd\ (\pi\ ?j) \neq ipd(\pi\ k)$› **using** *GreatestI-nat*[*of* ‹*?Q*› ‹j› ‹k›, *OF assms*] *bound* **by** *auto*

have *greatest*: ‹$\bigwedge\ j.\ k\ cd^{\pi}\!\to j \Longrightarrow ipd\ (\pi\ j) \neq ipd\ (\pi\ k) \Longrightarrow j \leq\ ?j$› **using** *Greatest-le-nat*[*of* ‹*?Q*› - ‹k›] *bound* **by** *auto*
have *less-not-ipdk*: ‹$\bigwedge\ j.\ k\ cd^{\pi}\!\to j \Longrightarrow j <\ ?j \Longrightarrow ipd\ (\pi\ j) \neq ipd\ (\pi\ k)$› **by** (*metis* (*lifting*) *ipd*$'$ *kcdj same-ipd-stable*)
hence *le-not-ipdk*: ‹$\bigwedge\ j.\ k\ cd^{\pi}\!\to j \Longrightarrow j \leq\ ?j \Longrightarrow ipd\ (\pi\ j) \neq ipd\ (\pi\ k)$› **using** *kcdj ipd*$'$ **by** (*case-tac* ‹$j =\ ?j$›,*auto*)
have $*$: ‹$\{j \in \{i.\ k\ cd^{\pi}\!\to i\} \cup \{k\}.\ ?P\ (\pi\ j)\} = insert\ ?j\ \{\ i\ .\ ?j\ cd^{\pi}\!\to i\}$ ›
apply *auto*
apply (*metis* (*lifting, no-types*) *greatest cr-wn*$''$ *kcdj le-antisym le-refl*)
apply (*metis kcdj*)
apply (*metis ipd*$'$)
apply (*metis* (*full-types*) *cd-trans kcdj*)
apply (*subgoal-tac* ‹$k\ cd^{\pi}\!\to x$›)
apply (*metis* (*lifting, no-types*) *is-cdi-def less-not-ipdk*)
by (*metis* (*full-types*) *cd-trans kcdj*)
have ‹*finite* $(\{i\ .\ k\ cd^{\pi}\!\to i\} \cup \{k\})$› **unfolding** *is-cdi-def* **by** *auto*
note *filter-sorted-list-of-set*[*OF this*, *of* ‹*?P o* π›]
hence ‹$[n{\leftarrow}cs^{\pi}\ k\ .\ ipd\ n \neq ipd(\pi\ k)] = map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ \{j \in \{i.\ k\ cd^{\pi}\!\to i\} \cup \{k\}.\ ?P\ (\pi\ j)\})$› **unfolding** *csk filter-map* **by** *auto*
also
have ‹$\ldots =\ map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ (insert\ ?j\ \{\ i\ .\ ?j\ cd^{\pi}\!\to i\}))$› **unfolding** $*$ **by** *auto*
also
have ‹$\ldots = cs^{\pi}\ ?j$› **using** *csj* **by** *auto*
finally
show ‹*?thesis*› **by** *metis*
qed

lemma *cs-ipd*: **assumes** *ipd*: ‹$\pi\ m = ipd\ (\pi\ k)$› ‹$\forall\ n \in \{k..{<}m\}.\ \pi\ n \neq ipd\ (\pi\ k)$›
and *km*: ‹$k < m$› **shows** ‹$cs^{\pi}\ m = [n{\leftarrow}cs^{\pi}\ k\ .\ ipd\ n \neq \pi\ m]\ @\ [\pi\ m]$›
proof *cases*
assume ‹$\exists\ j.\ m\ icd^{\pi}\!\to j$›

then obtain j **where** *jicd*: $\langle m\ icd^{\pi}\!\rightarrow j\rangle$ **by** *blast*
hence $*$: $\langle cs^{\pi}\ m = cs^{\pi}\ j\ @\ [\pi\ m]\rangle$ **by** (*metis icd-cs*)
have j: $\langle j = (GREATEST\ j.\ k\ cd^{\pi}\!\rightarrow j \wedge ipd\ (\pi\ j) \neq \pi\ m)\rangle$ **using** *jicd assms ipd-icd-greatest-cd-not-ipd* **by** *blast*
moreover
have $\langle ipd\ (\pi\ j) \neq ipd\ (\pi\ k)\rangle$ **by** (*metis is-cdi-def is-icdi-def is-ipd-def cd-not-pd ipd*(*1*) *ipd-is-ipd jicd path-nodes less-imp-le term-path-stable*)
moreover
have $\langle k\ cd^{\pi}\!\rightarrow j\rangle$ **unfolding** j **by** (*metis* (*lifting, no-types*) *assms*(*3*) *cd-ipd-is-cd icd-imp-cd ipd*(*1*) *ipd*(*2*) *j jicd*)
ultimately
have $\langle cs^{\pi}\ j = [n\leftarrow cs^{\pi}\ k\ .\ ipd\ n \neq \pi\ m]\rangle$ **using** *cs-not-ipd*[*of* ‹*k*› ‹*π*› ‹*j*›] *ipd*(*1*) **by** *metis*
thus ‹*?thesis*› **using** $*$ **by** *metis*
next
assume *noicd*: $\langle\neg\ (\exists\ j.\ m\ icd^{\pi}\!\rightarrow j)\rangle$
hence *csm*: $\langle cs^{\pi}\ m = [\pi\ m]\rangle$ **by** *auto*
have $\langle\bigwedge j.\ k\ cd^{\pi}\!\rightarrow j \Longrightarrow ipd(\pi\ j) = \pi\ m\rangle$ **using** *cd-is-cd-ipd*[*OF km ipd*] **by** (*metis excd-impl-exicd noicd*)
hence $*$: $\langle\{j \in \{i.\ k\ cd^{\pi}\!\rightarrow i\} \cup \{k\}.\ ipd\ (\pi\ j) \neq \pi\ m\} = \{\}\rangle$ **using** *ipd*(*1*) **by** *auto*
have $**$: $\langle((\lambda n.\ ipd\ n \neq \pi\ m)\ o\ \pi) = (\lambda n.\ ipd\ (\pi\ n) \neq \pi\ m)\rangle$ **by** *auto*
have *fin*: $\langle finite\ (\{i.\ k\ cd^{\pi}\!\rightarrow i\} \cup \{k\})\rangle$ **unfolding** *is-cdi-def* **by** *auto*
note *csk* = *cs-sorted-list-of-cd*[*of* ‹*π*› ‹*k*›]
hence $\langle[n\leftarrow cs^{\pi}\ k\ .\ ipd\ n \neq \pi\ m] = [n\leftarrow (map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ (\{i.\ k\ cd^{\pi}\!\rightarrow i\} \cup \{k\})))\ .\ ipd\ n \neq \pi\ m]\rangle$ **by** *simp*
also
have $\langle\ldots = map\ \pi\ [n <\!-\ sorted\text{-}list\text{-}of\text{-}set\ (\{i.\ k\ cd^{\pi}\!\rightarrow i\} \cup \{k\}).\ ipd\ (\pi\ n) \neq \pi\ m]\rangle$ **by** (*auto simp add*: *filter-map* $**$)
also
have $\langle\ldots = []\rangle$ **unfolding** $*$ *filter-sorted-list-of-set*[*OF fin, of* $\langle\lambda n.\ ipd\ (\pi\ n) \neq \pi\ m\rangle$] **by** *auto*
finally
show ‹*?thesis*› **using** *csm* **by** (*metis append-Nil*)
qed

lemma *converged-ipd-same-icd*: **assumes** *path*: ‹*is-path π*› ‹*is-path π′*› **and** *converge*: $\langle l < m\rangle$ $\langle cs^{\pi}\ m = cs^{\pi'}\ m'\rangle$
and *csk*: $\langle cs^{\pi}\ k = cs^{\pi'}\ k'\rangle$ **and** *icd*: $\langle l\ icd^{\pi}\!\rightarrow k\rangle$ **and** *suc*: $\langle\pi\ (Suc\ k) = \pi'\ (Suc\ k')\rangle$
and *ipd*: $\langle\pi'\ m' = ipd\ (\pi\ k)\rangle$ $\langle\forall\ n \in \{k'..<m'\}.\ \pi'\ n \neq ipd\ (\pi\ k)\rangle$
shows $\langle\exists\, l'.\ cs^{\pi}\ l = cs^{\pi'}\ l'\rangle$
proof *cases*
assume l: $\langle l = Suc\ k\rangle$
hence $\langle Suc\ k\ cd^{\pi}\!\rightarrow k\rangle$ **using** *icd* **by** (*metis is-icdi-def*)
hence $\langle\pi\ (Suc\ k) \neq ipd\ (\pi\ k)\rangle$ **unfolding** *is-cdi-def* **by** *auto*
hence $\langle\pi'\ (Suc\ k') \neq ipd\ (\pi'\ k')\rangle$ **by** (*metis csk last-cs suc*)
moreover
have $\langle\pi'\ (Suc\ k') \neq return\rangle$ **by** (*metis* $\langle Suc\ k\ cd^{\pi}\!\rightarrow k\rangle$ *ret-no-cd suc*)
ultimately
have $\langle Suc\ k'\ cd^{\pi'}\!\rightarrow k'\rangle$ **unfolding** *is-cdi-def* **using** *path*(*2*) **apply** *auto*
by (*metis ipd-not-self le-Suc-eq le-antisym path-nodes term-path-stable*)
hence $\langle Suc\ k'\ icd^{\pi'}\!\rightarrow k'\rangle$ **unfolding** *is-icdi-def* **using** *path*(*2*) **by** *fastforce*
hence $\langle cs^{\pi'}\ (Suc\ k') = cs^{\pi'}\ k'\ @[\pi'\ (Suc\ k')]\rangle$ **using** *icd-cs* **by** *auto*
moreover
have $\langle cs^{\pi}\ l = cs^{\pi}\ k\ @\ [\pi\ l]\rangle$ **using** *icd icd-cs* **by** *auto*
ultimately
have $\langle cs^{\pi}\ l = cs^{\pi'}\ (Suc\ k')\rangle$ **by** (*metis csk l suc*)
thus ‹*?thesis*› **by** *blast*
next

assume *nsuck*: ‹$l \neq Suc\ k$›
have *kk'[simp]*: ‹$\pi'\ k' = \pi\ k$› **by** (*metis csk last-cs*)
have *kl*: ‹$k < l$› **using** *icd* **unfolding** *is-icdi-def is-cdi-def* **by** *auto*
hence *skl*: ‹$Suc\ k < l$› **by** (*metis Suc-lessI nsuck*)
hence *lpd*: ‹$\pi\ l\ pd\rightarrow \pi\ (Suc\ k)$› **using** *icd icd-pd-intermediate* **by** *auto*
have *km*: ‹$k < m$› **by** (*metis converge(1) kl order.strict-trans*)
have *lcd*: ‹$l\ cd^{\pi}\rightarrow k$› **using** *icd is-icdi-def* **by** *auto*
hence *ipdk-pdl*: ‹$ipd\ (\pi\ k)\ pd\rightarrow (\pi\ l)$› **by** (*metis ipd-pd-cd*)
have $*$: ‹$ipd\ (\pi\ k) \in nodes$› **by** (*metis ipdk-pdl pd-node1*)
have *nretk*: ‹$\pi\ k \neq return$› **by** (*metis kl lcd path(1) ret-no-cd term-path-stable less-imp-le*)
have $**$: ‹$\neg\ (\pi\ l)\ pd\rightarrow ipd\ (\pi\ k)$› **proof**
  assume *a*: ‹$\pi\ l\ pd\rightarrow ipd\ (\pi\ k)$›
   hence ‹$\pi\ l\ pd\rightarrow (\pi\ k)$› **by** (*metis is-ipd-def* ‹$k < l$› *ipd-is-ipd ipdk-pdl path(1) path-nodes pd-antisym term-path-stable less-imp-le*)
  moreover
  have ‹$\pi\ l \neq (\pi\ k)$› **by** (*metis $*$ a ipd-not-self ipdk-pdl lcd pd-antisym ret-no-cd*)
  ultimately
  show ‹*False*› **using** *lcd cd-not-pd* **by** *auto*
qed

have *km'*: ‹$k' < m'$› **using** *cs-order[OF path csk converge(2) nretk km]* **.**

obtain π'' n'' **where** *path''*: ‹*is-path* π''› **and** *π''0*: ‹$\pi''\ 0 = ipd\ (\pi\ k)$› **and** *π''n*: ‹$\pi''\ n'' = return$› **and** *notπl*: ‹$\forall\ i{\leq}n''.\ \pi''\ i \neq \pi\ l$› **using** *no-pd-path[OF $*$ $**$]* **.**
let ‹$?\pi'$› = ‹$(\pi'\ @^{m'}\ \pi'')$ « $Suc\ k'$›
have ‹*is-path* $?\pi'$› **by** (*metis π''0 ipd(1) path'' path(2) path-cons path-path-shift*)
moreover
have ‹$?\pi'\ 0 = \pi\ (Suc\ k)$› **using** *km' suc* **by** *auto*
moreover
have ‹$?\pi'\ (m' - Suc\ k' + n'') = return$› **using** *$\pi''$n km' π''0 ipd(1)* **by** *auto*
ultimately
obtain l'' **where** *l''*: ‹$l'' \leq m' - Suc\ k' + n''$› ‹$?\pi'\ l'' = \pi\ l$› **using** *lpd* **unfolding** *is-pd-def* **by** *blast*
have *l''m*: ‹$l'' \leq m' - Suc\ k'$› **apply** (*rule ccontr*) **using** *l'' notπl km'* **by** (*cases* ‹$Suc\ (k' + l'') \leq m'$›, *auto*)
let ‹$?l'$› = ‹$Suc\ (\ k' + l'')$›
have *lm'*: ‹$?l' \leq m'$› **using** *l''m km'* **by** *auto*

— Now we have found our desired l'
have *1*: ‹$\pi'\ ?l' = \pi\ l$› **using** *l'' l''m lm'* **by** *auto*
have *2*: ‹$k' < ?l'$› **by** *simp*
have *3*: ‹$?l' < m'$› **apply** (*rule ccontr*) **using** *lm'* **by** (*simp, metis $**$ 1 ipd(1) ipdk-pdl*)

— Need the least such l'

let ‹$?P$› = ‹$\lambda\ l'.\ \pi'\ l' = \pi\ l \wedge k' < l' \wedge l' < m'$›

have $*$: ‹$?P\ ?l'$› **using** *1 2 3* **by** *blast*

define l' **where** *l'*: ‹$l' == LEAST\ l'.\ ?P\ l'$›

have *πl'*: ‹$\pi'\ l' = \pi\ l$› **using** *l' 1 2 3 LeastI[of* ‹$?P$›*]* **by** *blast*
have *kl'*: ‹$k' < l'$› **using** *l' 1 2 3 LeastI[of* ‹$?P$›*]* **by** *blast*
have *lm'*: ‹$l' < m'$› **using** *l' 1 2 3 LeastI[of* ‹$?P$›*]* **by** *blast*

have *nretl'*: ‹$\pi'\ l' \neq return$› **by** (*metis π''n πl' le-refl notπl*)

have *nipd'*: ‹$\forall\ j \in \{k'..l'\}.\ \pi'\ j \neq ipd\ (\pi'\ k')$› **using** *lm' kk' ipd(2) kl'* **by** *force*

have *lcd'*: ‹l' $cd^{\pi'}\rightarrow$ k'› **by** (*metis is-cdi-def kl' nipd' nretl' path*(*2*))

have *licd*: ‹l' $icd^{\pi'}\rightarrow$ k'› **proof** −
  have ‹$\forall$ $m \in \{k'<..<l'\}$. ¬ l' $cd^{\pi'}\rightarrow$ m› **proof** (*rule ccontr*)
    assume ‹¬ ($\forall$ $m \in \{k'<..<l'\}$. ¬ l' $cd^{\pi'}\rightarrow$ m)›
    then obtain j' **where** *kj'*: ‹$k' < j'$› **and** *jl'*: ‹$j' < l'$› **and** *lcdj'*: ‹l' $cd^{\pi'}\rightarrow$ j'› **by** *force*
    have *jm'*: ‹$j'<m'$› **by** (*metis jl' lm' order.strict-trans*)
    have ‹π' $j' \neq \pi$ l› **apply** (*rule ccontr*) **using** *l' kj' jm' jl' Least-le*[*of* ‹*?P*› ‹j'›] **by** *auto*
    hence ‹¬ π' l' $pd\rightarrow$ π' j'› **using** *cd-not-pd lcdj' πl'* **by** *metis*
    moreover have ‹π' $j' \in$ *nodes*› **using** *path*(*2*) *path-nodes* **by** *auto*
    ultimately
    obtain π_1 n_1 **where** *path*$_1$: ‹*is-path* π_1› **and** $\pi 0_1$: ‹π_1 *0* $= \pi'$ j'› **and** πn_1: ‹π_1 n_1 $=$ *return*› **and** *nl'*: ‹$\forall$ $l \leq n_1$. π_1 $l \neq \pi'$ l'› **unfolding** *is-pd-def* **by** *blast*
    let ‹*?π''*› = ‹($\pi'@^{j'}$ π_1) « *Suc k'*›
    have ‹*is-path ?π''*› **by** (*metis* $\pi 0_1$ *path*(*2*) *path*$_1$ *path-cons path-path-shift*)
    moreover
    have ‹*?π''* *0* $= \pi$ (*Suc k*)› **by** (*simp*, *metis kj' less-eq-Suc-le suc*)
    moreover
    have *kj'*: ‹*Suc* $k' \leq j'$› **by** (*metis kj' less-eq-Suc-le*)
    hence ‹*?π''* (j' − *Suc* k' + n_1) = *return*› **by** (*simp*, *metis* $\pi 0_1$ πn_1)
    ultimately
    obtain l'' **where** ∗: ‹*?π''* $l'' = \pi$ l› **and** ∗∗: ‹$l'' \leq j'$ − *Suc* k' + n_1› **using** *lpd is-pd-def* **by** *blast*
    show ‹*False*› **proof** (*cases*)
      assume *a*: ‹$l'' \leq j'$ − *Suc* k'›
      hence ‹π' (l'' + *Suc* k') $= \pi$ l› **using** ∗ *kj'* **by**(*simp*, *metis Nat.le-diff-conv2 add-Suc diff-add-inverse le-add1 le-add-diff-inverse2*)
      moreover
      have ‹l'' + *Suc* $k' < l'$› **by** (*metis a jl' add-diff-cancel-right' kj' le-add-diff-inverse less-imp-diff-less ordered-cancel-comm-monoid-diff-class.le-diff-conv2*)
      moreover
      have ‹l'' + *Suc* $k' < m'$› **by** (*metis Suc-lessD calculation*(*2*) *less-trans-Suc lm'*)
      moreover
      have ‹$k' < l''$ + *Suc* k'› **by** *simp*
      ultimately
      show ‹*False*› **using** *Least-le*[*of* ‹*?P*› ‹l'' + *Suc* k'›] *l'* **by** *auto*
    next
      assume *a*: ‹¬ $l'' \leq j'$ − *Suc* k'›
      hence ‹¬ *Suc* (k' + l'') $\leq j'$› **by** *simp*
      hence ‹π_1 (*Suc* (k' + l'') − j') $= \pi$ l› **using** ∗ *kj'* **by** *simp*
      moreover
      have ‹*Suc* (k' + l'') − $j' \leq n_1$› **using** ∗∗ *kj'* **by** *simp*
      ultimately
      show ‹*False*› **using** *nl'* **by** (*metis πl'*)
    qed
  qed
  thus ‹*?thesis*› **unfolding** *is-icdi-def* **using** *lcd' path*(*2*) **by** *simp*
qed
hence ‹$cs^{\pi'}$ l' = $cs^{\pi'}$ k' @ [π' l']› **by** (*metis icd-cs*)
hence ‹$cs^{\pi'}$ l' = cs^{π} l› **by** (*metis πl' csk icd icd-cs*)
thus ‹*?thesis*› **by** *metis*
qed

lemma *converged-same-icd*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π'› **and** *converge*: ‹$l < n$› ‹cs^{π} n = $cs^{\pi'}$ n'› **and** *csk*: ‹cs^{π} k = $cs^{\pi'}$ k'› **and** *icd*: ‹l $icd^{\pi}\rightarrow$ k› **and** *suc*: ‹π (*Suc k*) $= \pi'$ (*Suc* k')›

shows ‹$\exists l'.\ cs^{\pi}\ l = cs^{\pi'}\ l'$› **proof** –

have *nret*: ‹$\pi\ k \neq$ *return*› **using** *icd* **unfolding** *is-icdi-def is-cdi-def* **using** *term-path-stable less-imp-le* **by** *metis*
have *kl*: ‹$k < l$› **using** *icd* **unfolding** *is-icdi-def is-cdi-def* **by** *auto*
have *kn*: ‹$k < n$› **using** *converge kl* **by** *simp*
from *path-ipd-swap*[*OF path*(*1*) *nret kn*]
obtain ϱ m **where** *path*ϱ: ‹*is-path* ϱ› **and** $\pi\varrho$: ‹$\pi =_n \ \varrho$› **and** *km*: ‹$k < m$› **and** *ipd*: ‹$\varrho\ m =$ *ipd* ($\varrho\ k$)› ‹$\forall\ l \in \{k..<m\}.\ \varrho\ l \neq$ *ipd* ($\varrho\ k$)› **.**
have *csk1*: ‹$cs^{\varrho}\ k = cs^{\pi}\ k$› **using** *cs-path-swap-le path path*ϱ $\pi\varrho$ *kn* **by** *auto*
have *suc*ϱ: ‹ϱ (*Suc* k) $= \pi$ (*Suc* k)› **by** (*metis* $\pi\varrho$ *eq-up-to-def kn less-eq-Suc-le*)

have *nret*$'$: ‹$\pi'\ k' \neq$ *return*› **by** (*metis csk last-cs nret*)
have *kn*$'$: ‹$k' < n'$› **using** *cs-order*[*OF path csk converge*(*2*) *nret kn*] **.**
from *path-ipd-swap*[*OF path*(*2*) *nret*$'$ *kn*$'$]
obtain ϱ' m' **where** *path*ϱ': ‹*is-path* ϱ'› **and** $\pi\varrho'$: ‹$\pi' =_{n'} \varrho'$› **and** *km*$'$: ‹$k' < m'$› **and** *ipd*$'$: ‹$\varrho'\ m' =$ *ipd* ($\varrho'\ k'$)› ‹$\forall\ l \in \{k'..<m'\}.\ \varrho'\ l \neq$ *ipd* ($\varrho'\ k'$)› **.**
have *csk1*$'$: ‹$cs^{\varrho'}\ k' = cs^{\pi'}\ k'$› **using** *cs-path-swap-le path path*ϱ' $\pi\varrho'$ *kn*$'$ **by** *auto*
have *suc*ϱ': ‹ϱ' (*Suc* k') $= \pi'$ (*Suc* k')› **by** (*metis* $\pi\varrho'$ *eq-up-to-def kn*$'$ *less-eq-Suc-le*)

have *icd*ϱ: ‹l *icd*$^{\varrho}\!\rightarrow k$› **using** *icdi-path-swap-le*[*OF path*ϱ *icd* $\pi\varrho$] *converge* **by** *simp*

have *lm*: ‹$l < m$› **using** *ipd*(*1*) *icd*ϱ *km* **unfolding** *is-icdi-def is-cdi-def* **by** *auto*

have *csk*$'$: ‹$cs^{\varrho}\ k = cs^{\varrho'}\ k'$› **using** *csk1 csk1*$'$ *csk* **by** *auto*

hence *kk*$'$: ‹$\varrho'\ k' = \varrho\ k$› **using** *last-cs* **by** *metis*

have *suc*$'$: ‹ϱ (*Suc* k) $= \varrho'$ (*Suc* k')› **using** *suc suc*ϱ *suc*ϱ' **by** *auto*

have *mm*$'$: ‹$\varrho'\ m' = \varrho\ m$› **using** *ipd*(*1*) *ipd*$'$(*1*) *kk*$'$ **by** *auto*

from *cs-ipd*[*OF ipd km*] *cs-ipd*[*OF ipd*$'$ *km*$'$,*unfolded mm*$'$, *folded csk*$'$]
have *csm*: ‹$cs^{\varrho}\ m = cs^{\varrho'}\ m'$› **by** *metis*

from *converged-ipd-same-icd*[*OF path*ϱ *path*ϱ' *lm csm csk*$'$ *icd*ϱ *suc*$'$ *ipd*$'$[*unfolded kk*$'$]]
obtain l' **where** *csl*: ‹$cs^{\varrho}\ l = cs^{\varrho'}\ l'$› **by** *blast*

have *csl*ϱ: ‹$cs^{\pi}\ l = cs^{\varrho}\ l$› **using** $\pi\varrho$ *converge*(*1*) *cs-path-swap-le less-imp-le-nat path*(*1*) *path*ϱ **by** *blast*

have *nretl*: ‹$\varrho\ l \neq$ *return*› **by** (*metis icd*ϱ *icd-imp-cd ret-no-cd*)

have *csn*$'$: ‹$cs^{\varrho}\ n = cs^{\varrho'}\ n'$› **using** *converge*(*2*) *cs-path-swap path path*ϱ *path*ϱ' $\pi\varrho$ $\pi\varrho'$ **by** *auto*

have *ln*$'$: ‹$l' < n'$› **using** *cs-order*[*OF path*ϱ *path*ϱ' *csl csn*$'$ *nretl converge*(*1*)] **.**

have *csl*ϱ': ‹$cs^{\pi'}\ l' = cs^{\varrho'}\ l'$› **using** *cs-path-swap-le*[*OF path*(*2*) *path*ϱ' $\pi\varrho'$] *ln*$'$ **by** *auto*

have *csl*$'$: ‹$cs^{\pi}\ l = cs^{\pi'}\ l'$› **using** *csl*$\varrho$ *csl*ϱ' *csl* **by** *auto*
thus ‹*?thesis*› **by** *blast*
qed

lemma *cd-is-cs-less*: **assumes** ‹l *cd*$^{\pi}\!\rightarrow k$› **shows** ‹$cs^{\pi}\ k \prec cs^{\pi}\ l$› **proof** –
obtain *xs* **where** *csl*: ‹$cs^{\pi}\ l = cs^{\pi}\ k$ @ *xs* @[$\pi\ l$]› **using** *cd-in-cs*[*OF assms*] **by** *blast*
hence *len*: ‹*length*($cs^{\pi}\ k$) < *length* ($cs^{\pi}\ l$)› **by** *auto*

have *take*: ‹*take* (*length* (cs^π *k*)) (cs^π *l*) = cs^π *k*› **using** *csl* **by** *auto*
show ‹*?thesis*› **using** *cs-less.intros*[*OF len take*] **.**
qed

lemma *cs-select-id*: **assumes** ‹*is-path* π› ‹π *k* ≠ *return*› **shows** ‹π¡cs^π *k* = *k*› (**is** ‹*?k* = *k*›) **proof** −
have ∗: ‹⋀ *i* . cs^π *i* = cs^π *k* ⟹ *i* = *k*› **using** *cs-inj*[*OF assms*] **by** *metis*
hence ‹cs^π *?k* = cs^π *k*› **unfolding** *cs-select-def* **using** *theI*[*of* ‹λ *i*. cs^π *i* = cs^π *k*› ‹*k*›] **by** *auto*
thus ‹*?k* = *k*› **using** ∗ **by** *auto*
qed

lemma *cs-single-nocd*: **assumes** ‹cs^π *i* = [*x*]› **shows** ‹∀ *k*. ¬ *i* cd^π→ *k*› **proof** −
have ‹¬ (∃ *k*. *i* icd^π→ *k*)› **apply** (*rule ccontr*) **using** *assms cs-not-nil* **by** *auto*
hence ‹¬ (∃ *k*. *i* cd^π→ *k*)› **by** (*metis excd-impl-exicd*)
thus ‹*?thesis*› **by** *blast*
qed

lemma *cs-single-pd-intermed*: **assumes** ‹*is-path* π› ‹cs^π *n* = [π *n*]› ‹*k* ≤ *n*› **shows** ‹π *n pd*→ π *k*› **proof** −
have ‹∀ *l*. ¬ *n* icd^π→ *l*› **by** (*metis assms*(*2*) *cs-single-nocd icd-imp-cd*)
thus ‹*?thesis*› **by** (*metis assms*(*1*) *assms*(*3*) *no-icd-pd*)
qed

lemma *cs-first-pd*: **assumes** *path*: ‹*is-path* π› **and** *pd*: ‹π *n pd*→ π *0*› **and** *first*: ‹∀ *l* < *n*. π *l* ≠ π *n*› **shows** ‹cs^π *n* = [π *n*]›
by (*metis cs-cases first first-pd-no-cd icd-imp-cd path pd*)

lemma *converged-pd-cs-single*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π′› **and** *converge*: ‹*l* < *m*› ‹cs^π *m* = $cs^{\pi'}$ *m*′›
and *π0*: ‹π *0* = π′ *0*› **and** *mpdl*: ‹π *m pd*→ π *l*› **and** *csl*: ‹cs^π *l* = [π *l*]›
shows ‹∃ *l*′. cs^π *l* = $cs^{\pi'}$ *l*′› **proof** −
have ∗: ‹π *l pd*→ π′ *0*› **using** *cs-single-pd-intermed*[*OF path*(*1*) *csl*] *π0*[*symmetric*] **by** *auto*
have *πm*: ‹π *m* = π′ *m*′› **by** (*metis converge*(*2*) *last-cs*)
hence ∗∗: ‹π′ *m*′ *pd*→ π *l*› **using** *mpdl* **by** *metis*

obtain *l*′ **where** *lm*′: ‹*l*′ ≤ *m*′› **and** *πl*: ‹π′ *l*′ = π *l*› (**is** ‹*?P l*′›) **using** *path-pd-pd0*[*OF path*(*2*) ∗∗ ∗] **.**

let ‹*?l*› = ‹(*LEAST l*′. π′ *l*′ = π *l*)›

have *πl*′: ‹π′ *?l* = π *l*› **using** *LeastI*[*of* ‹*?P*›,*OF πl*] **.**
moreover
have ‹∀ *i* <*?l*. π′ *i* ≠ π *l*› **using** *Least-le*[*of* ‹*?P*›] **by** (*metis not-less*)
hence ‹∀ *i* <*?l*. π′ *i* ≠ π′ *?l*› **using** *πl*′ **by** *metis*
moreover
have ‹π′ *?l pd*→ π′ *0*› **using** ∗ *πl*′ **by** *metis*
ultimately
have ‹$cs^{\pi'}$ *?l* = [π′ *?l*]› **using** *cs-first-pd*[*OF path*(*2*)] **by** *metis*
thus ‹*?thesis*› **using** *csl πl*′ **by** *metis*
qed

lemma *converged-cs-single*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π′› **and** *converge*: ‹*l* < *m*› ‹cs^π *m* = $cs^{\pi'}$ *m*′›

and *π0*: ‹π *0* = π′ *0*› **and** *csl*: ‹cs^π *l* = [π *l*]›
shows ‹∃ *l*′. cs^π *l* = $cs^{\pi'}$ *l*′› **proof** *cases*
assume ∗: ‹π *l* = *return*›
hence ‹π *m* = *return*› **by** (*metis converge*(*1*) *path*(*1*) *term-path-stable less-imp-le*)

hence ‹cs^{π} m = [*return*]› **using** *cs-return* **by** *auto*
hence ‹$cs^{\pi'}$ m' = [*return*]› **using** *converge* **by** *simp*
moreover
have ‹cs^{π} l = [*return*]› **using** $*$ *cs-return* **by** *auto*
ultimately show ‹*?thesis*› **by** *metis*
next
assume *nret*: ‹π $l \neq$ *return*›
have πm: ‹π m = π' m'› **by** (*metis converge*(*2*) *last-cs*)

obtain π_1 n **where** *path1*: ‹*is-path* π_1› **and** *upto*: ‹$\pi =_m \pi_1$› **and** πn: ‹π_1 n = *return*› **using** *path*(*1*) *path-swap-ret* **by** *blast*

obtain π_1' n' **where** *path1'*: ‹*is-path* π_1'› **and** *upto'*: ‹$\pi' =_{m'} \pi_1'$› **and** $\pi n'$: ‹π_1' n' = *return*› **using** *path*(*2*) *path-swap-ret* **by** *blast*

have $\pi 1l$: ‹π_1 l = π l› **using** *upto converge*(*1*) **by** (*metis eq-up-to-def nat-less-le*)

have *cs1l*: ‹cs^{π_1} l = cs^{π} l› **using** *cs-path-swap-le upto path1 path*(*1*) *converge*(*1*) **by** *auto*

have *csl1*: ‹cs^{π_1} l = [π_1 l]› **by** (*metis* $\pi 1l$ *cs1l csl*)

have *converge1*: ‹cs^{π_1} n = $cs^{\pi_1'}$ n'› **using** πn $\pi n'$ *cs-return* **by** *auto*

have *ln*: ‹$l < n$› **using** *nret* πn $\pi 1l$ *term-path-stable*[*OF path1* πn] **by** (*auto*, *metis linorder-neqE-nat less-imp-le*)

have $\pi 01$: ‹π_1 *0* = π_1' *0*› **using** $\pi 0$ *eq-up-to-apply*[*OF upto*] *eq-up-to-apply*[*OF upto'*] **by** *auto*

have *pd*: ‹π_1 n *pd*$\rightarrow$ π_1 l› **using** πn **by** (*metis path1 path-nodes return-pd*)

obtain l' **where** *csl*: ‹cs^{π_1} l = $cs^{\pi_1'}$ l'› **using** *converged-pd-cs-single*[*OF path1 path1' ln converge1* $\pi 01$ *pd csl1*] **by** *blast*

have *cs1m*: ‹cs^{π_1} m = cs^{π} m› **using** *cs-path-swap upto path1 path*(*1*) **by** *auto*
have *cs1m'*: ‹$cs^{\pi_1'}$ m' = $cs^{\pi'}$ m'› **using** *cs-path-swap upto' path1' path*(*2*) **by** *auto*
hence *converge1*: ‹cs^{π_1} m = $cs^{\pi_1'}$ m'› **using** *converge*(*2*) *cs1m* **by** *metis*

have *nret1*: ‹π_1 $l \neq$ *return*› **using** *nret* $\pi 1l$ **by** *auto*

have *lm'*: ‹$l' < m'$› **using** *cs-order*[*OF path1 path1' csl converge1 nret1 converge*(*1*)] **.**

have ‹$cs^{\pi'}$ l' = $cs^{\pi_1'}$ l'› **using** *cs-path-swap-le*[*OF path*(*2*) *path1' upto'*] *lm'* **by** *auto*
moreover
have ‹cs^{π} l = cs^{π_1} l› **using** *cs-path-swap-le*[*OF path*(*1*) *path1 upto*] *converge*(*1*) **by** *auto*
ultimately
have ‹cs^{π} l = $cs^{\pi'}$ l'› **using** *csl* **by** *auto*
thus ‹*?thesis*› **by** *blast*
qed

lemma *converged-cd-same-suc*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π'› **and** *init*: ‹π *0* = π' *0*›
and *cd-suc*: ‹$\forall$ k k'. cs^{π} k = $cs^{\pi'}$ k' $\wedge$ l $cd^{\pi}\rightarrow$ k $\longrightarrow$ π (*Suc* k) = π' (*Suc* k')› **and** *converge*: ‹$l < m$› ‹cs^{π} m = $cs^{\pi'}$ m'›
shows ‹$\exists$ l'. cs^{π} l = $cs^{\pi'}$ l'›
using *path init cd-suc converge* **proof** (*induction* ‹π› ‹l› *rule*: *cs-induct*,*cases*)
case (*cs* π l)

assume $*$: ‹$\exists\, k.\ l\ icd^{\pi}\!\rightarrow k$›
let ‹?k› = ‹THE $k.\ l\ icd^{\pi}\!\rightarrow k$›
have *icd*: ‹$l\ icd^{\pi}\!\rightarrow\ ?k$› **by** (*metis* $*$ *icd-is-the-icd*)
hence *lcdk*: ‹$l\ cd^{\pi}\!\rightarrow\ ?k$› **by** (*metis is-icdi-def*)
hence *kl*: ‹$?k<l$› **using** *is-cdi-def* **by** *metis*

have ‹$\bigwedge j.\ ?k\ cd^{\pi}\!\rightarrow j \Longrightarrow l\ cd^{\pi}\!\rightarrow j$› **using** *icd cd-trans is-icdi-def* **by** *fast*
hence *suc'*: ‹$\forall\ j\ j'.\ cs^{\pi}\ j = cs^{\pi'}\ j' \wedge\ ?k\ cd^{\pi}\!\rightarrow j \longrightarrow \pi\ (Suc\ j) = \pi'\ (Suc\ j')$› **using** *cs.prems(4)* **by** *blast*

from *cs.IH[OF* $*$ *cs(2) path(2) cs(4) suc']* *cs.prems kl*
have ‹$\exists\, k'.\ cs^{\pi}\ (THE\ k.\ l\ icd^{\pi}\!\rightarrow k) = cs^{\pi'}\ k'$› **by** (*metis Suc-lessD less-trans-Suc*)
then obtain k' **where** *csk*: ‹$cs^{\pi}\ ?k = cs^{\pi'}\ k'$› **by** *blast*

have *suc2*: ‹$\pi\ (Suc\ ?k) = \pi'\ (Suc\ k')$› **using** *cs.prems(4) lcdk csk* **by** *auto*

have *km*: ‹$?k < m$› **using** *kl cs.prems(5)* **by** *simp*

from *converged-same-icd[OF cs(2) path(2) cs.prems(5) cs.prems(6) csk icd suc2]*
show ‹?case› **.**
next
case ($cs\ \pi\ l$)
assume ‹$\neg\ (\exists\, k.\ l\ icd^{\pi}\!\rightarrow k)$›
hence ‹$cs^{\pi}\ l = [\pi\ l]$› **by** *auto*
with *cs converged-cs-single*
show ‹?case› **by** *metis*
qed

lemma *converged-cd-diverge*:
assumes *path*: ‹$is\text{-}path\ \pi$› ‹$is\text{-}path\ \pi'$› **and** *init*: ‹$\pi\ 0 = \pi'\ 0$› **and** *notin*: ‹$\neg\ (\exists\, l'.\ cs^{\pi}\ l = cs^{\pi'}\ l')$› **and**
converge: ‹$l < m$› ‹$cs^{\pi}\ m = cs^{\pi'}\ m'$›
obtains $k\ k'$ **where** ‹$cs^{\pi}\ k = cs^{\pi'}\ k'$› ‹$l\ cd^{\pi}\!\rightarrow k$› ‹$\pi\ (Suc\ k) \neq \pi'\ (Suc\ k')$›
using *assms converged-cd-same-suc* **by** *blast*

lemma *converged-cd-same-suc-return*: **assumes** *path*: ‹$is\text{-}path\ \pi$› ‹$is\text{-}path\ \pi'$› **and** *π0*: ‹$\pi\ 0 = \pi'\ 0$›
and *cd-suc*: ‹$\forall\ k\ k'.\ cs^{\pi}\ k = cs^{\pi'}\ k' \wedge l\ cd^{\pi}\!\rightarrow k \longrightarrow \pi\ (Suc\ k) = \pi'\ (Suc\ k')$› **and** *ret*: ‹$\pi'\ n' = return$›
shows ‹$\exists\, l'.\ cs^{\pi}\ l = cs^{\pi'}\ l'$›**proof** *cases*
assume ‹$\pi\ l = return$›
hence ‹$cs^{\pi}\ l = cs^{\pi'}\ n'$› **using** *ret cs-return* **by** *presburger*
thus ‹?thesis› **by** *blast*
next
assume *nretl*: ‹$\pi\ l \neq return$›
have ‹$\pi\ l \in nodes$› **using** *path path-nodes* **by** *auto*
then obtain $\pi l\ n$ **where** *ipl*: ‹$is\text{-}path\ \pi l$› **and** *πl*: ‹$\pi\ l = \pi l\ 0$› **and** *retn*: ‹$\pi l\ n = return$› **and** *notl*: ‹$\forall\ i{>}0.\ \pi l\ i \neq \pi\ l$› **by** (*metis direct-path-return nretl*)
hence *ip*: ‹$is\text{-}path\ (\pi @^{l}\ \pi l)$› **and** *l*: ‹$(\pi @^{l}\ \pi l)\ l = \pi\ l$› **and** *retl*: ‹$(\pi @^{l}\ \pi l)\ (l + n) = return$› **and** *nl*: ‹$\forall\ i{>}l.\ (\pi @^{l}\ \pi l)\ i \neq \pi\ l$› **using** *path-cons[OF path(1) ipl πl]* **by** *auto*

have *π0'*: ‹$(\pi @^{l}\ \pi l)\ 0 = \pi'\ 0$› **unfolding** *cs-0* **using** *πl π0* **by** *auto*

have *csn*: ‹$cs^{\pi @^{l}\ \pi l}\ (l{+}n) = cs^{\pi'}\ n'$› **using** *ret retl cs-return* **by** *metis*

have *eql*: ‹$(\pi @^{l}\ \pi l) =_{l} \pi$› **by** (*metis path-append-eq-up-to*)

have *csl′*: ‹$cs^{\pi @^{l} \pi l}\ \ l = cs^{\pi}\ l$› **using** *eql cs-path-swap ip path*(*1*) **by** *metis*

have ‹$0 < n$› **using** *nretl*[*unfolded* πl] *retn* **by** (*metis neq0-conv*)
hence *ln*: ‹$l < l + n$› **by** *simp*

have $*$: ‹$\forall\ \ k\ k'.\ cs^{\pi\ @^{l}\ \pi l}\ \ k\ =\ cs^{\pi'}\ k' \wedge\ l\ cd^{\pi\ @^{l}\ \pi l}{\rightarrow}\ k\ \longrightarrow\ (\pi\ @^{l}\ \pi l)\ (Suc\ k)\ =\ \pi'\ (Suc\ k')$› **proof** (*rule*,*rule*,*rule*)
fix $k\ k'$ **assume** $*$: ‹$cs^{\pi\ @^{l}\ \pi l}\ \ k\ =\ cs^{\pi'}\ k' \wedge\ l\ cd^{\pi\ @^{l}\ \pi l}{\rightarrow}\ k$›
hence *kl*: ‹$k < l$› **using** *is-cdi-def* **by** *auto*
hence ‹$cs^{\pi}\ k = cs^{\pi'}\ k' \wedge l\ cd^{\pi}{\rightarrow}\ k$› **using** *eql* $*$ *cs-path-swap-le*[*OF ip path*(*1*) *eql*,*of* ‹*k*›] *cdi-path-swap-le*[*OF path*(*1*) - *eql*,*of* ‹*l*› ‹*k*›] **by** *auto*
hence ‹$\pi\ (Suc\ k) = \pi'\ (Suc\ k')$› **using** *cd-suc* **by** *blast*
then show ‹$(\pi\ @^{l}\ \pi l)\ (Suc\ k) = \pi'\ (Suc\ k')$› **using** *cs-path-swap-le*[*OF ip path*(*1*) *eql*,*of* ‹*Suc k*›] *kl* **by** *auto*
qed
obtain l' **where** ‹$cs^{\pi\ @^{l}\ \pi l}\ \ l = cs^{\pi'}\ l'$› **using** *converged-cd-same-suc*[*OF ip path*(*2*) $\pi 0'$ $*$ *ln csn*] **by** *blast*
moreover
have ‹$cs^{\pi @^{l}\ \pi l}\ \ l = cs^{\pi}\ l$› **using** *eql* **by** (*metis cs-path-swap ip path*(*1*))
ultimately
show ‹*?thesis*› **by** *metis*
qed

lemma *converged-cd-diverge-return*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π'› **and** *init*: ‹$\pi\ 0 = \pi'\ 0$›
and *notin*: ‹$\neg\ (\exists\, l'.\ cs^{\pi}\ l = cs^{\pi'}\ l')$› **and** *ret*: ‹$\pi'\ m' = return$›
obtains $k\ k'$ **where** ‹$cs^{\pi}\ k = cs^{\pi'}\ k'$› ‹$l\ cd^{\pi}{\rightarrow}\ k$› ‹$\pi\ (Suc\ k) \neq \pi'\ (Suc\ k')$› **using** *converged-cd-same-suc-return*[*OF path init - ret*, *of* ‹*l*›] *notin* **by** *blast*

lemma *returned-missing-cd-or-loop*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π'› **and** $\pi 0$: ‹$\pi\ 0 = \pi'\ 0$›
and *notin′*: ‹$\neg(\exists\ k'.\ cs^{\pi}\ k = cs^{\pi'}\ k')$› **and** *nret*: ‹$\forall\ n'.\ \pi'\ n' \neq return$› **and** *ret*: ‹$\pi\ n = return$›
obtains $i\ i'$ **where** ‹$i<k$› ‹$cs^{\pi}\ i = cs^{\pi'}\ i'$› ‹$\pi\ (Suc\ i) \neq \pi'\ (Suc\ i')$› ‹$k\ cd^{\pi}{\rightarrow}\ i \vee (\forall\ j'> i'.\ j'\ cd^{\pi'}{\rightarrow}\ i')$›
proof −
obtain *f* **where** *icdf*: ‹$\forall\ i'.\ f\ (Suc\ i')\ icd^{\pi'}{\rightarrow}\ f\ i'$› **and** *ran*: ‹$range\ f = \{i'.\ \forall\ j'{>}i'.\ j'\ cd^{\pi'}{\rightarrow}\ i'\}$› **and** *icdf0*: ‹$\neg\ (\exists\, i'.\ f\ 0\ cd^{\pi'}{\rightarrow}\ i')$› **using** *path*(*2*) *path-nret-inf-icd-seq nret* **by** *blast*
show ‹*thesis*› **proof** *cases*
assume ‹$\exists\ j.\ \neg\ (\exists\ i.\ cs^{\pi}\ i = cs^{\pi'}\ (f\, j))$›
then obtain *j* **where** *ni*π: ‹$\neg\ (\exists\ i.\ cs^{\pi'}\ (f\, j) = cs^{\pi}\ i)$› **by** *metis*
note *converged-cd-diverge-return*[*OF path*(*2*,*1*) $\pi 0$[*symmetric*] *ni*π *ret*] *that*
then obtain $i\ k'$ **where** *csk*: ‹$cs^{\pi}\ i = cs^{\pi'}\ k'$› **and** *cdj*: ‹$f\, j\ cd^{\pi'}{\rightarrow}\ k'$› **and** *div*: ‹$\pi\ (Suc\ i) \neq \pi'\ (Suc\ k')$› **by** *metis*
have ‹$k' \in range\ f$› **using** *cdj* **proof** (*induction* ‹*j*›)
case *0* **thus** ‹*?case*› **using** *icdf0* **by** *blast*
next
case (*Suc j*)
have *icdfj*: ‹$f\ (Suc\ j)\ icd^{\pi'}{\rightarrow}\ f\, j$› **using** *icdf* **by** *auto*
show ‹*?case*› **proof** *cases*
assume ‹$f\ (Suc\ j)\ icd^{\pi'}{\rightarrow}\ k'$›
hence ‹$k' = f\, j$› **using** *icdfj* **by** (*metis icd-uniq*)
thus ‹*?case*› **by** *auto*
next
assume ‹$\neg\ f\ (Suc\ j)\ icd^{\pi'}{\rightarrow}\ k'$›
hence ‹$f\, j\ cd^{\pi'}{\rightarrow}\ k'$› **using** *cd-impl-icd-cd*[*OF Suc.prems icdfj*] **by** *auto*

thus ‹*?case*› **using** *Suc.IH* **by** *auto*
qed
qed
hence *alldep*: ‹$\forall\ i'{>}k'.\ i'\ cd^{\pi'}\!\!\rightarrow k'$› **using** *ran* **by** *auto*
show ‹*thesis*› **proof** *cases*
assume ‹$i < k$› **with** *alldep that*[*OF - csk div*] **show** ‹*thesis*› **by** *blast*
next
assume ‹$\neg\ i < k$›
hence *ki*: ‹$k{\le}i$› **by** *auto*
have ‹$k \neq i$› **using** *notin' csk* **by** *auto*
hence *ki'*: ‹$k{<}i$› **using** *ki* **by** *auto*
obtain *ka k'* **where** ‹$cs^{\pi}\ ka = cs^{\pi'}\ k'$› ‹$k\ cd^{\pi}\!\!\rightarrow ka$› ‹$\pi\ (Suc\ ka) \neq \pi'\ (Suc\ k')$›
using *converged-cd-diverge*[*OF path π0 notin' ki' csk*] **by** *blast*
moreover
hence ‹$ka < k$› **unfolding** *is-cdi-def* **by** *auto*
ultimately
show ‹*?thesis*› **using** *that* **by** *blast*
qed
next
assume ‹$\neg(\exists\ j.\ \neg\ (\exists\ i.\ cs^{\pi}\ i = cs^{\pi'}\ (f\ j)))$›
hence *allin*: ‹$\forall\ j.\ (\exists\ i.\ cs^{\pi}\ i = cs^{\pi'}\ (f\ j))$› **by** *blast*
define f' **where** f': ‹$f' \equiv \lambda\ j.\ (SOME\ i.\ cs^{\pi}\ i = cs^{\pi'}\ (f\ j))$›
have ‹$\forall\ i.\ f'\ i < f'\ (Suc\ i)$› **proof**
fix *i*
have *csi*: ‹$cs^{\pi'}\ (f\ i) = cs^{\pi}\ (f'\ i)$› **unfolding** f' **using** *allin* **by** (*metis* (*mono-tags*) *someI-ex*)
have *cssuci*: ‹$cs^{\pi'}\ (f\ (Suc\ i)) = cs^{\pi}\ (f'\ (Suc\ i))$› **unfolding** f' **using** *allin* **by** (*metis* (*mono-tags*) *someI-ex*)
have *fi*: ‹$f\ i < f\ (Suc\ i)$› **using** *icdf* **unfolding** *is-icdi-def is-cdi-def* **by** *auto*
have ‹$f\ (Suc\ i)\ cd^{\pi'}\!\!\rightarrow f\ i$› **using** *icdf* **unfolding** *is-icdi-def* **by** *blast*
hence *nreti*: ‹$\pi'\ (f\ i) \neq return$› **by** (*metis cd-not-ret*)
show ‹$f'\ i < f'\ (Suc\ i)$› **using** *cs-order*[*OF path(2,1) csi cssuci nreti fi*] **.**
qed
hence *kle*: ‹$k < f'\ (Suc\ k)$› **using** *mono-ge-id*[*of* ‹f'› ‹$Suc\ k$›] **by** *auto*
have *cssk*: ‹$cs^{\pi}\ (f'\ (Suc\ k)) = cs^{\pi'}\ (f\ (Suc\ k))$› **unfolding** f' **using** *allin* **by** (*metis* (*mono-tags*) *someI-ex*)
obtain *ka k'* **where** ‹$cs^{\pi}\ ka = cs^{\pi'}\ k'$› ‹$k\ cd^{\pi}\!\!\rightarrow ka$› ‹$\pi\ (Suc\ ka) \neq \pi'\ (Suc\ k')$›
using *converged-cd-diverge*[*OF path π0 notin' kle cssk*] **by** *blast*
moreover
hence ‹$ka < k$› **unfolding** *is-cdi-def* **by** *auto*
ultimately
show ‹*?thesis*› **using** *that* **by** *blast*
qed
qed

lemma *missing-cd-or-loop*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π'› **and** *π0*: ‹$\pi\ 0 = \pi'\ 0$› **and** *notin'*: ‹$\neg(\exists\ k'.\ cs^{\pi}\ k = cs^{\pi'}\ k')$›
obtains *i i'* **where** ‹$i < k$› ‹$cs^{\pi}\ i = cs^{\pi'}\ i'$› ‹$\pi\ (Suc\ i) \neq \pi'\ (Suc\ i')$› ‹$k\ cd^{\pi}\!\!\rightarrow i \vee (\forall\ j'{>}\ i'.\ j'\ cd^{\pi'}\!\!\rightarrow i')$›
proof *cases*
assume ‹$\exists\ n'.\ \pi'\ n' = return$›
then obtain n' **where** *retn*: ‹$\pi'\ n' = return$› **by** *blast*
note *converged-cd-diverge-return*[*OF path π0 notin' retn*]
then obtain *ka k'* **where** ‹$cs^{\pi}\ ka = cs^{\pi'}\ k'$› ‹$k\ cd^{\pi}\!\!\rightarrow ka$› ‹$\pi\ (Suc\ ka) \neq \pi'\ (Suc\ k')$› **by** *blast*
moreover
hence ‹$ka < k$› **unfolding** *is-cdi-def* **by** *auto*
ultimately show ‹*thesis*› **using** *that* **by** *simp*

next
  assume ‹¬ (∃ n'. π' n' = *return*)›
  hence *notret*: ‹∀ n'. π' n' ≠ *return*› **by** *auto*
  then obtain πl *n* **where** *ipl*: ‹*is-path* πl› **and** πl: ‹π *k* = πl *0*› **and** *retn*: ‹πl *n* = *return*› **using** *reaching-ret path*(*1*) *path-nodes* **by** *metis*
  hence *ip*: ‹*is-path* (π@$^k\pi l$)› **and** *l*: ‹(π@$^k\pi l$) *k* = π *k*› **and** *retl*: ‹(π@$^k\pi l$) (*k* + *n*) = *return*› **using** *path-cons*[*OF path*(*1*) *ipl* πl] **by** *auto*

  have $\pi 0'$: ‹(π@$^k\pi l$) *0* = π' *0*› **unfolding** *cs-0* **using** πl $\pi 0$ **by** *auto*

  have *eql*: ‹(π@$^k\pi l$) $=_k$ π› **by** (*metis path-append-eq-up-to*)

  have *csl'*: ‹$cs^{\pi @^k \pi l}$ *k* = cs^{π} *k*› **using** *eql cs-path-swap ip path*(*1*) **by** *metis*

  hence *notin*: ‹¬(∃ k'. $cs^{\pi @^k \pi l}$ *k* = $cs^{\pi'}$ k')› **using** *notin'* **by** *auto*

  obtain *i* i' **where** ∗: ‹*i* < *k*› **and** *csi*: ‹$cs^{\pi @^k \pi l}$ *i* = $cs^{\pi'}$ i'› **and** *suci*: ‹(π @^{k} πl) (*Suc i*) ≠ π' (*Suc* i')› **and** *cdloop*: ‹*k* $cd^{\pi @^k \pi l}$→ *i* ∨ (∀ j'>i'. j' $cd^{\pi'}$→ i')›
  using *returned-missing-cd-or-loop*[*OF ip path*(*2*) $\pi 0'$ *notin notret retl*] **by** *blast*

  have ‹*i* ≠ *k*› **using** *notin csi* **by** *auto*
  hence *ik*: ‹*i* < *k*› **using** ∗ **by** *auto*
  hence ‹cs^{π} *i* = $cs^{\pi'}$ i'› **using** *csi cs-path-swap-le*[*OF ip path*(*1*) *eql*] **by** *auto*
  moreover
  have ‹π (*Suc i*) ≠ π' (*Suc* i')› **using** *ik eq-up-to-apply*[*OF eql*, *of* ‹*Suc i*›] *suci* **by** *auto*
  moreover
  have ‹*k* cd^{π}→ *i* ∨ (∀ j'>i'. j' $cd^{\pi'}$→ i')› **using** *cdloop cdi-path-swap-le*[*OF path*(*1*) - *eql*, *of* ‹*k*› ‹*i*›] **by** *auto*
  ultimately
  show ‹*thesis*› **using** *that*[*OF* ∗] **by** *blast*
qed

lemma *path-shift-set-cd*: **assumes** ‹*is-path* π› **shows** ‹{*k* + *j*| *j* . *n* $cd^{\pi «k}$→ *j* } = {*i*. (*k*+*n*) cd^{π}→ *i* ∧ *k* ≤ *i* }›
proof −
  { **fix** *i*
    assume ‹*i*∈{*k*+*j* | *j* . *n* $cd^{\pi «k}$→ *j* }›
    then obtain *j* **where** ‹*i* = *k*+*j*› ‹*n* $cd^{\pi «k}$→ *j*› **by** *auto*
    hence ‹*k*+*n* cd^{π}→ *i* ∧ *k* ≤ *i*› **using** *cd-path-shift*[*OF* - *assms*, *of* ‹*k*› ‹*k*+*j*› ‹*k*+*n*›] **by** *simp*
    hence ‹*i*∈{ *i*. *k*+*n* cd^{π}→ *i* ∧ *k* ≤ *i* }› **by** *blast*
  }
  moreover
  { **fix** *i*
    assume ‹*i*∈{ *i*. *k*+*n* cd^{π}→ *i* ∧ *k* ≤ *i* }›
    hence ∗: ‹*k*+*n* cd^{π}→ *i* ∧ *k* ≤ *i*› **by** *blast*
    then obtain *j* **where** *i*: ‹*i* = *k*+*j*› **by** (*metis le-Suc-ex*)
    hence ‹*k*+*n* cd^{π}→ *k*+*j*› **using** ∗ **by** *auto*
    hence ‹*n* $cd^{\pi «k}$→ *j*› **using** *cd-path-shift*[*OF* - *assms*, *of* ‹*k*› ‹*k*+*j*› ‹*k*+*n*›] **by** *simp*
    hence ‹*i*∈{*k*+*j* | *j* . *n* $cd^{\pi «k}$→ *j* }› **using** *i* **by** *simp*
  }
  ultimately show ‹*?thesis*› **by** *blast*
qed

lemma *cs-path-shift-set-cd*: **assumes** *path*: ‹*is-path* π› **shows** ‹$cs^{\pi «k}$ *n* = *map* π (*sorted-list-of-set* {*i*. *k*+*n*

$cd^{\pi}\rightarrow\ i \land k \leq i$ }) @ [π (k+n)]›
proof −
  have *mono*:‹∀ n m. n < m ⟶ k + n < k + m› **by** *auto*
  have *fin*: ‹*finite* {i. n $cd^{\pi\ «\ k}\rightarrow$ i}› **unfolding** *is-cdi-def* **by** *auto*
  have ∗: ‹(λ x. k+x) ‘{i. n $cd^{\pi«k}\rightarrow$ i} = {k + i | i. n $cd^{\pi«k}\rightarrow$ i}› **by** *auto*
  have ‹$cs^{\pi«k}$ n = *map* (π«k) (*sorted-list-of-set* {i. n $cd^{\pi«k}\rightarrow$ i}) @ [(π«k) n]› **using** *cs-sorted-list-of-cd′* **by** *blast*
  also
  have ‹. . . = *map* π (*map* (λ x. k+x) (*sorted-list-of-set*{i. n $cd^{\pi«k}\rightarrow$ i})) @ [π (k+n)]› **by** *auto*
  also
  have ‹. . . = *map* π (*sorted-list-of-set* ((λ x. k+x) ‘{i. n $cd^{\pi«k}\rightarrow$ i})) @ [π (k+n)]› **using** *sorted-list-of-set-map-mono*[*OF mono fin*] **by** *auto*
  also
  have ‹. . . = *map* π (*sorted-list-of-set* ({k + i | i. n $cd^{\pi«k}\rightarrow$ i})) @ [π (k+n)]› **using** ∗ **by** *auto*
  also
  have ‹. . . = *map* π (*sorted-list-of-set* ({i. k+n $cd^{\pi}\rightarrow$ i ∧ k ≤ i})) @ [π (k+n)]› **using** *path-shift-set-cd*[*OF path*] **by** *auto*
  finally
  show ‹*?thesis*› **.**
qed

lemma *cs-split-shift-cd*: **assumes** ‹n $cd^{\pi}\rightarrow$ j› **and** ‹j < k› **and** ‹k < n› **and** ‹∀ j′<k. n $cd^{\pi}\rightarrow$ j′ ⟶ j′ ≤ j›
shows ‹cs^{π} n = cs^{π} j @ $cs^{\pi«k}$ (n−k)›
proof −
  have *path*: ‹*is-path* π› **using** *assms* **unfolding** *is-cdi-def* **by** *auto*
  have *1*: ‹{i. n $cd^{\pi}\rightarrow$ i} = {i. n $cd^{\pi}\rightarrow$ i ∧ i < k} ∪ {i. n $cd^{\pi}\rightarrow$ i ∧ k ≤ i}› **by** *auto*
  have *le*: ‹∀ i∈ {i. n $cd^{\pi}\rightarrow$ i ∧ i < k}. ∀ j∈ {i. n $cd^{\pi}\rightarrow$ i ∧ k ≤ i}. i < j› **by** *auto*

  have *2*: ‹{i. n $cd^{\pi}\rightarrow$ i ∧ i < k} = {i . j $cd^{\pi}\rightarrow$ i} ∪ {j}› **proof** −
    { **fix** *i* **assume** ‹i∈{i. n $cd^{\pi}\rightarrow$ i ∧ i < k}›
      hence *cd*: ‹n $cd^{\pi}\rightarrow$ i› **and** *ik*:‹i < k› **by** *auto*
      have ‹i∈{i . j $cd^{\pi}\rightarrow$ i} ∪ {j}› **proof** *cases*
        assume ‹i < j› **hence** ‹j $cd^{\pi}\rightarrow$ i› **by** (*metis is-cdi-def assms*(*1*) *cd cdi-prefix nat-less-le*)
        thus ‹*?thesis*› **by** *simp*
      next
        assume ‹¬ i < j›
        moreover
        have ‹i ≤ j› **using** *assms*(*4*) *ik cd* **by** *auto*
        ultimately
        show ‹*?thesis*› **by** *auto*
      qed
    }
    moreover
    { **fix** *i* **assume** ‹i∈{i . j $cd^{\pi}\rightarrow$ i} ∪ {j}›
      hence ‹j $cd^{\pi}\rightarrow$ i ∨ i = j› **by** *auto*
       hence ‹i∈{i. n $cd^{\pi}\rightarrow$ i ∧ i < k}› **using** *assms*(*1*,*2*) *cd-trans*[*OF - assms*(*1*)] **apply** *auto* **unfolding** *is-cdi-def*
      by (*metis* (*poly-guards-query*) *diff-diff-cancel diff-is-0-eq le-refl le-trans nat-less-le*)
    }
    ultimately show ‹*?thesis*› **by** *blast*
  qed

  have ‹cs^{π} n = *map* π (*sorted-list-of-set* {i. n $cd^{\pi}\rightarrow$ i}) @ [π n]› **using** *cs-sorted-list-of-cd′* **by** *simp*
  also
  have ‹. . . = *map* π (*sorted-list-of-set* ({i. n $cd^{\pi}\rightarrow$ i ∧ i < k} ∪ {i. n $cd^{\pi}\rightarrow$ i ∧ k ≤ i})) @ [π n]› **using** *1* **by** *metis*

also
have ‹$\ldots = map\ \pi\ ((sorted\text{-}list\text{-}of\text{-}set\ \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge i < k\})\ @\ (sorted\text{-}list\text{-}of\text{-}set\ \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge k \leq i\}))\ @\ [\pi\ n]$›
using *sorted-list-of-set-append*[*OF - - le*] *is-cdi-def* **by** *auto*
also
have ‹$\ldots = (map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge i < k\}))\ @\ (map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge k \leq i\}))\ @\ [\pi\ n]$› **by** *auto*
also
have ‹$\ldots = cs^{\pi}\ j\ @\ (map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge k \leq i\}))\ @\ [\pi\ n]$› **unfolding** *2* **using** *cs-sorted-list-of-cd* **by** *auto*
also
have ‹$\ldots = cs^{\pi}\ j\ @\ cs^{\pi «k}\ (n-k)$› **using** *cs-path-shift-set-cd*[*OF path, of* ‹k› ‹$n-k$›] *assms*(*3*) **by** *auto*
finally
show ‹*?thesis*› **.**
qed

lemma *cs-split-shift-nocd*: **assumes** ‹$is\text{-}path\ \pi$› **and** ‹$k < n$› **and** ‹$\forall j.\ n\ cd^{\pi}\rightarrow\ j \longrightarrow k \leq j$› **shows** ‹$cs^{\pi}\ n = cs^{\pi «k}\ (n-k)$›
proof −
have *path*: ‹$is\text{-}path\ \pi$› **using** *assms* **by** *auto*
have *1*: ‹$\{i.\ n\ cd^{\pi}\rightarrow\ i\} = \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge i < k\} \cup \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge k \leq i\}$› **by** *auto*
have *le*: ‹$\forall\ i\in \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge i < k\}.\ \forall\ j\in \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge k \leq i\}.\ i < j$› **by** *auto*
have *2*: ‹$\{i.\ n\ cd^{\pi}\rightarrow\ i \wedge i < k\} = \{\}$› **using** *assms* **by** *auto*

have ‹$cs^{\pi}\ n = map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ \{i.\ n\ cd^{\pi}\rightarrow\ i\})\ @\ [\pi\ n]$› **using** *cs-sorted-list-of-cd′* **by** *simp*
also
have ‹$\ldots = map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ (\{i.\ n\ cd^{\pi}\rightarrow\ i \wedge i < k\} \cup \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge k \leq i\}))\ @\ [\pi\ n]$› **using** *1* **by** *metis*
also
have ‹$\ldots = map\ \pi\ (sorted\text{-}list\text{-}of\text{-}set\ \{i.\ n\ cd^{\pi}\rightarrow\ i \wedge k \leq i\})\ @\ [\pi\ n]$›
unfolding *2* **by** *auto*
also
have ‹$\ldots = cs^{\pi «k}\ (n-k)$› **using** *cs-path-shift-set-cd*[*OF path, of* ‹k› ‹$n-k$›] *assms*(*2*) **by** *auto*
finally show ‹*?thesis*› **.**
qed

lemma *shifted-cs-eq-is-eq*: **assumes** ‹$is\text{-}path\ \pi$› **and** ‹$is\text{-}path\ \pi'$› **and** ‹$cs^{\pi}\ k = cs^{\pi'}\ k'$› **and** ‹$cs^{\pi «k}\ n = cs^{\pi' «k'}\ n'$› **shows** ‹$cs^{\pi}\ (k+n) = cs^{\pi'}\ (k'+n')$›
proof (*rule ccontr*)
note $path = assms(1,2)$
note $csk = assms(3)$
note $csn = assms(4)$
assume *ne*: ‹$cs^{\pi}\ (k+n) \neq cs^{\pi'}\ (k'+n')$›
have *nretkn*:‹$\pi\ (k+n) \neq return$› **proof**
assume *1*:‹$\pi\ (k+n) = return$›
hence ‹$(\pi «k)\ n = return$› **by** *auto*
hence ‹$(\pi' «k')\ n' = return$› **using** *last-cs assms*(*4*) **by** *metis*
hence ‹$\pi'\ (k' + n') = return$› **by** *auto*
thus ‹*False*› **using** *ne 1 cs-return* **by** *auto*
qed
hence *nretk*: ‹$\pi\ k \neq return$› **using** *term-path-stable*[*OF assms*(*1*), *of* ‹k› ‹$k + n$›] **by** *auto*
have *nretkn′*: ‹$\pi'\ (k'+n') \neq return$› **proof**
assume *1*:‹$\pi'\ (k'+n') = return$›
hence ‹$(\pi' «k')\ n' = return$› **by** *auto*
hence ‹$(\pi «k)\ n = return$› **using** *last-cs assms*(*4*) **by** *metis*

hence ‹π $(k + n)$ = *return*› **by** *auto*
thus ‹*False*› **using** *ne 1 cs-return* **by** *auto*
qed
hence *nretk′*: ‹π' k' $\neq$ *return*› **using** *term-path-stable*[*OF assms*(*2*), *of* ‹k'› ‹k' +n'›] **by** *auto*
have *n0*:‹n > *0*› **proof** (*rule ccontr*)
assume ∗: ‹¬ *0* < n›
hence *1*:‹$cs^{\pi«k}$ *0* = $cs^{\pi'«k'}$ n'› **using** *assms*(*3*,*4*) **by** *auto*
have ‹(π«k) *0* = (π'«k') *0*› **using** *assms*(*3*) *last-cs path-shift-def* **by** (*metis monoid-add-class.add.right-neutral*)
hence ‹$cs^{\pi'«k'}$ *0* = $cs^{\pi'«k'}$ n'› **using** *1 cs-0* **by** *metis*
hence *n0′*: ‹n' = *0*› **using** *cs-inj*[*of* ‹π'«k'› ‹*0*› ‹n'›] ∗ *assms*(*2*) **by** (*metis path-shift-def assms*(*4*) *last-cs nretkn path-path-shift*)
thus ‹*False*› **using** *ne* ∗ *assms*(*3*) **by** *fastforce*
qed
have *n0′*:‹n' > *0*› **proof** (*rule ccontr*)
assume ∗: ‹¬ *0* < n'›
hence *1*:‹$cs^{\pi'«k'}$ *0* = $cs^{\pi«k}$ n› **using** *assms*(*3*,*4*) **by** *auto*
have ‹(π'«k') *0* = (π«k) *0*› **using** *assms*(*3*) *last-cs path-shift-def* **by** (*metis monoid-add-class.add.right-neutral*)
hence ‹$cs^{\pi«k}$ *0* = $cs^{\pi«k}$ n› **using** *1 cs-0* **by** *metis*
hence *n0*: ‹n = *0*› **using** *cs-inj*[*of* ‹π«k› ‹*0*› ‹n›] ∗ *assms*(*1*) **by** (*metis path-shift-def assms*(*4*) *last-cs nretkn path-path-shift*)
thus ‹*False*› **using** *ne* ∗ *assms*(*3*) **by** *fastforce*
qed
have *cdleswap′*: ‹∀ j'<k'. (k'+n') $cd^{\pi'}$→ j' ⟶ (∃j<k. (k+n) cd^{π}→ j ∧ cs^{π} j = $cs^{\pi'}$ j')› **proof** (*rule*,*rule*,*rule*, *rule ccontr*)
fix j' **assume** *jk′*: ‹j'<k'› **and** *ncdj′*: ‹(k'+n') $cd^{\pi'}$→ j'› **and** *ne*: ‹¬ (∃j<k. k + n cd^{π}→ j ∧ cs^{π} j = $cs^{\pi'}$ j')›
hence *kcdj′*: ‹k' $cd^{\pi'}$→ j'› **using** *cr-wn′* **by** *blast*

then obtain j **where** *kcdj*: ‹k cd^{π}→ j› **and** *csj*: ‹cs^{π} j = $cs^{\pi'}$ j'› **using** *csk cs-path-swap-cd path* **by** *metis*
hence *jk*: ‹j < k› **unfolding** *is-cdi-def* **by** *auto*

have *ncdn*: ‹¬ (k+n) cd^{π}→ j› **using** *ne csj jk* **by** *blast*

obtain l' **where** *lnocd′*: ‹l' = n' ∨ n' $cd^{\pi'«k'}$→ l'› **and** *cslsing′*: ‹$cs^{\pi'«k'}$ l' = [(π'«k') l']›
proof *cases*
assume ‹$cs^{\pi'«k'}$ n' = [(π'«k') n']› **thus** ‹*thesis*› **using** *that*[*of* ‹n'›] **by** *auto*
next
assume ∗: ‹$cs^{\pi'«k'}$ n' ≠ [(π'«k') n']›
then obtain x ys **where** ‹$cs^{\pi'«k'}$ n' = [x]@ys@[(π'«k') n']› **by** (*metis append-Cons append-Nil cs-length-g-one cs-length-one*(*1*) *neq-Nil-conv*)
then obtain l' **where** ‹$cs^{\pi'«k'}$ l' = [x]› **and** *cdl′*: ‹n' $cd^{\pi'«k'}$→ l'› **using** *cs-split*[*of* ‹π'«k'› ‹n'› ‹*Nil*› ‹x› ‹ys›] **by** *auto*
hence ‹$cs^{\pi'«k'}$ l' = [(π'«k') l']› **using** *last-cs* **by** (*metis last.simps*)
thus ‹*thesis*› **using** *that cdl′* **by** *auto*
qed
hence *ln′*: ‹l'≤n'› **unfolding** *is-cdi-def* **by** *auto*
hence *lcdj′*: ‹k'+l' $cd^{\pi'}$→ j'› **using** *jk′ ncdj′* **by** (*metis add-le-cancel-left cdi-prefix trans-less-add1*)

obtain l **where** *lnocd*: ‹l = n ∨ n $cd^{\pi«k}$→ l› **and** *csl*: ‹$cs^{\pi«k}$ l = $cs^{\pi'«k'}$ l'› **using** *lnocd′* **proof**
assume ‹l' = n'› **thus** ‹*thesis*› **using** *csn that*[*of* ‹n›] **by** *auto*
next
assume ‹n' $cd^{\pi'«k'}$→ l'›
then obtain l **where** ‹n $cd^{\pi«k}$→ l› ‹$cs^{\pi«k}$ l = $cs^{\pi'«k'}$ l'› **using** *cs-path-swap-cd path csn* **by** (*metis*

path-path-shift)
thus ‹*thesis*› **using** *that* **by** *auto*
qed

have *cslsing*: ‹$cs^{\pi «k}$ l = [(π«k) l]› **using** *cslsing′ last-cs csl last.simps* **by** *metis*

have *ln*: ‹$l \leq n$› **using** *lnocd* **unfolding** *is-cdi-def* **by** *auto*
hence *nretkl*: ‹$\pi\ (k + l) \neq return$› **using** *term-path-stable*[*of* ‹π› ‹k+l› ‹k+n›] *nretkn path*(*1*) **by** *auto*

have ∗: ‹$n\ cd^{\pi «k}\rightarrow l \Longrightarrow k{+}n\ cd^{\pi}\rightarrow k{+}l$› **using** *cd-path-shift*[*of* ‹k› ‹k+l› ‹π› ‹k+n›] *path*(*1*) **by** *auto*

have *ncdl*: ‹$\neg\ (k{+}l)\ cd^{\pi}\rightarrow j$› **apply** *rule* **using** *lnocd* **apply** *rule* **using** *ncdn* **apply** *blast* **using** *cd-trans ncdn* ∗ **by** *blast*

hence ‹$\exists\ i\in \{j..k{+}l\}.\ \pi\ i = ipd\ (\pi\ j)$› **unfolding** *is-cdi-def* **using** *path*(*1*) *jk nretkl* **by** *auto*
hence ‹$\exists\ i\in \{k{<}..k{+}l\}.\ \pi\ i = ipd\ (\pi\ j)$› **using** *kcdj* **unfolding** *is-cdi-def* **by** *force*

then obtain *i* **where** *ki*: ‹$k < i$› **and** *il*: ‹$i \leq k{+}l$› **and** *ipdi*: ‹$\pi\ i = ipd\ (\pi\ j)$› **by** *force*

hence ‹$(\pi«k)\ (i{-}k) = ipd\ (\pi\ j)$› ‹$i{-}k \leq l$› **by** *auto*
hence *pd*: ‹$(\pi«k)\ l\ pd\rightarrow ipd\ (\pi\ j)$› **using** *cs-single-pd-intermed*[*OF* - *cslsing*] *path*(*1*) *path-path-shift* **by** *metis*
moreover
have ‹$(\pi«k)\ l = \pi'\ (k' + l')$› **using** *csl last-cs* **by** (*metis path-shift-def*)
moreover
have ‹$\pi\ j = \pi'\ j'$› **using** *csj last-cs* **by** *metis*
ultimately
have ‹$\pi'\ (k'{+}l')\ pd\rightarrow ipd\ (\pi'\ j')$› **by** *simp*
moreover
have ‹$ipd\ (\pi'\ j')\ pd\rightarrow \pi'\ (k'{+}l')$› **using** *ipd-pd-cd*[*OF lcdj′*] **.**
ultimately
have ‹$\pi'\ (k'{+}l') = ipd\ (\pi'\ j')$› **using** *pd-antisym* **by** *auto*
thus ‹*False*› **using** *lcdj′* **unfolding** *is-cdi-def* **by** *force*
qed

— Symmetric version of the above statement
have *cdleswap*: ‹$\forall\ j{<}k.\ (k{+}n)\ cd^{\pi}\rightarrow j \longrightarrow (\exists j'{<}k'.\ (k'{+}n')\ cd^{\pi'}\rightarrow j' \wedge cs^{\pi}\ j = cs^{\pi'}\ j')$› **proof** (*rule*,*rule*,*rule*, *rule ccontr*)
fix *j* **assume** *jk*: ‹$j{<}k$› **and** *ncdj*: ‹$(k{+}n)\ cd^{\pi}\rightarrow j$› **and** *ne*: ‹$\neg\ (\exists j'{<}k'.\ k' + n'\ cd^{\pi'}\rightarrow j' \wedge cs^{\pi}\ j = cs^{\pi'}\ j')$›
hence *kcdj*: ‹$k\ cd^{\pi}\rightarrow j$› **using** *cr-wn′* **by** *blast*

then obtain *j′* **where** *kcdj′*: ‹$k'\ cd^{\pi'}\rightarrow j'$› **and** *csj*: ‹$cs^{\pi}\ j = cs^{\pi'}\ j'$› **using** *csk cs-path-swap-cd path* **by** *metis*
hence *jk′*: ‹$j' < k'$› **unfolding** *is-cdi-def* **by** *auto*

have *ncdn′*: ‹$\neg\ (k'{+}n')\ cd^{\pi'}\rightarrow j'$› **using** *ne csj jk′* **by** *blast*

obtain *l* **where** *lnocd*: ‹$l = n \vee n\ cd^{\pi «k}\rightarrow l$› **and** *cslsing*: ‹$cs^{\pi «k}\ l = [(\pi«k)\ l]$›
proof *cases*
assume ‹$cs^{\pi «k}\ n = [(\pi«k)\ n]$› **thus** ‹*thesis*› **using** *that*[*of* ‹n›] **by** *auto*
next
assume ∗: ‹$cs^{\pi «k}\ n \neq [(\pi«k)\ n]$›
then obtain *x ys* **where** ‹$cs^{\pi «k}\ n = [x]@ys@[(\pi«k)\ n]$› **by** (*metis append-Cons append-Nil cs-length-g-one cs-length-one*(*1*) *neq-Nil-conv*)

then obtain *l* **where** ‹$cs^{\pi«k}$ $l = [x]$› **and** *cdl*: ‹$n\ cd^{\pi«k}\!\!\rightarrow l$› **using** *cs-split*[*of* ‹π«k› ‹n› ‹Nil› ‹x› ‹ys›] **by** *auto*
hence ‹$cs^{\pi«k}$ $l = [(\pi«k)\ l]$› **using** *last-cs* **by** (*metis last.simps*)
thus ‹*thesis*› **using** *that cdl* **by** *auto*
qed
hence *ln*: ‹$l \le n$› **unfolding** *is-cdi-def* **by** *auto*
hence *lcdj*: ‹$k+l\ cd^{\pi}\!\!\rightarrow j$› **using** *jk ncdj* **by** (*metis add-le-cancel-left cdi-prefix trans-less-add1*)

obtain l' **where** *lnocd'*: ‹$l' = n' \lor n'\ cd^{\pi'«k'}\!\!\rightarrow l'$› **and** *csl*: ‹$cs^{\pi«k}$ $l = cs^{\pi'«k'}$ l'› **using** *lnocd* **proof**
assume ‹$l = n$› **thus** ‹*thesis*› **using** *csn that*[*of* ‹n'›] **by** *auto*
next
assume ‹$n\ cd^{\pi«k}\!\!\rightarrow l$›
then obtain l' **where** ‹$n'\ cd^{\pi'«k'}\!\!\rightarrow l'$› ‹$cs^{\pi«k}$ $l = cs^{\pi'«k'}$ l'› **using** *cs-path-swap-cd path csn* **by** (*metis path-path-shift*)
thus ‹*thesis*› **using** *that* **by** *auto*
qed

have *cslsing'*: ‹$cs^{\pi'«k'}$ $l' = [(\pi'«k')\ l']$› **using** *cslsing last-cs csl last.simps* **by** *metis*

have *ln'*: ‹$l' \le n'$› **using** *lnocd'* **unfolding** *is-cdi-def* **by** *auto*
hence *nretkl'*: ‹$\pi'\ (k' + l') \neq return$› **using** *term-path-stable*[*of* ‹π'› ‹k'+l'› ‹k'+n'›] *nretkn' path*(*2*) **by** *auto*

have ∗: ‹$n'\ cd^{\pi'«k'}\!\!\rightarrow l' \Longrightarrow k'+n'\ cd^{\pi'}\!\!\rightarrow k'+l'$› **using** *cd-path-shift*[*of* ‹k'› ‹k'+l'› ‹π'› ‹k'+n'›] *path*(*2*) **by** *auto*

have *ncdl'*: ‹$\neg\ (k'+l')\ cd^{\pi'}\!\!\rightarrow j'$› **apply** *rule* **using** *lnocd'* **apply** *rule* **using** *ncdn'* **apply** *blast* **using** *cd-trans ncdn'* ∗ **by** *blast*

hence ‹$\exists\ i' \in \{j'..k'+l'\}.\ \pi'\ i' = ipd\ (\pi'\ j')$› **unfolding** *is-cdi-def* **using** *path*(*2*) *jk' nretkl'* **by** *auto*
hence ‹$\exists\ i' \in \{k'<..k'+l'\}.\ \pi'\ i' = ipd\ (\pi'\ j')$› **using** *kcdj'* **unfolding** *is-cdi-def* **by** *force*

then obtain i' **where** *ki'*: ‹$k' < i'$› **and** *il'*: ‹$i' \le k'+l'$› **and** *ipdi*: ‹$\pi'\ i' = ipd\ (\pi'\ j')$› **by** *force*

hence ‹$(\pi'«k')\ (i'-k') = ipd\ (\pi'\ j')$› ‹$i'-k' \le l'$› **by** *auto*
hence *pd*: ‹$(\pi'«k')\ l'\ pd\!\!\rightarrow ipd\ (\pi'\ j')$› **using** *cs-single-pd-intermed*[*OF - cslsing'*] *path*(*2*) *path-path-shift* **by** *metis*
moreover
have ‹$(\pi'«k')\ l' = \pi\ (k + l)$› **using** *csl last-cs* **by** (*metis path-shift-def*)
moreover
have ‹$\pi'\ j' = \pi\ j$› **using** *csj last-cs* **by** *metis*
ultimately
have ‹$\pi\ (k+l)\ pd\!\!\rightarrow ipd\ (\pi\ j)$› **by** *simp*
moreover
have ‹$ipd\ (\pi\ j)\ pd\!\!\rightarrow \pi\ (k+l)$› **using** *ipd-pd-cd*[*OF lcdj*] **.**
ultimately
have ‹$\pi\ (k+l) = ipd\ (\pi\ j)$› **using** *pd-antisym* **by** *auto*
thus ‹*False*› **using** *lcdj* **unfolding** *is-cdi-def* **by** *force*
qed

have *cdle*: ‹$\exists j.\ (k+n)\ cd^{\pi}\!\!\rightarrow j \land j < k$› (**is** ‹$\exists\ j.\ ?P\ j$›) **proof** (*rule ccontr*)
assume ‹$\neg\ (\exists j.\ (k+n)\ cd^{\pi}\!\!\rightarrow j \land j < k)$›
hence *allge*: ‹$\forall j.\ (k+n)\ cd^{\pi}\!\!\rightarrow j \longrightarrow k \le j$› **by** *auto*
have *allge'*: ‹$\forall j'.\ (k'+n')\ cd^{\pi'}\!\!\rightarrow j' \longrightarrow k' \le j'$› **proof** (*rule, rule, rule ccontr*)
fix j'

assume $*$: $\langle k' + n'\ cd^{\pi'} \rightarrow j' \rangle$ **and** $\langle \neg\ k' \leq j' \rangle$
then obtain j **where** $\langle j<k \rangle$ $\langle (k+n)\ cd^{\pi} \rightarrow j \rangle$ **using** *cdleswap'* **by** (*metis le-neq-implies-less nat-le-linear*)
thus $\langle$*False*$\rangle$ **using** *allge* **by** *auto*
qed
have $\langle cs^{\pi}\ (k + n) = cs^{\pi \ll k}\ n \rangle$ **using** *cs-split-shift-nocd*[*OF assms*(*1*) - *allge*] *n0* **by** *auto*
moreover
have $\langle cs^{\pi'}\ (k' + n') = cs^{\pi' \ll k'}\ n' \rangle$ **using** *cs-split-shift-nocd*[*OF assms*(*2*) - *allge'*] *n0'* **by** *auto*
ultimately
show $\langle$*False*$\rangle$ **using** *ne assms*(*4*) **by** *auto*
qed

define j **where** $\langle j == GREATEST\ j.\ (k+n)\ cd^{\pi} \rightarrow j \wedge j < k \rangle$
have *cdj*:$\langle (k+n)\ cd^{\pi} \rightarrow j \rangle$ **and** *jk*: $\langle j < k \rangle$ **and** *jge*:$\langle \forall\ j' < k.\ (k+n)\ cd^{\pi} \rightarrow j' \longrightarrow j' \leq j \rangle$ **proof** $-$
have *bound*: $\langle \forall\ y.\ ?P\ y \longrightarrow y \leq k \rangle$ **by** *auto*
show $\langle (k+n)\ cd^{\pi} \rightarrow j \rangle$ **using** *GreatestI-nat*[*of* $\langle ?P \rangle$] *j-def bound cdle* **by** *blast*
show $\langle j < k \rangle$ **using** *GreatestI-nat*[*of* $\langle ?P \rangle$] *bound j-def cdle* **by** *blast*
show $\langle \forall\ j' < k.\ (k+n)\ cd^{\pi} \rightarrow j' \longrightarrow j' \leq j \rangle$ **using** *Greatest-le-nat*[*of* $\langle ?P \rangle$] *bound j-def* **by** *blast*
qed

obtain j' **where** *cdj'*:$\langle (k'+n')\ cd^{\pi'} \rightarrow j' \rangle$ **and** *csj*: $\langle cs^{\pi}\ j = cs^{\pi'}\ j' \rangle$ **and** *jk'*: $\langle j' < k' \rangle$ **using** *cdleswap cdj jk* **by** *blast*
have *jge'*:$\langle \forall\ i' < k'.\ (k'+n')\ cd^{\pi'} \rightarrow i' \longrightarrow i' \leq j' \rangle$ **proof**(*rule*,*rule*,*rule*)
fix i'
assume *ik'*: $\langle i' < k' \rangle$ **and** *cdi'*: $\langle k' + n'\ cd^{\pi'} \rightarrow i' \rangle$
then obtain i **where** *cdi*:$\langle (k+n)\ cd^{\pi} \rightarrow i \rangle$ **and** *csi*: $\langle\ cs^{\pi'}\ i' = cs^{\pi}\ i \rangle$ **and** *ik*: $\langle i<k \rangle$ **using** *cdleswap'* **by** *force*
have *ij*: $\langle i \leq j \rangle$ **using** *jge cdi ik* **by** *auto*
show $\langle i' \leq j' \rangle$ **using** *cs-order-le*[*OF assms*(*1*,*2*) *csi*[*symmetric*] *csj* - *ij*] *cd-not-ret*[*OF cdi*] **by** *simp*
qed
have $\langle cs^{\pi}\ (k + n) = cs^{\pi}\ j\ @\ cs^{\pi \ll k}\ n \rangle$ **using** *cs-split-shift-cd*[*OF cdj jk* - *jge*] *n0* **by** *auto*
moreover
have $\langle cs^{\pi'}\ (k' + n') = cs^{\pi'}\ j'\ @\ cs^{\pi' \ll k'}\ n' \rangle$ **using** *cs-split-shift-cd*[*OF cdj' jk'* - *jge'*] *n0'* **by** *auto*
ultimately
have $\langle cs^{\pi}\ (k+n) = cs^{\pi'}\ (k'+n') \rangle$ **using** *csj assms*(*4*) **by** *auto*
thus $\langle$*False*$\rangle$ **using** *ne* **by** *simp*
qed

lemma *cs-eq-is-eq-shifted*: **assumes** $\langle$*is-path* $\pi\rangle$ **and** $\langle$*is-path* $\pi'\rangle$ **and** $\langle cs^{\pi}\ k = cs^{\pi'}\ k' \rangle$ **and** $\langle cs^{\pi}\ (k+n) = cs^{\pi'}\ (k'+n') \rangle$ **shows** $\langle cs^{\pi \ll k}\ n = cs^{\pi' \ll k'}\ n' \rangle$
proof (*rule ccontr*)
assume *ne*: $\langle cs^{\pi \ll k}\ n \neq cs^{\pi' \ll k'}\ n' \rangle$
have *nretkn*:$\langle \pi\ (k+n) \neq return \rangle$ **proof**
assume *1*:$\langle \pi\ (k+n) = return \rangle$
hence *2*:$\langle \pi'\ (k'+n') = return \rangle$ **using** *assms*(*4*) *last-cs* **by** *metis*
hence $\langle (\pi \ll k)\ n = return \rangle$ $\langle (\pi' \ll k')\ n' = return \rangle$ **using** *1* **by** *auto*
hence $\langle cs^{\pi \ll k}\ n = cs^{\pi' \ll k'}\ n' \rangle$ **using** *cs-return* **by** *metis*
thus $\langle$*False*$\rangle$ **using** *ne* **by** *simp*
qed
hence *nretk*: $\langle \pi\ k \neq return \rangle$ **using** *term-path-stable*[*OF assms*(*1*), *of* $\langle k \rangle$ $\langle k + n \rangle$] **by** *auto*
have *nretkn'*: $\langle \pi'\ (k'+n') \neq return \rangle$ **proof**
assume *1*:$\langle \pi'\ (k'+n') = return \rangle$
hence *2*:$\langle \pi\ (k+n) = return \rangle$ **using** *assms*(*4*) *last-cs* **by** *metis*
hence $\langle (\pi \ll k)\ n = return \rangle$ $\langle (\pi' \ll k')\ n' = return \rangle$ **using** *1* **by** *auto*
hence $\langle cs^{\pi \ll k}\ n = cs^{\pi' \ll k'}\ n' \rangle$ **using** *cs-return* **by** *metis*
thus $\langle$*False*$\rangle$ **using** *ne* **by** *simp*

qed
hence *nretk'*: ‹π' $k' \neq$ *return*› **using** *term-path-stable*[*OF assms*(*2*), *of* ‹k'› ‹k' +n'›] **by** *auto*
have *n0*:‹$n > 0$› **proof** (*rule ccontr*)
assume ∗: ‹$\neg$ $0 < n$›
hence ‹$cs^{\pi'}$ $k' = cs^{\pi'}$ (k'+ n')› **using** *assms*(*3*,*4*) **by** *auto*
hence *n0*: ‹$n = 0$› ‹$n' = 0$› **using** *cs-inj*[*OF assms*(*2*) *nretkn'*, *of* ‹k'›] ∗ **by** *auto*
have ‹cs^{π} « ^{k} $n = cs^{\pi'}$ « $^{k'}$ n'› **unfolding** *n0 cs-0* **by** (*auto* , *metis last-cs assms*(*3*))
thus ‹*False*› **using** *ne* **by** *simp*
qed
have *n0'*:‹$n' > 0$› **proof** (*rule ccontr*)
assume ∗: ‹$\neg$ $0 < n'$›
hence ‹cs^{π} $k = cs^{\pi}$ (k+ n)› **using** *assms*(*3*,*4*) **by** *auto*
hence *n0*: ‹$n = 0$› ‹$n' = 0$› **using** *cs-inj*[*OF assms*(*1*) *nretkn*, *of* ‹k›] ∗ **by** *auto*
have ‹cs^{π} « ^{k} $n = cs^{\pi'}$ « $^{k'}$ n'› **unfolding** *n0 cs-0* **by** (*auto* , *metis last-cs assms*(*3*))
thus ‹*False*› **using** *ne* **by** *simp*
qed
have *cdle*: ‹∃*j*. (k+n) cd^{π}→ $j \wedge j < k$› (**is** ‹∃ *j*. *?P j*›) **proof** (*rule ccontr*)
assume ‹¬ (∃*j*. (k+n) cd^{π}→ $j \wedge j < k$)›
hence *allge*: ‹∀*j*. (k+n) cd^{π}→ $j \longrightarrow k \leq j$› **by** *auto*
have *allge'*: ‹∀j'. (k'+n') $cd^{\pi'}$→ $j' \longrightarrow k' \leq j'$› **proof** (*rule*, *rule*)
fix j'
assume ∗: ‹k' + n' $cd^{\pi'}$→ j'›
obtain *j* **where** *cdj*: ‹k+n cd^{π}→ j› **and** *csj*: ‹cs^{π} $j = cs^{\pi'}$ j'› **using** *cs-path-swap-cd*[*OF assms*(*2*,*1*) *assms*(*4*)[*symmetric*] ∗] **by** *metis*
hence *kj*:‹$k \leq j$› **using** *allge* **by** *auto*
thus *kj'*: ‹$k' \leq j'$› **using** *cs-order-le*[*OF assms*(*1*,*2*,*3*) *csj nretk*] **by** *simp*
qed
have ‹cs^{π} (k + n) = cs^{π} « ^{k} n› **using** *cs-split-shift-nocd*[*OF assms*(*1*) - *allge*] *n0* **by** *auto*
moreover
have ‹$cs^{\pi'}$ (k' + n') = $cs^{\pi'}$ « $^{k'}$ n'› **using** *cs-split-shift-nocd*[*OF assms*(*2*) - *allge'*] *n0'* **by** *auto*
ultimately
show ‹*False*› **using** *ne assms*(*4*) **by** *auto*
qed
define *j* **where** ‹*j* == *GREATEST j*. (k+n) cd^{π}→ $j \wedge j < k$›
have *cdj*:‹(k+n) cd^{π}→ j› **and** *jk*: ‹$j < k$› **and** *jge*:‹∀ j'< *k*. (k+n) cd^{π}→ $j' \longrightarrow j' \leq j$› **proof** −
have *bound*: ‹∀ *y*. *?P y* ⟶ $y \leq k$› **by** *auto*
show ‹(k+n) cd^{π}→ j› **using** *GreatestI-nat*[*of* ‹*?P*›] *bound j-def cdle* **by** *blast*
show ‹$j < k$› **using** *GreatestI-nat*[*of* ‹*?P*›] *bound j-def cdle* **by** *blast*
show ‹∀ j'< *k*. (k+n) cd^{π}→ $j' \longrightarrow j' \leq j$› **using** *Greatest-le-nat*[*of* ‹*?P*›] *bound j-def* **by** *blast*
qed
obtain j' **where** *cdj'*:‹(k'+n') $cd^{\pi'}$→ j'› **and** *csj*: ‹cs^{π} $j = cs^{\pi'}$ j'› **using** *cs-path-swap-cd assms cdj* **by** *blast*
have *jge'*:‹∀ i'< k'. (k'+n') $cd^{\pi'}$→ $i' \longrightarrow i' \leq j'$› **proof**(*rule*,*rule*,*rule*)
fix i'
assume *ik'*: ‹$i' < k'$› **and** *cdi'*: ‹k' + n' $cd^{\pi'}$→ i'›
then obtain *i* **where** *cdi*:‹(k+n) cd^{π}→ i› **and** *csi*: ‹ $cs^{\pi'}$ $i' = cs^{\pi}$ i› **using** *cs-path-swap-cd*[*OF assms*(*2*,*1*) *assms*(*4*)[*symmetric*]] **by** *blast*
have *nreti'*: ‹π' $i' \neq$ *return*› **by** (*metis cd-not-ret cdi'*)
have *ik*: ‹$i < k$› **using** *cs-order*[*OF assms*(*2*,*1*) *csi* - *nreti' ik'*] *assms*(*3*) **by** *auto*
have *ij*: ‹$i \leq j$› **using** *jge cdi ik* **by** *auto*
show ‹$i' \leq j'$› **using** *cs-order-le*[*OF assms*(*1*,*2*) *csi*[*symmetric*] *csj* - *ij*] *cd-not-ret*[*OF cdi*] **by** *simp*
qed
have *jk'*: ‹$j' < k'$› **using** *cs-order*[*OF assms*(*1*,*2*) *csj assms*(*3*) *cd-not-ret*[*OF cdj*] *jk*] **.**
have ‹cs^{π} (k + n) = cs^{π} j @ cs^{π} « ^{k} n› **using** *cs-split-shift-cd*[*OF cdj jk* - *jge*] *n0* **by** *auto*
moreover

have ‹$cs^{\pi'}$ $(k' + n') = cs^{\pi'}$ j' @ $cs^{\pi'}$ « k' n'› **using** *cs-split-shift-cd*[*OF cdj' jk' - jge'*] *n0'* **by** *auto*
ultimately
have ‹$cs^{\pi«k}$ $n = cs^{\pi'«k'}$ n'› **using** *csj assms*(*4*) **by** *auto*
thus ‹*False*› **using** *ne* **by** *simp*
qed

lemma *converged-cd-diverge-cs*: **assumes** ‹*is-path* π› **and** ‹*is-path* π'› **and** ‹cs^{π} j $= cs^{\pi'}$ j'› **and** ‹$j<l$› **and** ‹$\neg$ ($\exists$ l'. cs^{π} $l = cs^{\pi'}$ l')› **and** ‹$l < m$› **and** ‹cs^{π} $m = cs^{\pi'}$ m'›
obtains k k' **where** ‹$j \leq k$› ‹cs^{π} $k = cs^{\pi'}$ k'› **and** ‹l $cd^{\pi}\rightarrow$ k› **and** ‹π (*Suc* k) $\neq \pi'$ (*Suc* k')›
proof –
have ‹*is-path* (π«j)› ‹*is-path* (π'«j')› **using** *assms*(*1*,*2*) *path-path-shift* **by** *auto*
moreover
have ‹(π«j) *0* = (π'«j') *0*› **using** *assms*(*3*) *last-cs* **by** (*metis path-shift-def add.right-neutral*)
moreover
have ‹$\neg$($\exists$ l'. $cs^{\pi«j}$ ($l-j$) $= cs^{\pi'«j'}$ l')› **proof**
assume ‹$\exists$ l'. cs^{π} « ^{j} ($l - j$) $= cs^{\pi'}$ « $^{j'}$ l'›
then obtain l' **where** *csl*: ‹$cs^{\pi«j}$ ($l - j$) $= cs^{\pi'«j'}$ l'› **by** *blast*

have ‹cs^{π} $l = cs^{\pi'}$ ($j' + l'$)› **using** *shifted-cs-eq-is-eq*[*OF assms*(*1*,*2*,*3*) *csl*] *assms*(*4*) **by** *auto*
thus ‹*False*› **using** *assms*(*5*) **by** *blast*
qed
moreover
have ‹$l-j < m-j$› **using** *assms* **by** *auto*
moreover
have ‹π $j \neq$ *return*› **using** *cs-return assms*(*1*–*5*) *term-path-stable* **by** (*metis nat-less-le*)
hence ‹$j'<m'$› **using** *cs-order*[*OF assms*(*1*,*2*,*3*,*7*)] *assms* **by** *auto*
hence ‹$cs^{\pi«j}$ ($m-j$) $= cs^{\pi'«j'}$ ($m'-j'$)› **using** *cs-eq-is-eq-shifted*[*OF assms*(*1*,*2*,*3*),*of* ‹$m-j$› ‹$m'-j'$›] *assms*(*4*,*6*,*7*) **by** *auto*
ultimately
obtain k k' **where** *csk*: ‹$cs^{\pi«j}$ $k = cs^{\pi'«j'}$ k'› **and** *lcdk*: ‹$l-j$ $cd^{\pi«j}\rightarrow$ k› **and** *suc*:‹(π«j) (*Suc* k) $\neq$ (π'«j') (*Suc* k')› **using** *converged-cd-diverge* **by** *blast*

have ‹cs^{π} (j+k) $= cs^{\pi'}$ (j'+k')› **using** *shifted-cs-eq-is-eq*[*OF assms*(*1*–*3*) *csk*] **.**
moreover
have ‹l $cd^{\pi}\rightarrow$ j+k› **using** *lcdk assms*(*1*,*2*,*4*) **by** (*metis add.commute add-diff-cancel-right' cd-path-shift le-add1*)
moreover
have ‹π (*Suc* (j+k)) $\neq \pi'$ (*Suc* (j'+ k'))› **using** *suc* **by** *auto*
moreover
have ‹$j \leq j$+k› **by** *auto*
ultimately
show ‹*thesis*› **using** *that*[*of* ‹j+k› ‹j'+k'›] **by** *auto*
qed

lemma *cs-ipd-conv*: **assumes** *csk*: ‹cs^{π} $k = cs^{\pi'}$ k'› **and** *ipd*: ‹π l = *ipd* (π k)› ‹π' l' = *ipd*(π' k')›
and *nipd*: ‹$\forall$ $n \in \{k..<l\}$. π $n \neq$ *ipd* (π k)› ‹$\forall$ $n' \in \{k'..<l'\}$. π' $n' \neq$ *ipd* (π' k')› **and** *kl*: ‹$k < l$› ‹$k' < l'$›
shows ‹cs^{π} $l = cs^{\pi'}$ l'› **using** *cs-ipd*[*OF ipd*(*1*) *nipd*(*1*) *kl*(*1*)] *cs-ipd*[*OF ipd*(*2*) *nipd*(*2*) *kl*(*2*)] *csk ipd* **by** (*metis* (*no-types*) *last-cs*)

lemma *cp-eq-cs*: **assumes** ‹((σ,k),(σ',k'))$\in$*cp*› **shows** ‹$cs^{path\ \sigma}$ $k = cs^{path\ \sigma'}$ k'›
using *assms*
apply(*induction rule*: *cp.induct*)
apply *blast*+

apply *simp*
done

lemma *cd-cs-swap*: **assumes** ‹$l\ cd^{\pi}\rightarrow k$› ‹$cs^{\pi}\ l = cs^{\pi'}\ l'$› ‹$cs^{\pi}\ k = cs^{\pi'}\ k'$› **shows** ‹$l'\ cd^{\pi'}\rightarrow k'$› **proof** −
have ‹$\exists\ i.\ l\ icd^{\pi}\rightarrow i$› **using** *assms(1) excd-impl-exicd* **by** *blast*
hence ‹$cs^{\pi}\ l \neq [\pi\ l]$› **by** *auto*
hence ‹$cs^{\pi'}\ l' \neq [\pi'\ l']$› **using** *assms last-cs* **by** *metis*
hence ‹$\exists\ i'.\ l'\ icd^{\pi'}\rightarrow i'$› **by** (*metis cs-cases*)
hence *path'*: ‹*is-path* π'› **unfolding** *is-icdi-def is-cdi-def* **by** *auto*
from *cd-in-cs*[*OF assms(1)*]
obtain *ys* **where** *csl*: ‹$cs^{\pi}\ l = cs^{\pi}\ k$ @ ys @ $[\pi\ l]$› **by** *blast*
obtain *xs* **where** *csk*: ‹$cs^{\pi}\ k = xs$@$[\pi\ k]$› **by** (*metis append-butlast-last-id cs-not-nil last-cs*)
have πl: ‹$\pi\ l = \pi'\ l'$› **using** *assms last-cs* **by** *metis*
have *csl'*: ‹$cs^{\pi'}\ l' = xs$@$[\pi\ k]$@ys@$[\pi'\ l']$› **by** (*metis* πl *append-eq-appendI assms(2) csk csl*)
from *cs-split*[*of* ‹π'› ‹l'› ‹xs› ‹$\pi\ k$› ‹ys›]
obtain *m* **where** *csm*: ‹$cs^{\pi'}\ m = xs$ @ $[\pi\ k]$› **and** *lcdm*: ‹$l'\ cd^{\pi'}\rightarrow m$› **using** *csl'* **by** *metis*
have *csm'*: ‹$cs^{\pi'}\ m = cs^{\pi'}\ k'$› **by** (*metis assms(3) csk csm*)
have ‹$\pi'\ m \neq$ *return*› **using** *lcdm* **unfolding** *is-cdi-def* **using** *term-path-stable* **by** (*metis nat-less-le*)
hence ‹$m = k'$› **using** *cs-inj path' csm'* **by** *auto*
thus ‹*?thesis*› **using** *lcdm* **by** *auto*
qed

2.6 Facts about Observations

lemma *kth-obs-not-none*: **assumes** ‹*is-kth-obs* (*path* σ) k i› **obtains** a **where** ‹*obsp* σ i = *Some* a› **using** *assms* **unfolding** *is-kth-obs-def obsp-def* **by** *auto*

lemma *kth-obs-unique*: ‹*is-kth-obs* π k i $\Longrightarrow$ *is-kth-obs* π k j $\Longrightarrow$ $i = j$› **proof** (*induction* ‹i› ‹j› *rule*: *nat-sym-cases*)
case *sym* **thus** ‹*?case*› **by** *simp*
next
case *eq* **thus** ‹*?case*› **by** *simp*
next
case (*less i j*)
have ‹*obs-ids* $\pi \cap \{..<i\} \subseteq$ *obs-ids* $\pi \cap \{..<j\}$› **using** *less(1)* **by** *auto*
moreover
have ‹$i \in$ *obs-ids* $\pi \cap \{..<j\}$› **using** *less* **unfolding** *is-kth-obs-def obs-ids-def* **by** *auto*
moreover
have ‹$i \notin$ *obs-ids* $\pi \cap \{..<i\}$› **by** *auto*
moreover
have ‹*card* (*obs-ids* $\pi \cap \{..<i\}$) = *card* (*obs-ids* $\pi \cap \{..<j\}$)› **using** *less.prems* **unfolding** *is-kth-obs-def* **by** *auto*
moreover
have ‹*finite* (*obs-ids* $\pi \cap \{..<i\}$)› ‹*finite* (*obs-ids* $\pi \cap \{..<j\}$)› **by** *auto*
ultimately
have ‹*False*› **by** (*metis card-subset-eq*)
thus ‹*?case*› **..**
qed

lemma *obs-none-no-kth-obs*: **assumes** ‹*obs* σ k = *None*› **shows** ‹$\neg$ ($\exists$ i. *is-kth-obs* (*path* σ) k i)›
apply *rule*
using *assms*
unfolding *obs-def obsp-def*
apply (*auto split*: *option.split-asm*)
by (*metis assms kth-obs-not-none kth-obs-unique obs-def option.distinct(2) the-equality*)

lemma *obs-some-kth-obs* : **assumes** ‹*obs σ k ≠ None*› **obtains** *i* **where** ‹*is-kth-obs* (*path σ*) *k i*› **by** (*metis obs-def assms*)

lemma *not-none-is-obs*: **assumes** ‹*att*(*π i*) ≠ *None*› **shows** ‹*is-kth-obs π* (*card* (*obs-ids π* ∩ {..<*i*})) *i*› **unfolding** *is-kth-obs-def* **using** *assms* **by** *auto*

lemma *in-obs-ids-is-kth-obs*: **assumes** ‹*i* ∈ *obs-ids π*› **obtains** *k* **where** ‹*is-kth-obs π k i*› **proof**
  have ‹*att* (*π i*) ≠ *None*› **using** *assms obs-ids-def* **by** *auto*
  thus ‹*is-kth-obs π* (*card* (*obs-ids π* ∩ {..<*i*})) *i*› **using** *not-none-is-obs* **by** *auto*
qed

lemma *kth-obs-stable*: **assumes** ‹*is-kth-obs π l j*› ‹*k* < *l*› **shows** ‹∃ *i*. *is-kth-obs π k i*› **using** *assms* **proof** (*induction* ‹*l*› *arbitrary*: ‹*j*› *rule*: *less-induct*)
  case (*less l j*)
  have *cardl*: ‹*card* (*obs-ids π* ∩ {..<*j*}) = *l*› **using** *less is-kth-obs-def* **by** *auto*
  then obtain *i* **where** *ex*: ‹*i* ∈ *obs-ids π* ∩ {..<*j*}› (**is** ‹*?P i*›) **using** *less*(*3*) **by** (*metis card.empty empty-iff less-irrefl subsetI subset-antisym zero-diff zero-less-diff*)
  have *bound*: ‹∀ *i*. *i* ∈ *obs-ids π* ∩ {..<*j*} ⟶ *i* ≤ *j*› **by** *auto*
  let ‹*?i*› = ‹*GREATEST i*. *i* ∈ *obs-ids π* ∩ {..<*j*}›
  have ∗: ‹*?i* < *j*› ‹*?i* ∈ *obs-ids π*› **using** *GreatestI-nat*[*of* ‹*?P*› ‹*i*› ‹*j*›] *ex bound* **by** *auto*
  have ∗∗: ‹∀ *i*. *i* ∈ *obs-ids π* ∧ *i*<*j* ⟶ *i* ≤ *?i*› **using** *Greatest-le-nat*[*of* ‹*?P*› - ‹*j*›] *ex bound* **by** *auto*
  have ‹(*obs-ids π* ∩ {..<*?i*}) ∪ {*?i*} = *obs-ids π* ∩ {..<*j*}› **apply** *rule* **apply** *auto* **using** ∗[*simplified*] **apply** *simp+* **using** ∗∗[*simplified*] **by** *auto*
  moreover
  have ‹*?i* ∉ (*obs-ids π* ∩ {..<*?i*})› **by** *auto*
  ultimately
  have ‹*Suc* (*card* (*obs-ids π* ∩ {..<*?i*})) = *l*› **using** *cardl* **by** (*metis Un-empty-right Un-insert-right card-insert-disjoint finite-Int finite-lessThan*)
  hence ‹*card* (*obs-ids π* ∩ {..<*?i*}) = *l* − *1*› **by** *auto*
  hence *iko*: ‹*is-kth-obs π* (*l* − *1*) *?i*› **using** ∗(*2*) **unfolding** *is-kth-obs-def obs-ids-def* **by** *auto*
  have *ll*: ‹*l* − *1* < *l*› **by** (*metis One-nat-def diff-Suc-less less.prems*(*2*) *not-gr0 not-less0*)
  note *IV*=*less*(*1*)[*OF ll iko*]
  show ‹*?thesis*› **proof** *cases*
    assume ‹*k* < *l* − *1*› **thus** ‹*?thesis*› **using** *IV* **by** *simp*
  next
    assume ‹¬ *k* < *l* − *1*›
    hence ‹*k* = *l* − *1*› **using** *less* **by** *auto*
    thus ‹*?thesis*› **using** *iko* **by** *blast*
  qed
qed

lemma *kth-obs-mono*: **assumes** ‹*is-kth-obs π k i*› ‹*is-kth-obs π l j*› ‹*k* < *l*› **shows** ‹*i* < *j*› **proof** (*rule ccontr*)
  assume ‹¬ *i* < *j*›
  hence ‹{..<*j*} ⊆ {..<*i*}› **by** *auto*
  hence ‹*obs-ids π* ∩ {..<*j*} ⊆ *obs-ids π* ∩ {..<*i*}› **by** *auto*
  moreover
  have ‹*finite* (*obs-ids π* ∩ {..<*i*})› **by** *auto*
  ultimately
  have ‹*card* (*obs-ids π* ∩ {..<*j*}) ≤ *card* (*obs-ids π* ∩ {..<*i*})› **by** (*metis card-mono*)
  thus ‹*False*› **using** *assms* **unfolding** *is-kth-obs-def* **by** *auto*
qed

lemma *kth-obs-le-iff*: **assumes** ‹*is-kth-obs π k i*› ‹*is-kth-obs π l j*› **shows** ‹*k* < *l* ⟷ *i* < *j*› **by** (*metis assms kth-obs-unique kth-obs-mono not-less-iff-gr-or-eq*)

lemma *ret-obs-all-obs*: **assumes** *path*: ‹*is-path* π› **and** *iki*: ‹*is-kth-obs* π *k i*› **and** *ret*: ‹π *i* = *return*› **and** *kl*: ‹*k* < *l*› **obtains** *j* **where** ‹*is-kth-obs* π *l j*›
proof−
  show ‹*thesis*›
  using *kl iki ret* **proof** (*induction* ‹*l* − *k*› *arbitrary*: ‹*k*› ‹*i*› *rule*: *less-induct*)
    case (*less k i*)
    note *kl* = ‹*k* < *l*›
    note *iki* = ‹*is-kth-obs* π *k i*›
    note *ret* = ‹π *i* = *return*›
    have *card*: ‹*card* (*obs-ids* π ∩ {..<*i*}) = *k*› **and** *att-ret*: ‹*att return* ≠ *None*›**using** *iki ret* **unfolding** *is-kth-obs-def* **by** *auto*
    have *rets*: ‹π (*Suc i*) = *return*› **using** *path ret term-path-stable* **by** *auto*
    hence *attsuc*: ‹*att* (π (*Suc i*)) ≠ *None*› **using** *att-ret* **by** *auto*
    hence ∗: ‹*i* ∈ *obs-ids* π› **using** *att-ret ret* **unfolding** *obs-ids-def* **by** *auto*
    have ‹{..< *Suc i*} = *insert i* {..<*i*}› **by** *auto*
    hence *a*: ‹*obs-ids* π ∩ {..< *Suc i*} = *insert i* (*obs-ids* π ∩ {..<*i*})› **using** ∗ **by** *auto*
    have *b*: ‹*i* ∉ *obs-ids* π ∩ {..<*i*}› **by** *auto*
    have ‹*finite* (*obs-ids* π ∩ {..<*i*})› **by** *auto*
    hence ‹*card* (*obs-ids* π ∩ {..<*Suc i*}) = *Suc k*› **by** (*metis card card-insert-disjoint a b*)
    hence *iksuc*: ‹*is-kth-obs* π (*Suc k*) (*Suc i*)› **using** *attsuc* **unfolding** *is-kth-obs-def* **by** *auto*
    have *suckl*: ‹*Suc k* ≤ *l*› **using** *kl* **by** *auto*
    note *less*
    thus ‹*thesis*› **proof** (*cases* ‹*Suc k* < *l*›)
      assume *skl*: ‹*Suc k* < *l*›
      from *less*(*1*)[*OF* - *skl iksuc rets*] *skl*
      show ‹*thesis*› **by** *auto*
    next
      assume ‹¬ *Suc k* < *l*›
      hence ‹*Suc k* = *l*› **using** *suckl* **by** *auto*
      thus ‹*thesis*› **using** *iksuc that* **by** *auto*
    qed
  qed
qed

lemma *no-kth-obs-missing-cs*: **assumes** *path*: ‹*is-path* π› ‹*is-path* π'› **and** *iki*: ‹*is-kth-obs* π *k i*› **and** *not-in-*π': ‹¬(∃ *i'*. *is-kth-obs* π' *k i'*)› **obtains** *l j* **where** ‹*is-kth-obs* π *l j*› ‹¬ (∃ *j'*. cs^{π} *j* = $cs^{\pi'}$ *j'*)›
proof (*rule ccontr*)
  assume ‹¬ *thesis*›
  hence *all-in-*π': ‹∀ *l j*. *is-kth-obs* π *l j* ⟶ (∃ *j'* . cs^{π} *j* = $cs^{\pi'}$ *j'*)› **using** *that* **by** *blast*
  then obtain *i'* **where** *csi*: ‹cs^{π} *i* = $cs^{\pi'}$ *i'*› **using** *assms* **by** *blast*
  hence ‹*att*(π' *i'*) ≠ *None*› **using** *iki* **by** (*metis is-kth-obs-def last-cs*)
  then obtain *k'* **where** *ik'*: ‹*is-kth-obs* π' *k' i'*› **by** (*metis not-none-is-obs*)
  hence *kk'*: ‹*k'* < *k*› **using** *not-in-*π' *kth-obs-stable* **by** (*auto*, *metis not-less-iff-gr-or-eq*)
  show ‹*False*› **proof** (*cases* ‹π *i* = *return*›)
    assume ‹π *i* ≠ *return*›
    thus ‹*False*› **using** *kk' ik' csi iki* **proof** (*induction* ‹*k*› *arbitrary*: ‹*i*› ‹*i'*› ‹*k'*›)
      case *0* **thus** ‹*?case*› **by** *simp*
    next
      case (*Suc k i i' k'*)
      then obtain *j* **where** *ikj*: ‹*is-kth-obs* π *k j*› **by** (*metis kth-obs-stable lessI*)
      then obtain *j'* **where** *csj*: ‹cs^{π} *j* = $cs^{\pi'}$ *j'*› **using** *all-in-*π' **by** *blast*
      hence ‹*att*(π' *j'*) ≠ *None*› **using** *ikj* **by** (*metis is-kth-obs-def last-cs*)
      then obtain *k2* **where** *ik2*: ‹*is-kth-obs* π' *k2 j'*› **by** (*metis not-none-is-obs*)
      have *ji*: ‹*j* < *i*› **using** *kth-obs-mono* [*OF ikj* ‹*is-kth-obs* π (*Suc k*) *i*›] **by** *auto*
      hence *nretj*: ‹π *j* ≠ *return*› **using** *Suc*(*2*) *term-path-stable less-imp-le path*(*1*) **by** *metis*

```
      have ji′: ‹j′ < i′› using cs-order[OF path - - nretj, of ‹j′› ‹i› ‹i′›] csj ‹cs^π i = cs^π′ i′›  ji by auto
      have ‹k2 ≠ k′› using ik2 Suc(4) ji′ kth-obs-unique[of ‹π′› ‹k′› ‹i′› ‹j′›] by (metis less-irrefl)
      hence k2k′: ‹k2 < k′› using kth-obs-mono[OF ‹is-kth-obs π′ k′ i′› ik2] ji′ by (metis not-less-iff-gr-or-eq)
      hence k2k: ‹k2 < k› using Suc by auto
      from Suc.IH[OF nretj k2k ik2 csj ikj] show ‹False› .
    qed
  next
    assume ‹π i = return›
    hence reti′: ‹π′ i′ = return› by (metis csi last-cs)
    from ret-obs-all-obs[OF path(2) ik′ reti′ kk′, of ‹False›] not-in-π′
    show ‹False› by blast
  qed
qed

lemma kth-obs-cs-missing-cs:  assumes path: ‹is-path π› ‹is-path π′› and iki: ‹is-kth-obs π k i› and iki′:
‹is-kth-obs π′ k i′› and csi: ‹cs^π i ≠ cs^π′ i′›
obtains l j where ‹j ≤ i› ‹is-kth-obs π l j› ‹¬ (∃ j′. cs^π j = cs^π′ j′)› | l′ j′ where ‹j′ ≤ i′› ‹is-kth-obs π′ l′
j′› ‹¬ (∃ j. cs^π j = cs^π′ j′)›
proof (rule ccontr)
  assume nt: ‹¬ thesis›
  show ‹False› using iki iki′ csi that proof (induction ‹k› arbitrary: ‹i› ‹i′› rule: less-induct)
    case (less k i i′)
    hence all-in-π′: ‹∀ l j. j≤i ∧ is-kth-obs π l j ⟶ (∃ j′ . cs^π j = cs^π′ j′)›
    and all-in-π: ‹∀ l′ j′. j′ ≤ i′ ∧ is-kth-obs π′ l′ j′ ⟶ (∃ j . cs^π j = cs^π′ j′)› by (metis nt) (metis nt less(6))
    obtain j j′ where csji: ‹cs^π j = cs^π′ i′› and csij: ‹cs^π i = cs^π′ j′› using all-in-π all-in-π′ less by blast
     then obtain l l′ where ilj: ‹is-kth-obs π l j› and ilj′: ‹is-kth-obs π′ l′ j′› by (metis is-kth-obs-def last-cs
less.prems(1,2))
    have lnk: ‹l ≠ k› using ilj csji less(2) less(4) kth-obs-unique by auto
    have lnk′: ‹l′ ≠ k› using ilj′ csij less(3) less(4) kth-obs-unique by auto
    have cseq: ‹∀ l j j′. l < k ∧  is-kth-obs π l j ∧ is-kth-obs π′ l j′ ⟶ cs^π j = cs^π′ j′› proof −
      { fix t p p′ assume tk: ‹t < k› and ikp: ‹is-kth-obs π t p› and ikp′: ‹is-kth-obs π′ t p′›
          hence pi: ‹p < i› and pi′: ‹p′ < i′› by (metis kth-obs-mono less.prems(1)) (metis kth-obs-mono
less.prems(2) tk ikp′)
        have ∗: ‹⋀j l. j ≤ p ⟹ is-kth-obs π l j ⟹ ∃j′. cs^π j = cs^π′ j′› using pi all-in-π′ by auto
        have ∗∗: ‹⋀j′ l′. j′ ≤ p′ ⟹ is-kth-obs π′ l′ j′ ⟹ ∃j. cs^π j = cs^π′ j′› using pi′ all-in-π by auto
        have ‹cs^π p = cs^π′ p′› apply(rule ccontr) using less(1)[OF tk ikp ikp′] ∗ ∗∗ by blast
      }
      thus ‹?thesis› by blast
    qed
    have ii′nret: ‹π i ≠ return ∨ π′ i′ ≠ return› using less cs-return by auto
    have a: ‹k < l ∨ k < l′› proof (rule ccontr)
      assume ‹¬(k < l ∨ k < l′)›
      hence ∗: ‹l < k› ‹l′ < k› using lnk lnk′ by auto
      hence ji: ‹j < i› and ji′: ‹j′ < i′› using ilj ilj′ less(2,3) kth-obs-mono by auto
      show ‹False› using ii′nret proof
        assume nreti: ‹π i ≠ return›
        hence nretj′: ‹π′ j′ ≠ return› using last-cs csij by metis
        show ‹False› using cs-order[OF path(2,1) csij[symmetric] csji[symmetric] nretj′ ji′] ji by simp
      next
        assume nreti′: ‹π′ i′ ≠ return›
        hence nretj′: ‹π j ≠ return› using last-cs csji by metis
        show ‹False› using cs-order[OF path csji csij nretj′ ji] ji′ by simp
      qed
    qed
```

have ‹$l < k \vee l' < k$› **proof** (*rule ccontr*)
assume ‹$\neg$ ($l< k \vee l' < k$)›
hence ‹$k < l$› ‹$k < l'$› **using** *lnk lnk'* **by** *auto*
hence *ji*: ‹$i < j$› **and** *ji'*: ‹$i' < j'$› **using** *ilj ilj' less(2,3) kth-obs-mono* **by** *auto*
show ‹*False*› **using** *ii'nret* **proof**
assume *nreti*: ‹$\pi\ i \neq$ *return*›
show ‹*False*› **using** *cs-order*[*OF path csij csji nreti ji*] *ji'* **by** *simp*
next
assume *nreti'*: ‹$\pi'\ i' \neq$ *return*›
show ‹*False*› **using** *cs-order*[*OF path(2,1) csji*[*symmetric*] *csij*[*symmetric*] *nreti' ji'*] *ji* **by** *simp*
qed
qed
hence ‹$k < l \wedge l' < k \vee k < l' \wedge l < k$› **using** *a* **by** *auto*
thus ‹*False*› **proof**
assume ‹$k < l \wedge l' < k$›
hence *kl*: ‹$k < l$› **and** *lk'*: ‹$l' < k$› **by** *auto*
hence *ij*: ‹$i < j$› **and** *ji'*: ‹$j' < i'$› **using** *less(2,3) ilj ilj' kth-obs-mono* **by** *auto*
have *nreti*: ‹$\pi\ i \neq$ *return*› **by** (*metis csji ii'nret ij last-cs path(1) term-path-stable less-imp-le*)
obtain *h* **where** *ilh*: ‹*is-kth-obs* $\pi\ l'\ h$› **using** *ji' all-in-π ilj' no-kth-obs-missing-cs path(1) path(2)* **by** (*metis kl lk' ilj kth-obs-stable*)
hence ‹$cs^{\pi}\ h = cs^{\pi'}\ j'$› **using** *cseq lk' ilj'* **by** *blast*
hence ‹$cs^{\pi}\ i = cs^{\pi}\ h$› **using** *csij* **by** *auto*
hence *hi*: ‹$h = i$› **using** *cs-inj nreti path(1)* **by** *metis*
have ‹$l' = k$› **using** *less(2) ilh* **unfolding** *hi* **by** (*metis is-kth-obs-def*)
thus ‹*False*› **using** *lk'* **by** *simp*
next
assume ‹$k < l' \wedge l < k$›
hence *kl'*: ‹$k < l'$› **and** *lk*: ‹$l < k$› **by** *auto*
hence *ij'*: ‹$i' < j'$› **and** *ji*: ‹$j < i$› **using** *less(2,3) ilj ilj' kth-obs-mono* **by** *auto*
have *nreti'*: ‹$\pi'\ i' \neq$ *return*› **by** (*metis csij ii'nret ij' last-cs path(2) term-path-stable less-imp-le*)
obtain *h'* **where** *ilh'*: ‹*is-kth-obs* $\pi'\ l\ h'$› **using** *all-in-π' ilj no-kth-obs-missing-cs path(1) path(2) kl' lk ilj' kth-obs-stable* **by** *metis*
hence ‹$cs^{\pi}\ j = cs^{\pi'}\ h'$› **using** *cseq lk ilj* **by** *blast*
hence ‹$cs^{\pi'}\ i' = cs^{\pi'}\ h'$› **using** *csji* **by** *auto*
hence *hi*: ‹$h' = i'$› **using** *cs-inj nreti' path(2)* **by** *metis*
have ‹$l = k$› **using** *less(3) ilh'* **unfolding** *hi* **by** (*metis is-kth-obs-def*)
thus ‹*False*› **using** *lk* **by** *simp*
qed
qed
qed

2.7 Facts about Data

lemma *reads-restrict1*: ‹$\sigma \upharpoonright (reads\ n) = \sigma' \upharpoonright (reads\ n) \Longrightarrow \forall\ x \in reads\ n.\ \sigma\ x = \sigma'\ x$› **by** (*metis restrict-def*)

lemma *reads-restrict2*: ‹$\forall\ x \in reads\ n.\ \sigma\ x = \sigma'\ x \Longrightarrow \sigma \upharpoonright (reads\ n) = \sigma' \upharpoonright (reads\ n)$› **unfolding** *restrict-def* **by** *auto*

lemma *reads-restrict*: ‹$(\sigma \upharpoonright (reads\ n) = \sigma' \upharpoonright (reads\ n)) = (\forall\ x \in reads\ n.\ \sigma\ x = \sigma'\ x)$› **using** *reads-restrict1 reads-restrict2* **by** *metis*

lemma *reads-restr-suc*: ‹$\sigma \upharpoonright (reads\ n) = \sigma' \upharpoonright (reads\ n) \Longrightarrow suc\ n\ \sigma = suc\ n\ \sigma'$› **by** (*metis reads-restrict uses-suc*)

lemma *reads-restr-sem*: ‹$\sigma \upharpoonright (reads\ n) = \sigma' \upharpoonright (reads\ n) \Longrightarrow \forall\ v \in writes\ n.\ sem\ n\ \sigma\ v = sem\ n\ \sigma'\ v$› **by**

(*metis reads-restrict1 uses-writes*)

lemma *reads-obsp*: **assumes** ‹*path* σ *k* = *path* σ' *k'*› ‹σ^k ↾ (*reads* (*path* σ *k*)) = $\sigma'^{k'}$ ↾ (*reads* (*path* σ *k*))›
shows ‹*obsp* σ *k* = *obsp* σ' *k'*›
  using *assms*(*2*) *uses-att*
  unfolding *obsp-def assms*(*1*) *reads-restrict*
  apply (*cases* ‹*att* (*path* σ' *k'*)›)
  by *auto*

lemma *no-writes-unchanged0*: **assumes** ‹∀ *l*<*k*. *v*∉ *writes*(*path* σ *l*)› **shows** ‹(σ^k) *v* = σ *v*› **using** *assms*
proof (*induction* ‹*k*›)
  case *0* **thus** ‹*?case*› **by**(*auto simp add*: *kth-state-def*)
next
  case (*Suc k*)
  hence ‹(σ^k) *v* = σ *v*› **by** *auto*
  moreover
  have ‹$\sigma^{Suc\ k}$ = *snd* (*step* (*path* σ *k*,σ^k))› **by** (*metis kth-state-suc*)
  hence ‹$\sigma^{Suc\ k}$ = *sem* (*path* σ *k*) (σ^k)› **by** (*metis step-suc-sem snd-conv*)
  moreover
  have ‹*v* ∉ *writes* (*path* σ *k*)› **using** *Suc.prems* **by** *blast*
  ultimately
  show ‹*?case*› **using** *writes* **by** *metis*
qed

lemma *written-read-dd*: **assumes** ‹*is-path* π› ‹*v* ∈ *reads* (π *k*) › ‹*v* ∈ *writes* (π *j*)› ‹*j*<*k*› **obtains** *l* **where** ‹*k* $dd^{\pi,v}$→ *l*›
proof −
  let ‹*?l*› = ‹*GREATEST l*. *l* < *k* ∧ *v* ∈ *writes* (π *l*)›
  have ‹*?l* < *k*› **by** (*metis* (*no-types*, *lifting*) *GreatestI-ex-nat assms*(*3*) *assms*(*4*) *less-or-eq-imp-le*)
  moreover
  have ‹*v* ∈ *writes* (π *?l*)› **by** (*metis* (*no-types*, *lifting*) *GreatestI-nat assms*(*3*) *assms*(*4*) *less-or-eq-imp-le*)
  hence ‹*v* ∈ *reads* (π *k*) ∩ *writes* (π *?l*)› **using** *assms*(*2*) **by** *auto*
  moreover
  note *is-ddi-def*
  have ‹∀ *l* ∈ {*?l*<..<*k*}. *v* ∉ *writes* (π *l*)› **by** (*auto*, *metis* (*lifting*, *no-types*) *Greatest-le-nat le-antisym nat-less-le*)
  ultimately
  have ‹*k* $dd^{\pi,v}$→ *?l*› **using** *assms*(*1*) **unfolding** *is-ddi-def* **by** *blast*
  thus ‹*thesis*› **using** *that* **by** *simp*
qed

lemma *no-writes-unchanged*: **assumes** ‹*k* ≤ *l*› ‹∀ *j* ∈ {*k*..<*l*}. *v*∉ *writes*(*path* σ *j*)› **shows** ‹(σ^l) *v* = (σ^k) *v*›
using *assms*
proof (*induction* ‹*l* − *k*› *arbitrary*: ‹*l*›)
  case *0* **thus** ‹*?case*› **by**(*auto*)
next
  case (*Suc lk l*)
  hence *kl*: ‹*k* < *l*› **by** *auto*
  then obtain *l'* **where** *lsuc*: ‹*l* = *Suc l'*› **using** *lessE* **by** *blast*
  hence ‹*lk* = *l'* − *k*› **using** *Suc* **by** *auto*
  moreover
  have ‹∀ *j* ∈ {*k*..<*l'*}. *v* ∉ *writes* (*path* σ *j*)› **using** *Suc*(*4*) *lsuc* **by** *auto*
  ultimately
  have ‹($\sigma^{l'}$) *v* = (σ^k) *v*› **using** *Suc*(*1*)[*of* ‹*l'*›] *lsuc kl* **by** *fastforce*
  moreover

have ‹σ^l = *snd* (*step* (*path* σ l',$\sigma^{l'}$))› **by** (*metis kth-state-suc lsuc*)
hence ‹σ^l = *sem* (*path* σ l') ($\sigma^{l'}$)› **by** (*metis step-suc-sem snd-conv*)
moreover
have ‹$l' < l$› ‹$k \leq l'$› **using** *kl lsuc* **by** *auto*
hence ‹$v \notin$ *writes* (*path* σ l')› **using** *Suc.prems*(*2*) **by** *auto*
ultimately
show ‹*?case*› **using** *writes* **by** *metis*
qed

lemma *ddi-value*: **assumes** ‹l $dd^{(path\ \sigma),v}\!\rightarrow$ k› **shows** ‹(σ^l) v = ($\sigma^{Suc\ k}$) v›
using *assms no-writes-unchanged*[*of* ‹*Suc k*› ‹l› ‹v› ‹σ›] **unfolding** *is-ddi-def* **by** *auto*

lemma *written-value*: **assumes** ‹*path* σ l = *path* σ' l'› ‹σ^l ↾ *reads* (*path* σ l) = $\sigma'^{l'}$ ↾ *reads* (*path* σ l)› ‹$v \in$ *writes* (*path* σ l)›
shows ‹($\sigma^{Suc\ l}$) v = ($\sigma'^{Suc\ l'}$) v›
by (*metis assms reads-restr-sem snd-conv step-suc-sem kth-state-suc*)

2.8 Facts about Contradicting Paths

lemma *obsp-contradict*: **assumes** *csk*: ‹$cs^{path\ \sigma}$ k = $cs^{path\ \sigma'}$ k'› **and** *obs*: ‹*obsp* σ $k \neq$ *obsp* σ' k'› **shows** ‹(σ', k') $\mathfrak{c}$ (σ, k)›
proof −
have *pk*: ‹*path* σ k = *path* σ' k'› **using** *assms last-cs* **by** *metis*
hence ‹σ^k↾(*reads* (*path* σ k)) $\neq$ $\sigma'^{k'}$↾(*reads* (*path* σ k))› **using** *obs reads-obsp*[*OF pk*] **by** *auto*
thus ‹(σ',k') $\mathfrak{c}$ (σ,k)› **using** *contradicts.intros*(*2*)[*OF csk*[*symmetric*]] **by** *auto*
qed

lemma *missing-cs-contradicts*: **assumes** *notin*: ‹¬(∃ k'. $cs^{path\ \sigma}$ k = $cs^{path\ \sigma'}$ k')› **and** *converge*: ‹k<n› ‹$cs^{path\ \sigma}$ n = $cs^{path\ \sigma'}$ n'› **shows** ‹∃ j'. (σ', j') $\mathfrak{c}$ (σ, k)›
proof −
let ‹*?*π› = ‹*path* σ›
let ‹*?*π'› = ‹*path* σ'›
have *init*: ‹*?*π *0* = *?*π' *0*› **unfolding** *path-def* **by** *auto*
have *path*: ‹*is-path ?*π› ‹*is-path ?*π'› **using** *path-is-path* **by** *auto*
obtain j j' **where** *csj*: ‹$cs^{?\pi}$ j = $cs^{?\pi'}$ j'› **and** *cd*: ‹k $cd^{?\pi}\!\rightarrow$$j$› **and** *suc*: ‹*?*$\pi$ (*Suc* j) $\neq$ *?*π' (*Suc* j')› **using** *converged-cd-diverge*[*OF path init notin converge*] **.**
have *less*: ‹$cs^{?\pi}$ j ≺ $cs^{?\pi}$ k› **using** *cd cd-is-cs-less* **by** *auto*
have *nretj*: ‹*?*π j $\neq$ *return*› **by** (*metis cd is-cdi-def term-path-stable less-imp-le*)
have *cs*: ‹*?*π ¡ $cs^{?\pi'}$ j' = j› **using** *csj cs-select-id nretj path-is-path* **by** *metis*
have ‹(σ',j') $\mathfrak{c}$ (σ,k)› **using** *contradicts.intros*(*1*)[*of* ‹*?*π'› ‹j'› ‹*?*π› ‹k› ‹σ› ‹σ'›,*unfolded cs*] *less suc csj* **by** *metis*
thus ‹*?thesis*› **by** *blast*
qed

theorem *obs-neq-contradicts-term*: **fixes** σ σ' **defines** π: ‹π ≡ *path* σ› **and** π': ‹π' ≡ *path* σ'› **assumes** *ret*: ‹π n = *return*› ‹π' n' = *return*› **and** *obsne*: ‹*obs* σ $\neq$ *obs* σ'›
shows ‹∃ k k'. ((σ', k') $\mathfrak{c}$ (σ ,k) ∧ π k ∈ *dom* (*att*)) ∨ ((σ, k) $\mathfrak{c}$ (σ' ,k') ∧ π' k' ∈ *dom* (*att*))›
proof −
have *path*: ‹*is-path* π› ‹*is-path* π'› **using** π π' *path-is-path* **by** *auto*
obtain *k1* **where** *neq*: ‹*obs* σ *k1* $\neq$ *obs* σ' *k1*› **using** *obsne ext*[*of* ‹*obs* σ› ‹*obs* σ'›] **by** *blast*
hence ‹(∃ k i i'. *is-kth-obs* π k i ∧ *is-kth-obs* π' k i' ∧ *obsp* σ i $\neq$ *obsp* σ' i' ∧ cs^{π} i = $cs^{\pi'}$ i')
∨ (∃ k i. *is-kth-obs* π k i ∧ ¬ (∃ i'. cs^{π} i = $cs^{\pi'}$ i'))
∨ (∃ k i'. *is-kth-obs* π' k i' ∧ ¬ (∃ i. cs^{π} i = $cs^{\pi'}$ i'))›
proof(*cases rule*: *option-neq-cases*)

case (*none2 x*)
have *notinπ'*: ‹¬ (∃ *l*. *is-kth-obs* π' *k1 l*)› **using** *none2*(*2*) π' *obs-none-no-kth-obs* **by** *auto*
obtain *i* **where** *inπ*: ‹*is-kth-obs* π *k1 i*› **using** *obs-some-kth-obs*[*of* ‹σ› ‹*k1*›] *none2*(*1*) π **by** *auto*
obtain *l j* **where** ‹*is-kth-obs* π *l j*› ‹¬ (∃ *j'*. cs^{π} *j* = $cs^{\pi'}$ *j'*)› **using** *path inπ notinπ'* **by** (*metis no-kth-obs-missing-cs*)
thus ‹*?thesis*› **by** *blast*
next
case (*none1 x*)
have *notinπ*: ‹¬ (∃ *l*. *is-kth-obs* π *k1 l*)› **using** *none1*(*1*) π *obs-none-no-kth-obs* **by** *auto*
obtain *i'* **where** *inπ'*: ‹*is-kth-obs* π' *k1 i'*› **using** *obs-some-kth-obs*[*of* ‹σ'› ‹*k1*›] *none1*(*2*) π' **by** *auto*

obtain *l j* **where** ‹*is-kth-obs* π' *l j*› ‹¬ (∃ *j'*. cs^{π} *j'* = $cs^{\pi'}$ *j*)› **using** *path inπ' notinπ* **by** (*metis no-kth-obs-missing-cs*)
thus ‹*?thesis*› **by** *blast*
next
case (*some x y*)
obtain *i* **where** *inπ*: ‹*is-kth-obs* π *k1 i*› **using** *obs-some-kth-obs*[*of* ‹σ› ‹*k1*›] *some* π **by** *auto*
obtain *i'* **where** *inπ'*: ‹*is-kth-obs* π' *k1 i'*› **using** *obs-some-kth-obs*[*of* ‹σ'› ‹*k1*›] *some* π' **by** *auto*
show ‹*?thesis*› **proof** (*cases*)
assume ∗: ‹cs^{π} *i* = $cs^{\pi'}$ *i'*›
have ‹*obsp* σ *i* = *obs* σ *k1*› **by** (*metis obs-def* π *inπ kth-obs-unique the-equality*)
moreover
have ‹*obsp* σ' *i'* = *obs* σ' *k1*› **by** (*metis obs-def* π' *inπ' kth-obs-unique the-equality*)
ultimately
have ‹*obsp* σ *i* ≠ *obsp* σ' *i'*› **using** *neq* **by** *auto*
thus ‹*?thesis*› **using** ∗ *inπ inπ'* **by** *blast*
next
assume ∗: ‹cs^{π} *i* ≠ $cs^{\pi'}$ *i'*›
note *kth-obs-cs-missing-cs*[*OF path inπ inπ'* ∗]
thus ‹*?thesis*› **by** *metis*
qed
qed
thus ‹*?thesis*› **proof** (*cases rule*: *three-cases*)
case *1*
then obtain *k i i'* **where** *iki*: ‹*is-kth-obs* π *k i*› ‹*is-kth-obs* π' *k i'*› **and** *obsne*: ‹*obsp* σ *i* ≠ *obsp* σ' *i'*› **and** *csi*: ‹cs^{π} *i* = $cs^{\pi'}$ *i'*› **by** *auto*
note *obsp-contradict*[*OF csi*[*unfolded* π π'] *obsne*]
moreover
have ‹π *i* ∈ *dom att*› **using** *iki* **unfolding** *is-kth-obs-def* **by** *auto*
ultimately
show ‹*?thesis*› **by** *blast*
next
case *2*
then obtain *k i* **where** *iki*: ‹*is-kth-obs* π *k i*› **and** *notinπ'*: ‹¬ (∃*i'*. cs^{π} *i* = $cs^{\pi'}$ *i'*)› **by** *auto*
let ‹*?n*› = ‹*Suc* (*max n i*)›
have *nn*: ‹*n* < *?n*› **by** *auto*
have *iln*: ‹*i* < *?n*› **by** *auto*
have *retn*: ‹π *?n* = *return*› **using** *ret term-path-stable path* **by** *auto*
hence ‹cs^{π} *?n* = $cs^{\pi'}$ *n'*› **using** *ret*(*2*) *cs-return* **by** *auto*
then obtain *i'* **where** ‹(σ',*i'*) 𝔠 (σ,*i*)› **using** *missing-cs-contradicts*[*OF notinπ'*[*unfolded* π π'] *iln*] π π' **by** *auto*
moreover
have ‹π *i* ∈ *dom att*› **using** *iki is-kth-obs-def* **by** *auto*
ultimately
show ‹*?thesis*› **by** *blast*

next
case *3*
then obtain $k\ i'$ **where** *iki*: ‹*is-kth-obs* $\pi'\ k\ i'$› **and** *notin*π': ‹$\neg\ (\exists\, i.\ cs^{\pi}\ \ i = cs^{\pi'}\ \ i')$› **by** *auto*
let ‹*?n*› = ‹*Suc* (*max* $n'\ i'$)›
have *nn*: ‹$n' <$ *?n*› **by** *auto*
have *iln*: ‹$i' <$ *?n*› **by** *auto*
have *retn*: ‹π' *?n* = *return*› **using** *ret term-path-stable path* **by** *auto*
hence ‹$cs^{\pi}\ \ n = cs^{\pi'}$ *?n*› **using** *ret*(*1*) *cs-return* **by** *auto*
then obtain i **where** ‹$(\sigma,i)\ \mathfrak{c}\ (\sigma',i')$› **using** *missing-cs-contradicts notin*π' *iln* $\pi\ \pi'$ **by** *metis*
moreover
have ‹$\pi'\ i' \in$ *dom att*› **using** *iki is-kth-obs-def* **by** *auto*
ultimately
show ‹*?thesis*› **by** *blast*
qed
qed

lemma *obs-neq-some-contradicts'*: **fixes** $\sigma\ \sigma'$ **defines** π: ‹$\pi \equiv$ *path* σ› **and** π': ‹$\pi' \equiv$ *path* σ'›
assumes *obsnecs*: ‹*obsp* $\sigma\ i \neq$ *obsp* $\sigma'\ i' \vee cs^{\pi}\ \ i \neq cs^{\pi'}\ \ i'$›
and *iki*: ‹*is-kth-obs* $\pi\ k\ i$› **and** *iki'*: ‹*is-kth-obs* $\pi'\ k\ i'$›
shows ‹$\exists\ k\ k'.\ ((\sigma',\ k')\ \mathfrak{c}\ (\sigma\ ,k) \wedge \pi\ k \in$ *dom att*$) \vee ((\sigma,\ k)\ \mathfrak{c}\ (\sigma'\ ,k') \wedge \pi'\ k' \in$ *dom att*$)$›
using *obsnecs iki iki'* **proof** (*induction* ‹*k*› *arbitrary*: ‹*i*› ‹*i'*› *rule*: *less-induct*)
case (*less k i i'*)
note *iki* = ‹*is-kth-obs* $\pi\ k\ i$›
and *iki'* = ‹*is-kth-obs* $\pi'\ k\ i'$›
have *domi*: ‹$\pi\ i \in$ *dom att*› **by** (*metis is-kth-obs-def domIff iki*)
have *domi'*: ‹$\pi'\ i' \in$ *dom att*› **by** (*metis is-kth-obs-def domIff iki'*)
note *obsnecs* = ‹*obsp* $\sigma\ i \neq$ *obsp* $\sigma'\ i' \vee cs^{\pi}\ \ i \neq cs^{\pi'}\ \ i'$›
show ‹*?thesis*› **proof** *cases*
assume *csi*: ‹$cs^{\pi}\ \ i = cs^{\pi'}\ \ i'$›
hence $*$: ‹*obsp* $\sigma\ i \neq$ *obsp* $\sigma'\ i'$› **using** *obsnecs* **by** *auto*
note *obsp-contradict*[*OF* - $*$] *csi domi* $\pi\ \pi'$
thus ‹*?thesis*› **by** *blast*
next
assume *ncsi*: ‹$cs^{\pi}\ \ i \neq cs^{\pi'}\ \ i'$›
have *path*: ‹*is-path* π› ‹*is-path* π'› **using** $\pi\ \pi'$ *path-is-path* **by** *auto*
have π*0*: ‹$\pi\ \ 0 = \pi'\ \ 0$› **unfolding** $\pi\ \pi'$ *path-def* **by** *auto*
note *kth-obs-cs-missing-cs*[*of* ‹π› ‹π'› ‹*k*› ‹*i*› ‹*i'*›] $\pi\ \pi'$ *path-is-path iki iki' ncsi*
hence ‹$(\exists\ \ l\ j\ .j \leq i \wedge$ *is-kth-obs* $\pi\ l\ j \wedge \neg\ (\exists\ j'.\ cs^{\pi}\ \ j = cs^{\pi'}\ \ j')) \vee (\exists\ \ l'\ j'.\ j' \leq i' \wedge$ *is-kth-obs* $\pi'\ l'\ j' \wedge \neg\ (\exists\ j.\ cs^{\pi}\ \ j = cs^{\pi'}\ \ j'))$› **by** *metis*
thus ‹*?thesis*› **proof**
assume ‹$\exists\, l\ j.\ j \leq i \wedge$ *is-kth-obs* $\pi\ l\ j \wedge \neg\ (\exists\, j'.\ cs^{\pi}\ \ j = cs^{\pi'}\ \ j')$›
then obtain $l\ j$ **where** *ji*: ‹$j{\leq}i$› **and** *iobs*: ‹*is-kth-obs* $\pi\ l\ j$› **and** *notin*: ‹$\neg\ (\exists\, j'.\ cs^{\pi}\ \ j = cs^{\pi'}\ \ j')$› **by** *blast*
have *dom*: ‹$\pi\ j \in$ *dom att*› **using** *iobs is-kth-obs-def* **by** *auto*
obtain $n\ n'$ **where** *nj*: ‹$n < j$› **and** *csn*: ‹$cs^{\pi}\ \ n = cs^{\pi'}\ \ n'$› **and** *sucn*: ‹$\pi$ (*Suc* n) $\neq \pi'$ (*Suc* n')› **and** *cdloop*: ‹$j\ cd^{\pi}\!\rightarrow n \vee (\forall\ \ j'{>}\ n'.\ j'\ cd^{\pi'}\!\rightarrow n')$›
using *missing-cd-or-loop*[*OF path* π*0* *notin*] **by** *blast*
show ‹*?thesis*› **using** *cdloop* **proof**
assume *cdjn*: ‹$j\ cd^{\pi}\!\rightarrow\ n$›
hence *csnj*: ‹$cs^{\pi'}\ \ n' \prec cs^{\pi}\ \ j$› **using** *csn* **by** (*metis cd-is-cs-less*)
have *cssel*: ‹π (*Suc* (π ¡ $cs^{\pi'}\ \ n'$)) = π (*Suc* n)› **using** *csn* **by** (*metis cdjn cd-not-ret cs-select-id path*(*1*))
have ‹$(\sigma',n')\ \mathfrak{c}\ (\sigma,j)$› **using** *csnj* **apply**(*rule contradicts.intros*(*1*)) **using** *cssel* $\pi\ \pi'$ *sucn* **by** *auto*
thus ‹*?thesis*› **using** *dom* **by** *auto*
next
assume *loop*: ‹$\forall\ \ j'{>}n'.\ j'\ cd^{\pi'}\!\rightarrow n'$›

show ‹*?thesis*› **proof** *cases*
assume *in'*: ‹$i' \leq n'$›
have *nreti'*: ‹π' i' $\neq$ *return*› **by**(*metis le-eq-less-or-eq lessI loop not-le path*(*2*) *ret-no-cd term-path-stable*)
show ‹*?thesis*› **proof** *cases*
assume ‹$\exists\ \iota.\ cs^{\pi'}\ i' = cs^{\pi}\ \iota$›
then obtain ι **where** *csι*: ‹$cs^{\pi}\ \iota = cs^{\pi'}\ i'$› **by** *metis*
have *ιn*: ‹$\iota \leq n$› **using** *cs-order-le*[*OF path*(*2*,*1*) *csι*[*symmetric*] *csn*[*symmetric*] *nreti' in'*] **.**
hence *ιi*: ‹$\iota < i$› **using** *nj ji* **by** *auto*
have *domι*: ‹$\pi\ \iota \in$ *dom att*› **using** *domi' csι last-cs* **by** *metis*
obtain κ **where** *iκι*: ‹*is-kth-obs* $\pi\ \kappa\ \iota$› **using** *domι* **by** (*metis is-kth-obs-def domIff*)
hence *κk*: ‹$\kappa < k$› **using** *ιi iki* **by** (*metis kth-obs-le-iff*)
obtain ι' **where** *iκι'*: ‹*is-kth-obs* $\pi'\ \kappa\ \iota'$› **using** *κk iki'* **by** (*metis kth-obs-stable*)
have ‹$\iota' < i'$› **using** *κk iki' iκι'* **by** (*metis kth-obs-le-iff*)
hence *csι'*: ‹$cs^{\pi}\ \iota \neq cs^{\pi'}\ \iota'$› **unfolding** *csι* **using** *cs-inj*[*OF path*(*2*) *nreti'*, *of* ‹ι'›] **by** *blast*
thus ‹*?thesis*› **using** *less*(*1*)[*OF κk - iκι iκι'*] **by** *auto*
next
assume *notin''*: ‹$\neg(\exists\ \iota.\ cs^{\pi'}\ i' = cs^{\pi}\ \iota)$›
obtain $\iota\ \iota'$ **where** *ιi'*: ‹$\iota' < i'$› **and** *csι*: ‹$cs^{\pi}\ \iota = cs^{\pi'}\ \iota'$› **and** *sucι*: ‹$\pi$ (*Suc* ι) $\neq$ π' (*Suc* ι')› **and** *cdloop'*: ‹$i'\ cd^{\pi'}\!\rightarrow \iota' \vee (\forall\ j{>}\iota.\ j\ cd^{\pi}\!\rightarrow \iota)$›
using *missing-cd-or-loop*[*OF path*(*2*,*1*) *π0*[*symmetric*] *notin''*] **by** *metis*
show ‹*?thesis*› **using** *cdloop'* **proof**
assume *cdjn*: ‹$i'\ cd^{\pi'}\!\rightarrow \iota'$›
hence *csnj*: ‹$cs^{\pi}\ \iota \prec cs^{\pi'}\ i'$› **using** *csι* **by** (*metis cd-is-cs-less*)
have *cssel*: ‹π' (*Suc* (π' ¡ $cs^{\pi}\ \iota$)) = π' (*Suc* ι')› **using** *csι* **by** (*metis cdjn cd-not-ret cs-select-id path*(*2*))
have ‹(σ,ι) 𝔠 (σ',i')› **using** *csnj* **apply**(*rule contradicts.intros*(*1*)) **using** *cssel* π π' *sucι* **by** *auto*
thus ‹*?thesis*› **using** *domi'* **by** *auto*
next
assume *loop'*: ‹$\forall\ j{>}\iota.\ j\ cd^{\pi}\!\rightarrow \iota$›
have *ιn'*: ‹$\iota' < n'$› **using** *in' ιi'* **by** *auto*
have *nretι'*: ‹$\pi'\ \iota' \neq$ *return*› **by** (*metis csι last-cs le-eq-less-or-eq lessI path*(*1*) *path*(*2*) *sucι term-path-stable*)
have ‹$\iota < n$› **using** *cs-order*[*OF path*(*2*,*1*) *csι*[*symmetric*] *csn*[*symmetric*] *nretι' ιn'*] **.**
hence ‹$\iota < i$› **using** *nj ji* **by** *auto*
hence *cdiι*: ‹$i\ cd^{\pi}\!\rightarrow \iota$› **using** *loop'* **by** *auto*
hence *csιi*: ‹$cs^{\pi'}\ \iota' \prec cs^{\pi}\ i$› **using** *csι* **by** (*metis cd-is-cs-less*)
have *cssel*: ‹π (*Suc* (π ¡ $cs^{\pi'}\ \iota'$)) = π (*Suc* ι)› **using** *csι* **by** (*metis cdiι cd-not-ret cs-select-id path*(*1*))
have ‹(σ',ι') 𝔠 (σ,i)› **using** *csιi* **apply**(*rule contradicts.intros*(*1*)) **using** *cssel* π π' *sucι* **by** *auto*
thus ‹*?thesis*› **using** *domi* **by** *auto*
qed
qed
next
assume ‹$\neg\ i' \leq n'$›
hence *ni'*: ‹$n' < i'$› **by** *simp*
hence *cdin*: ‹$i'\ cd^{\pi'}\!\rightarrow n'$› **using** *loop* **by** *auto*
hence *csni*: ‹$cs^{\pi}\ n \prec cs^{\pi'}\ i'$› **using** *csn* **by** (*metis cd-is-cs-less*)
have *cssel*: ‹π' (*Suc* (π' ¡ $cs^{\pi}\ n$)) = π' (*Suc* n')› **using** *csn* **by** (*metis cdin cd-not-ret cs-select-id path*(*2*))
have ‹(σ,n) 𝔠 (σ',i')› **using** *csni* **apply**(*rule contradicts.intros*(*1*)) **using** *cssel* π π' *sucn* **by** *auto*
thus ‹*?thesis*› **using** *domi'* **by** *auto*
qed
qed
next

— Symmetric case as above, indices might be messy.

assume $\langle\exists\, l\, j.\ j \leq i' \wedge$ *is-kth-obs* $\pi'\ l\ j \wedge \neg\ (\exists j'.\ cs^{\pi}\ j' = cs^{\pi'}\ j)\rangle$

then obtain *l j* **where** *ji'*: $\langle j \leq i'\rangle$ **and** *iobs*: $\langle$*is-kth-obs* $\pi'\ l\ j\rangle$ **and** *notin*: $\langle\neg\ (\exists j'.\ cs^{\pi'}\ j = cs^{\pi}\ j')\rangle$ **by** *metis*

have *dom*: $\langle\pi'\ j \in$ *dom att*$\rangle$ **using** *iobs is-kth-obs-def* **by** *auto*

obtain *n n'* **where** *nj*: $\langle n < j\rangle$ **and** *csn*: $\langle cs^{\pi'}\ n = cs^{\pi}\ n'\rangle$ **and** *sucn*: $\langle\pi'\ (Suc\ n) \neq \pi\ (Suc\ n')\rangle$ **and** *cdloop*: $\langle j\ cd^{\pi'}\!\!\rightarrow n \vee (\forall\ j'> n'.\ j'\ cd^{\pi}\!\!\rightarrow n')\rangle$

using *missing-cd-or-loop*[*OF path*(*2*,*1*) $\pi 0$[*symmetric*]] *notin* **by** *metis*

show $\langle$*?thesis*$\rangle$ **using** *cdloop* **proof**

assume *cdjn*: $\langle j\ cd^{\pi'}\!\!\rightarrow n\rangle$

hence *csnj*: $\langle cs^{\pi}\ n' \prec cs^{\pi'}\ j\rangle$ **using** *csn* **by** (*metis cd-is-cs-less*)

have *cssel*: $\langle\pi'\ (Suc\ (\pi'\ ¡\ cs^{\pi}\ n')) = \pi'\ (Suc\ n)\rangle$ **using** *csn* **by** (*metis cdjn cd-not-ret cs-select-id path*(*2*))

have $\langle(\sigma,n')\ \mathfrak{c}\ (\sigma',j)\rangle$ **using** *csnj* **apply**(*rule contradicts.intros*(*1*)) **using** *cssel* $\pi'\ \pi$ *sucn* **by** *auto*

thus $\langle$*?thesis*$\rangle$ **using** *dom* **by** *auto*

next

assume *loop*: $\langle\forall\ j'>n'.\ j'\ cd^{\pi}\!\!\rightarrow n'\rangle$

show $\langle$*?thesis*$\rangle$ **proof** *cases*

assume *in'*: $\langle i \leq n'\rangle$

have *nreti*: $\langle\pi\ i \neq$ *return*$\rangle$ **by** (*metis le-eq-less-or-eq lessI loop not-le path*(*1*) *ret-no-cd term-path-stable*)

show $\langle$*?thesis*$\rangle$ **proof** *cases*

assume $\langle\exists\ \iota.\ cs^{\pi}\ i = cs^{\pi'}\ \iota\rangle$

then obtain ι **where** *cs*ι: $\langle cs^{\pi'}\ \iota = cs^{\pi}\ i\rangle$ **by** *metis*

have ιn: $\langle\iota \leq n\rangle$ **using** *cs-order-le*[*OF path cs*ι[*symmetric*] *csn*[*symmetric*] *nreti in'*] **.**

hence $\iota i'$: $\langle\iota < i'\rangle$ **using** *nj ji'* **by** *auto*

have *dom*ι: $\langle\pi'\ \iota \in$ *dom att*$\rangle$ **using** *domi cs*ι *last-cs* **by** *metis*

obtain κ **where** *i*$\kappa\iota$: $\langle$*is-kth-obs* $\pi'\ \kappa\ \iota\rangle$ **using** *dom*ι **by** (*metis is-kth-obs-def domIff*)

hence κk: $\langle\kappa < k\rangle$ **using** $\iota i'$ *iki'* **by** (*metis kth-obs-le-iff*)

obtain ι' **where** *i*$\kappa\iota'$: $\langle$*is-kth-obs* $\pi\ \kappa\ \iota'\rangle$ **using** κk *iki* **by** (*metis kth-obs-stable*)

have $\langle\iota' < i\rangle$ **using** κk *iki i*$\kappa\iota'$ **by** (*metis kth-obs-le-iff*)

hence *cs*ι': $\langle cs^{\pi'}\ \iota \neq cs^{\pi}\ \iota'\rangle$ **unfolding** *cs*ι **using** *cs-inj*[*OF path*(*1*) *nreti*, *of* $\langle\iota'\rangle$] **by** *blast*

thus $\langle$*?thesis*$\rangle$ **using** *less*(*1*)[*OF* κk - *i*$\kappa\iota'$ *i*$\kappa\iota$] **by** *auto*

next

assume *notin''*: $\langle\neg(\exists\ \iota.\ cs^{\pi}\ i = cs^{\pi'}\ \iota)\rangle$

obtain $\iota\ \iota'$ **where** ιi: $\langle\iota' < i\rangle$ **and** *cs*ι: $\langle cs^{\pi'}\ \iota = cs^{\pi}\ \iota'\rangle$ **and** *suc*ι: $\langle\pi'\ (Suc\ \iota) \neq \pi\ (Suc\ \iota')\rangle$ **and** *cdloop'*: $\langle i\ cd^{\pi}\!\!\rightarrow \iota' \vee (\forall\ j>\iota.\ j\ cd^{\pi'}\!\!\rightarrow \iota)\rangle$

using *missing-cd-or-loop*[*OF path* $\pi 0$ *notin''*] **by** *metis*

show $\langle$*?thesis*$\rangle$ **using** *cdloop'* **proof**

assume *cdjn*: $\langle i\ cd^{\pi}\!\!\rightarrow \iota'\rangle$

hence *csnj*: $\langle cs^{\pi'}\ \iota \prec cs^{\pi}\ i\rangle$ **using** *cs*ι **by** (*metis cd-is-cs-less*)

have *cssel*: $\langle\pi\ (Suc\ (\pi\ ¡\ cs^{\pi'}\ \iota)) = \pi\ (Suc\ \iota')\rangle$ **using** *cs*ι **by** (*metis cdjn cd-not-ret cs-select-id path*(*1*))

have $\langle(\sigma',\iota)\ \mathfrak{c}\ (\sigma,i)\rangle$ **using** *csnj* **apply**(*rule contradicts.intros*(*1*)) **using** *cssel* $\pi'\ \pi$ *suc*ι **by** *auto*

thus $\langle$*?thesis*$\rangle$ **using** *domi* **by** *auto*

next

assume *loop'*: $\langle\forall\ j>\iota.\ j\ cd^{\pi'}\!\!\rightarrow \iota\rangle$

have $\iota n'$: $\langle\iota' < n'\rangle$ **using** *in'* ιi **by** *auto*

have *nret*ι': $\langle\pi\ \iota' \neq$ *return*$\rangle$ **by** (*metis cs*ι *last-cs le-eq-less-or-eq lessI path*(*1*) *path*(*2*) *suc*ι *term-path-stable*)

have $\langle\iota < n\rangle$ **using** *cs-order*[*OF path cs*ι[*symmetric*] *csn*[*symmetric*] *nret*ι' $\iota n'$] **.**

hence $\langle\iota < i'\rangle$ **using** *nj ji'* **by** *auto*

hence *cdi*ι: $\langle i'\ cd^{\pi'}\!\!\rightarrow \iota\rangle$ **using** *loop'* **by** *auto*

hence *cs*$\iota i'$: $\langle cs^{\pi}\ \iota' \prec cs^{\pi'}\ i'\rangle$ **using** *cs*ι **by** (*metis cd-is-cs-less*)

have *cssel*: $\langle\pi'\ (Suc\ (\pi'\ ¡\ cs^{\pi}\ \iota')) = \pi'\ (Suc\ \iota)\rangle$ **using** *cs*ι **by** (*metis cdi*ι *cd-not-ret cs-select-id*

path(*2*))
have ‹(σ,ι') $\mathfrak{c}$ (σ',i')› **using** *csιi′* **apply**(*rule contradicts.intros*(*1*)) **using** *cssel* π' π *sucι* **by** *auto*
thus ‹*?thesis*› **using** *domi′* **by** *auto*
qed
qed
next
assume ‹$\neg$ $i \leq n'$›
hence *ni*: ‹$n'< i$› **by** *simp*
hence *cdin*: ‹i $cd^{\pi}\rightarrow$ n'› **using** *loop* **by** *auto*
hence *csni′*: ‹$cs^{\pi'}$ $n \prec cs^{\pi}$ i› **using** *csn* **by** (*metis cd-is-cs-less*)
have *cssel*: ‹π (*Suc* (π ¡ $cs^{\pi'}$ n)) $= \pi$ (*Suc* n')› **using** *csn* **by** (*metis cdin cd-not-ret cs-select-id path*(*1*))
have ‹(σ',n) $\mathfrak{c}$ (σ,i)› **using** *csni′* **apply**(*rule contradicts.intros*(*1*)) **using** *cssel* π' π *sucn* **by** *auto*
thus ‹*?thesis*› **using** *domi* **by** *auto*
qed
qed
qed
qed
qed

theorem *obs-neq-some-contradicts*: **fixes** σ σ' **defines** π: ‹$\pi \equiv$ *path* σ› **and** π': ‹$\pi' \equiv$ *path* σ'›
assumes *obsne*: ‹*obs* σ $k \neq$ *obs* σ' k› **and** *not-none*: ‹*obs* σ $k \neq$ *None*› ‹*obs* σ' $k \neq$ *None*›
shows ‹$\exists$ k k'. $((\sigma', k')$ $\mathfrak{c}$ $(\sigma ,k) \wedge \pi$ $k \in$ *dom att*) $\vee$ $((\sigma, k)$ $\mathfrak{c}$ $(\sigma' ,k') \wedge \pi'$ $k' \in$ *dom att*)›
proof −
obtain i **where** *iki*: ‹*is-kth-obs* π k i› **using** *not-none*(*1*) **by** (*metis* π *obs-some-kth-obs*)
obtain i' **where** *iki′*: ‹*is-kth-obs* π' k i'› **using** *not-none*(*2*) **by** (*metis* π' *obs-some-kth-obs*)
have ‹*obsp* σ $i =$ *obs* σ k› **by** (*metis* π *iki kth-obs-unique obs-def the-equality*)
moreover
have ‹*obsp* σ' $i' =$ *obs* σ' k› **by** (*metis* π' *iki′ kth-obs-unique obs-def the-equality*)
ultimately
have *obspne*: ‹*obsp* σ $i \neq$ *obsp* σ' i'› **using** *obsne* **by** *auto*
show ‹*?thesis*› **using** *obs-neq-some-contradicts′*[*OF* - *iki*[*unfolded* π] *iki′*[*unfolded* π']] **using** *obspne* π π' **by** *metis*
qed

theorem *obs-neq-ret-contradicts*: **fixes** σ σ' **defines** π: ‹$\pi \equiv$ *path* σ› **and** π': ‹$\pi' \equiv$ *path* σ'›
assumes *ret*: ‹π $n =$ *return*› **and** *obsne*: ‹*obs* σ' $i \neq$ *obs* σ i› **and** *obs*:‹*obs* σ' $i \neq$ *None*›
shows ‹$\exists$ k k'. $((\sigma', k')$ $\mathfrak{c}$ $(\sigma ,k) \wedge \pi$ $k \in$ *dom* (*att*)) $\vee$ $((\sigma, k)$ $\mathfrak{c}$ $(\sigma' ,k') \wedge \pi'$ $k' \in$ *dom* (*att*))›
proof (*cases* ‹$\exists$ j k'. *is-kth-obs* π' j k' $\wedge$ ($\nexists$ k. cs^{π} $k = cs^{\pi'}$ k')›)
case *True*
obtain l k' **where** *jk′*: ‹*is-kth-obs* π' l k'› **and** *unmatched*: ‹($\nexists$ k. cs^{π} $k = cs^{\pi'}$ k')› **using** *True* **by** *blast*
have $\pi 0$: ‹π $0 = \pi'$ 0› **using** π π' *path0* **by** *auto*
obtain j j' **where** *csj*: ‹cs^{π} $j = cs^{\pi'}$ j'› **and** *cd*: ‹k' $cd^{\pi'}\rightarrow j'$› **and** *suc*: ‹π (*Suc* j) $\neq \pi'$ (*Suc* j')›
using *converged-cd-diverge-return*[*of* ‹π'› ‹π› ‹k'› ‹n›] *ret unmatched path-is-path* π π' $\pi 0$ **by** *metis*
hence $*$: ‹(σ, j) $\mathfrak{c}$ (σ' ,k')› **using** *contradicts.intros*(*1*)[*of* ‹π› ‹j› ‹π'› ‹k'› ‹σ'› ‹σ›, *unfolded csj*] π π'
using *cd-is-cs-less cd-not-ret cs-select-id* **by** *auto*
have ‹π' $k' \in$ *dom*(*att*)› **using** *jk′* **by** (*meson domIff is-kth-obs-def*)
thus ‹*?thesis*› **using** $*$ **by** *blast*
next
case *False*
hence $*$: ‹$\bigwedge$ j k'. *is-kth-obs* π' j $k' \Longrightarrow \exists$ k. cs^{π} $k = cs^{\pi'}$ k'› **by** *auto*
obtain k' **where** k': ‹*is-kth-obs* π' i k'› **using** *obs* π' *obs-some-kth-obs* **by** *blast*
obtain l **where** ‹*is-kth-obs* π i l› **using** $*$ π π' k' *no-kth-obs-missing-cs path-is-path* **by** *metis*
thus ‹*?thesis*› **using** π π' *obs obs-neq-some-contradicts obs-none-no-kth-obs obsne* **by** *metis*
qed

2.9 Facts about Critical Observable Paths

lemma *contradicting-in-cp*: **assumes** *leq*:‹$\sigma =_L \sigma'$› **and** *cseq*: ‹$cs^{path\ \sigma}\ k = cs^{path\ \sigma'}\ k'$›
and *readv*: ‹$v \in reads(path\ \sigma\ k)$› **and** *vneq*: ‹$(\sigma^k)\ v \neq (\sigma'^{k'})\ v$› **shows** ‹$((\sigma,k),(\sigma',k')) \in cp$›
  using *cseq readv vneq* **proof**(*induction* ‹$k+k'$› *arbitrary*: ‹k› ‹k'› ‹v› *rule*: *less-induct*)
  fix $k\ k'\ v$
  assume *csk*: ‹$cs^{path\ \sigma}\ k = cs^{path\ \sigma'}\ k'$›
  assume *vread*: ‹$v \in reads\ (path\ \sigma\ k)$›
  assume *vneq*: ‹$(\sigma^k)\ v \neq (\sigma'^{k'})\ v$›
  assume *IH*: ‹$\bigwedge ka\ k'a\ v.\ ka + k'a < k + k' \Longrightarrow cs^{path\ \sigma}\ ka = cs^{path\ \sigma'}\ k'a \Longrightarrow v \in reads\ (path\ \sigma\ ka) \Longrightarrow (\sigma^{ka})\ v \neq (\sigma'^{k'a})\ v \Longrightarrow ((\sigma,\ ka),\ \sigma',\ k'a) \in cp$›

  define π **where** ‹$\pi \equiv path\ \sigma$›
  define π' **where** ‹$\pi' \equiv path\ \sigma'$›
  have *path*: ‹$\pi = path\ \sigma$› ‹$\pi' = path\ \sigma'$› **using** *π-def π′-def path-is-path* **by** *auto*
  have *ip*: ‹*is-path* π› ‹*is-path* π'› **using** *path path-is-path* **by** *auto*

  have *π0*: ‹$\pi'\ 0 = \pi\ 0$› **unfolding** *path path-def* **by** *auto*
  have *vread′*: ‹$v \in reads\ (path\ \sigma'\ k')$› **using** *csk vread* **by** (*metis last-cs*)
  have *cseq*: ‹$cs^{\pi'}\ k' = cs^{\pi}\ k$› **using** *csk path* **by** *simp*

  show ‹$((\sigma,\ k),\ \sigma',\ k') \in cp$› **proof** *cases*
   assume *vnw*: ‹$\forall\ l < k.\ v \notin writes\ (\pi\ l)$›
   hence *σv*: ‹$(\sigma^k)\ v = \sigma\ v$› **by** (*metis no-writes-unchanged0 path*(*1*))
   show ‹*?thesis*› **proof** *cases*
    assume *vnw′*: ‹$\forall\ l < k'.\ v \notin writes\ (\pi'\ l)$›
    hence *σv′*: ‹$(\sigma'^{k'})\ v = \sigma'\ v$› **by** (*metis no-writes-unchanged0 path*(*2*))
    with *σv vneq* **have** ‹$\sigma\ v \neq \sigma'\ v$› **by** *auto*
    hence *vhigh*: ‹$v \in hvars$› **using** *leq* **unfolding** *loweq-def restrict-def* **by** (*auto,metis*)
    thus ‹*?thesis*› **using** *cp.intros*(*1*)[*OF leq csk vread vneq*] *vnw vnw′ path* **by** *simp*
   next
    assume ‹$\neg(\forall\ l < k'.\ v \notin writes\ (\pi'\ l))$›
    then obtain l' **where** *kddl′*: ‹$k'\ dd^{\pi',v}\!\rightarrow l'$› **using** *path*(*2*) *path-is-path written-read-dd vread′* **by** *blast*
    hence *lv′*: ‹$v \in writes\ (\pi'\ l')$› **unfolding** *is-ddi-def* **by** *auto*
    have *lk′*: ‹$l' < k'$› **by** (*metis is-ddi-def kddl′*)
    have *nret*: ‹$\pi'\ l' \neq return$› **using** *lv′ writes-return* **by** *auto*

    have *notinπ*: ‹$\neg\ (\exists\,l.\ cs^{\pi'}\ l' = cs^{\pi}\ l)$› **proof**
     assume ‹$\exists\,l.\ cs^{\pi'}\ l' = cs^{\pi}\ l$›
     then obtain l **where** $cs^{\pi'}\ l' = cs^{\pi}\ l$ **..**
     note *csl* = ‹$cs^{\pi'}\ l' = cs^{\pi}\ l$›
     have *lk*: ‹$l < k$› **using** *lk′ cseq ip cs-order*[*of* ‹π'› ‹π› ‹l'› ‹l› ‹k'› ‹k›] *csl nret path* **by** *force*

     have ‹$v \in writes\ (\pi\ l)$› **using** *csl lv′ last-cs* **by** *metis*
     thus ‹*False*› **using** *lk vnw* **by** *blast*
    qed

    from *converged-cd-diverge*[*OF ip*(*2,1*) *π0 notinπ lk′ cseq*]
    obtain $i\ i'$ **where** *csi*: ‹$cs^{\pi'}\ i' = cs^{\pi}\ i$› **and** *lcdi*: ‹$l'\ cd^{\pi'}\!\rightarrow i'$› **and** *div*: ‹$\pi'\ (Suc\ i') \neq \pi\ (Suc\ i)$› **.**

    have *1*: ‹$\pi\ (Suc\ i) = suc\ (\pi\ i)\ (\sigma^i)$› **by** (*metis step-suc-sem fst-conv path*(*1*) *path-suc*)
    have *2*: ‹$\pi'\ (Suc\ i') = suc\ (\pi'\ i')\ (\sigma'^{i'})$› **by** (*metis step-suc-sem fst-conv path*(*2*) *path-suc*)
    have *3*: ‹$\pi'\ i' = \pi\ i$› **using** *csi last-cs* **by** *metis*
    have *nreads*: ‹$\sigma^i \upharpoonright reads\ (\pi\ i) \neq \sigma'^{i'} \upharpoonright reads\ (\pi\ i)$› **by** (*metis 1 2 3 div reads-restr-suc*)

then obtain v' **where** *v'read*: ‹$v' \in$ *reads*(*path* σ i)› ‹(σ^i) $v' \neq (\sigma'^{i'})$ v'› **unfolding** *path* **by** (*metis reads-restrict*)

have *nreti*: ‹π' $i' \neq$ *return*› **by** (*metis csi div ip*(*1*) *ip*(*2*) *last-cs lessI term-path-stable less-imp-le*)
have *ik'*: ‹$i' < k'$› **using** *lcdi lk' is-cdi-def* **by** *auto*
have *ik*: ‹$i < k$› **using** *cs-order*[*OF ip*(*2*,*1*) *csi cseq nreti ik'*] **.**

have *cpi*: ‹$((\sigma, i), (\sigma', i')) \in$ *cp*› **using** *IH*[*of* ‹i› ‹i'›] *v'read csi ik ik' path* **by** *auto*
hence *cpi'*: ‹$((\sigma', i'), (\sigma, i)) \in$ *cp*› **using** *cp.intros*(*4*) **by** *blast*

have *nwvi*: ‹$\forall j' \in \{$*LEAST* $i'.$ $i < i' \wedge (\exists i.$ $cs^{path\ \sigma'}$ $i = cs^{path\ \sigma}$ $i')..<k\}.$ $v \notin$ *writes* (*path* σ j')› **using** *vnw*[*unfolded path*]
by (*metis* (*poly-guards-query*) *atLeastLessThan-iff*)

from *cp.intros*(*3*)[*OF cpi' kddl'*[*unfolded path*] *lcdi*[*unfolded path*] *csk*[*symmetric*] *div*[*unfolded path*] *vneq*[*symmetric*] *nwvi*]

show ‹*?thesis*› **using** *cp.intros*(*4*) **by** *simp*
qed
next
assume *wv*: ‹$\neg$ ($\forall$ $l<k.$ $v \notin$ *writes* (π l))›
then obtain l **where** *kddl*: ‹k $dd^{\pi,v}\!\rightarrow$ l› **using** *path*(*1*) *path-is-path written-read-dd vread* **by** *blast*
hence *lv*: ‹$v \in$ *writes* (π l)› **unfolding** *is-ddi-def* **by** *auto*
have *lk*: ‹$l < k$› **by** (*metis is-ddi-def kddl*)
have *nret*: ‹π $l \neq$ *return*› **using** *lv writes-return* **by** *auto*
have *nwb*: ‹$\forall$ $i \in \{$*Suc* $l..< k\}.$ $v \notin$*writes*(π i)› **using** *kddl* **unfolding** *is-ddi-def* **by** *auto*
have *σvk*: ‹(σ^k) $v = (\sigma^{Suc\ l}$ $)$ v› **using** *kddl ddi-value path*(*1*) **by** *auto*

show ‹*?thesis*› **proof** *cases*
assume *vnw'*: ‹$\forall$ $l < k'.$ $v \notin$*writes* (π' l)›
hence *σv'*: ‹$(\sigma'^{k'})$ $v = \sigma'$ v› **by** (*metis no-writes-unchanged0 path*(*2*))

have *notinπ'*: ‹$\neg$ ($\exists l'.$ cs^{π} $l = cs^{\pi'}$ l')› **proof**
assume ‹$\exists l'.$ cs^{π} $l = cs^{\pi'}$ l'›
then obtain l' **where** cs^{π} $l = cs^{\pi'}$ l' **..**
note *csl* = ‹cs^{π} $l = cs^{\pi'}$ l'›
have *lk*: ‹$l' < k'$› **using** *lk cseq ip cs-order*[*of* ‹π› ‹π'› ‹l› ‹l'› ‹k› ‹k'›] *csl nret* **by** *metis*

have ‹$v \in$ *writes* (π' l')› **using** *csl lv last-cs* **by** *metis*
thus ‹*False*› **using** *lk vnw'* **by** *blast*
qed

from *converged-cd-diverge*[*OF ip*(*1*,*2*) *π0*[*symmetric*] *notinπ' lk cseq*[*symmetric*]]
obtain i i' **where** *csi*: ‹$cs^{\pi'}$ $i' = cs^{\pi}$ i› **and** *lcdi*: ‹l $cd^{\pi}\!\rightarrow$ i› **and** *div*: ‹π (*Suc* i) $\neq \pi'$ (*Suc* i')› **by** *metis*

have *1*: ‹π (*Suc* i) = *suc* (π i) (σ^i)› **by** (*metis step-suc-sem fst-conv path*(*1*) *path-suc*)
have *2*: ‹π' (*Suc* i') = *suc* (π' i') $(\sigma'^{i'})$› **by** (*metis step-suc-sem fst-conv path*(*2*) *path-suc*)
have *3*: ‹π' $i' = \pi$ i› **using** *csi last-cs* **by** *metis*
have *nreads*: ‹σ^i ↾ *reads* (π i) $\neq \sigma'^{i'}$ ↾ *reads* (π i)› **by** (*metis 1 2 3 div reads-restr-suc*)
have *contri*: ‹(σ',i') $\mathfrak{c}$ (σ,i)› **using** *contradicts.intros*(*2*)[*OF csi path nreads*] **.**

have *nreti*: ‹π $i \neq$ *return*› **by** (*metis csi div ip*(*1*) *ip*(*2*) *last-cs lessI term-path-stable less-imp-le*)
have *ik*: ‹$i < k$› **using** *lcdi lk is-cdi-def* **by** *auto*
have *ik'*: ‹$i' < k'$› **using** *cs-order*[*OF ip*(*1*,*2*) *csi*[*symmetric*] *cseq*[*symmetric*] *nreti ik*] **.**

have *nreads*: ‹$\sigma^{i} \upharpoonright$ *reads* $(\pi\ i) \neq \sigma'^{i'} \upharpoonright$ *reads* $(\pi\ i)$› **by** (*metis 1 2 3 div reads-restr-suc*)
then obtain v' **where** *v'read*: ‹$v' \in$ *reads*(*path* σ i)› ‹$(\sigma^{i})\ v' \neq (\sigma'^{i'})\ v'$› **unfolding** *path* **by** (*metis reads-restrict*)

have *cpi*: ‹$((\sigma,\ i),\ (\sigma',\ i')) \in cp$› **using** *IH*[*of* ‹*i*› ‹*i'*›] *v'read csi ik ik' path* **by** *auto*
hence *cpi'*: ‹$((\sigma',\ i'),\ (\sigma,\ i)) \in cp$› **using** *cp.intros*(*4*) **by** *blast*

have *vnwi*: ‹$\forall j' \in \{LEAST\ i'a.\ i' < i'a \wedge (\exists\, i.\ cs^{path\ \sigma}\ \ i = cs^{path\ \sigma'}\ \ i'a)..<k'\}.\ v \notin$ *writes* (*path* σ' j')› **using** *vnw'*[*unfolded path*]
by (*metis* (*poly-guards-query*) *atLeastLessThan-iff*)

from *cp.intros*(*3*)[*OF cpi kddl*[*unfolded path*] *lcdi*[*unfolded path*] *csk div*[*unfolded path*] *vneq vnwi*]

show ‹*?thesis*› **using** *cp.intros*(*4*) **by** *simp*
next
assume ‹$\neg\ (\forall\ \ l<k'.\ v \notin$ *writes* $(\pi'\ l))$›
then obtain l' **where** *kddl'*: ‹$k'\ dd^{\pi',v} \rightarrow l'$› **using** *path*(*2*) *path-is-path written-read-dd vread'* **by** *blast*
hence *lv'*: ‹$v \in$ *writes* $(\pi'\ l')$› **unfolding** *is-ddi-def* **by** *auto*
have *lk'*: ‹$l' < k'$› **by** (*metis is-ddi-def kddl'*)
have *nretl'*: ‹$\pi'\ l' \neq$ *return*› **using** *lv' writes-return* **by** *auto*
have *nwb'*: ‹$\forall\ \ i' \in \{Suc\ l'..< k'\}.\ v \notin$ *writes*$(\pi'\ i')$› **using** *kddl'* **unfolding** *is-ddi-def* **by** *auto*
have *σvk'*: ‹$(\sigma'^{k'})\ v = (\sigma'^{Suc\ l'}\ )\ v$› **using** *kddl' ddi-value path*(*2*) **by** *auto*

show ‹*?thesis*› **proof** *cases*
assume *csl*: ‹$cs^{\pi}\ \ l = cs^{\pi'}\ \ l'$›
hence *πl*: ‹$\pi\ \ l = \pi'\ l'$› **by** (*metis last-cs*)
have *σvls*: ‹$(\sigma^{Suc\ l}\ )\ v \neq (\sigma'^{Suc\ l'}\ )\ v$› **by** (*metis σvk σvk' vneq*)
have *rσ*: ‹$\sigma^{l} \upharpoonright$ *reads* $(\pi\ \ l) \neq \sigma'^{l'} \upharpoonright$ *reads* $(\pi\ \ l)$› **using** *path πl σvls written-value lv* **by** *blast*
then obtain v' **where** *v'read*: ‹$v' \in$ *reads*(*path* σ l)› ‹$(\sigma^{l})\ v' \neq (\sigma'^{l'})\ v'$› **unfolding** *path* **by** (*metis reads-restrict*)

have *cpl*: ‹$((\sigma,\ l),\ (\sigma',\ l')) \in cp$› **using** *IH*[*of* ‹*l*› ‹*l'*›] *v'read csl lk lk' path* **by** *auto*
show ‹$((\sigma,\ k),\ (\sigma',\ k')) \in cp$› **using** *cp.intros*(*2*)[*OF cpl kddl*[*unfolded path*] *kddl'*[*unfolded path*] *csk vneq*] **.**
next
assume *csl*: ‹$cs^{\pi}\ \ l \neq cs^{\pi'}\ \ l'$›
show ‹*?thesis*› **proof** *cases*
assume ‹$\exists\ \ i'.\ cs^{\pi}\ \ l = cs^{\pi'}\ \ i'$›
then obtain i' **where** *csli'*: ‹$cs^{\pi}\ \ l = cs^{\pi'}\ \ i'$› **by** *blast*
have *ilne'*: ‹$i' \neq l'$› **using** *csl csli'* **by** *auto*
have *ij'*: ‹$i' < k'$› **using** *cs-order*[*OF ip csli' cseq*[*symmetric*] *nret lk*] **.**
have *iv'*: ‹$v \in$ *writes*$(\pi'\ i')$› **using** *lv csli' last-cs* **by** *metis*
have *il'*: ‹$i' < l'$› **using** *kddl' ilne' ij' iv'* **unfolding** *is-ddi-def* **by** *auto*
have *nreti'*: ‹$\pi'\ i' \neq$ *return*› **using** *csli' nret last-cs* **by** *metis*

have *l'notinπ*: ‹$\neg(\exists\, i.\ cs^{\pi'}\ \ l' = cs^{\pi}\ \ i\ )$› **proof**
assume ‹$\exists\, i.\ cs^{\pi'}\ \ l' = cs^{\pi}\ \ i$›
then obtain i **where** *csil*: ‹$cs^{\pi}\ \ i = cs^{\pi'}\ \ l'$› **by** *metis*
have *ik*: ‹$i < k$› **using** *cs-order*[*OF ip*(*2*,*1*) *csil*[*symmetric*] *cseq nretl' lk'*] **.**
have *li*: ‹$l < i$› **using** *cs-order*[*OF ip*(*2*,*1*) *csli'*[*symmetric*] *csil*[*symmetric*] *nreti' il'*] **.**
have *iv*: ‹$v \in$ *writes*$(\pi\ i)$› **using** *lv' csil last-cs* **by** *metis*
show ‹*False*› **using** *kddl ik li iv is-ddi-def* **by** *auto*

qed

obtain n n' **where** *csn*: ‹cs^{π} n = $cs^{\pi'}$ n'› **and** *lcdn'*: ‹l' $cd^{\pi'}\rightarrow$ n'› **and** *sucn*: ‹π (*Suc* n) $\neq$ π' (*Suc* n')› **and** *in'*: ‹$i' \leq n'$›
using *converged-cd-diverge-cs* [*OF ip*(*2*,*1*) *csli'*[*symmetric*] *il' l'notinπ lk' cseq*] **by** *metis*

— Can apply the IH to n and n'

have *1*: ‹π (*Suc* n) = *suc* (π n) (σ^{n})› **by** (*metis step-suc-sem fst-conv path*(*1*) *path-suc*)
have *2*: ‹π' (*Suc* n') = *suc* (π' n') ($\sigma'^{n'}$)› **by** (*metis step-suc-sem fst-conv path*(*2*) *path-suc*)
have *3*: ‹π' n' = π n› **using** *csn last-cs* **by** *metis*
have *nreads*: ‹σ^{n} ↾ *reads* (π n) $\neq$ $\sigma'^{n'}$ ↾ *reads* (π n)› **by** (*metis 1 2 3 sucn reads-restr-suc*)
then obtain v' **where** *v'read*: ‹v'∈*reads* (*path* σ n)› ‹(σ^{n}) v' $\neq$ ($\sigma'^{n'}$) v'› **by** (*metis path*(*1*) *reads-restrict*)
moreover
have *nl'*: ‹$n' < l'$› **using** *lcdn' is-cdi-def* **by** *auto*
have *nk'*: ‹$n' < k'$› **using** *nl' lk'* **by** *simp*
have *nretn'*: ‹π' n' $\neq$ *return*› **by** (*metis ip*(*2*) *nl' nretl' term-path-stable less-imp-le*)
have *nk*: ‹$n < k$› **using** *cs-order*[*OF ip*(*2*,*1*) *csn*[*symmetric*] *cseq nretn' nk'*] **.**
hence *lenn*: ‹$n+n' < k+k'$› **using** *nk'* **by** *auto*
ultimately
have ‹((σ, n), (σ', n')) ∈ *cp*› **using** *IH csn path* **by** *auto*
hence *ncp*: ‹((σ', n'), (σ, n)) ∈ *cp*› **using** *cp.intros*(*4*) **by** *auto*

have *nles*: ‹n < (*LEAST* i'. $n < i'$ ∧ (∃ i. $cs^{\pi'}$ i = cs^{π} i'))› (**is** ‹- < (*LEAST* i. *?P* i)›) **using** *nk cseq LeastI*[*of* ‹*?P*› ‹k›] **by** *metis*
moreover
have *ln*: ‹$l \leq n$› **using** *cs-order-le*[*OF ip*(*2*,*1*) *csli'*[*symmetric*] *csn*[*symmetric*] *nreti' in'*] **.**
ultimately
have *lles*: ‹*Suc* l ≤ (*LEAST* i'. $n < i'$ ∧ (∃ i. $cs^{\pi'}$ i = cs^{π} i'))› **by** *auto*

have *nwcseq*: ‹∀ j'∈{*LEAST* i'. $n < i'$ ∧ (∃ i. $cs^{\pi'}$ i = cs^{π} i')..<k}. v ∉ *writes* (π j')› **proof**
fix j' **assume** ∗: ‹j' ∈ {*LEAST* i'. $n < i'$ ∧ (∃ i. $cs^{\pi'}$ i = cs^{π} i')..<k}›
hence ‹(*LEAST* i'. $n < i'$ ∧ (∃ i. $cs^{\pi'}$ i = cs^{π} i')) ≤ j'› **by** (*metis* (*poly-guards-query*) *atLeast-LessThan-iff*)
hence ‹*Suc* $l \leq j'$› **using** *lles* **by** *auto*
moreover
have ‹$j' < k$› **using** ∗ **by** (*metis* (*poly-guards-query*) *atLeastLessThan-iff*)
ultimately have ‹j'∈ {*Suc* l..<k}› **by** (*metis* (*poly-guards-query*) *atLeastLessThan-iff*)
thus ‹v∉*writes* (π j')› **using** *nwb* **by** *auto*
qed

from *cp.intros*(*3*)[*OF ncp*,*folded path*,*OF kddl' lcdn' cseq sucn*[*symmetric*] *vneq*[*symmetric*] *nwcseq*]
have ‹((σ', k'), σ, k) ∈ *cp*› **.**
thus ‹((σ, k), (σ', k')) ∈ *cp*› **using** *cp.intros*(*4*) **by** *auto*
next
assume *lnotinπ'*: ‹¬ (∃ i'. cs^{π} l = $cs^{\pi'}$ i')›
show ‹*?thesis*› **proof** *cases*
assume ‹∃ i. cs^{π} i = $cs^{\pi'}$ l'›
then obtain i **where** *csli*: ‹cs^{π} i = $cs^{\pi'}$ l'› **by** *blast*
have *ilne*: ‹$i \neq l$› **using** *csl csli* **by** *auto*
have *ij*: ‹$i < k$› **using** *cs-order*[*OF ip*(*2*,*1*) *csli*[*symmetric*] *cseq nretl' lk'*] **.**
have *iv*: ‹v ∈ *writes*(π i)› **using** *lv' csli last-cs* **by** *metis*
have *il*: ‹$i < l$› **using** *kddl ilne ij iv* **unfolding** *is-ddi-def* **by** *auto*
have *nreti*: ‹π i $\neq$ *return*› **using** *csli nretl' last-cs* **by** *metis*

obtain *n n'* **where** *csn*: ‹cs^{π} n = $cs^{\pi'}$ n'› **and** *lcdn*: ‹l $cd^{\pi}\rightarrow$ n› **and** *sucn*: ‹π (*Suc n*) $\neq$ π' (*Suc n'*)› **and** *ilen*: ‹$i \leq n$›
using *converged-cd-diverge-cs* [*OF ip csli il lnotinπ' lk cseq*[*symmetric*]] **by** *metis*

— Can apply the IH to n and n'

have *1*: ‹π (*Suc n*) = *suc* (π n) (σ^n)› **by** (*metis step-suc-sem fst-conv path*(*1*) *path-suc*)
have *2*: ‹π' (*Suc n'*) = *suc* (π' n') ($\sigma'^{n'}$)› **by** (*metis step-suc-sem fst-conv path*(*2*) *path-suc*)
have *3*: ‹π' n' = π n› **using** *csn last-cs* **by** *metis*
have *nreads*: ‹σ^n ↾ *reads* (π n) $\neq$ $\sigma'^{n'}$ ↾ *reads* (π n)› **by** (*metis 1 2 3 sucn reads-restr-suc*)
then obtain *v'* **where** *v'read*: ‹*v'*∈*reads* (*path σ n*)› ‹(σ^n) *v'* $\neq$ ($\sigma'^{n'}$) *v'*› **by** (*metis path*(*1*) *reads-restrict*)
moreover
have *nl*: ‹$n < l$› **using** *lcdn is-cdi-def* **by** *auto*
have *nk*: ‹$n < k$› **using** *nl lk* **by** *simp*
have *nretn*: ‹π n $\neq$ *return*› **by** (*metis ip*(*1*) *nl nret term-path-stable less-imp-le*)
have *nk'*: ‹$n' < k'$› **using** *cs-order*[*OF ip csn cseq*[*symmetric*] *nretn nk*] **.**
hence *lenn*: ‹n+n' < k+k'› **using** *nk* **by** *auto*
ultimately
have *ncp*: ‹((σ, n), (σ', n')) ∈ *cp*› **using** *IH csn path* **by** *auto*

have *nles'*: ‹n' < (*LEAST* i'. $n' < i'$ ∧ (∃ i. cs^{π} i = $cs^{\pi'}$ i'))› (**is** ‹- < (*LEAST i. ?P i*)›) **using** *nk' cseq LeastI*[*of* ‹*?P*› ‹*k'*›] **by** *metis*
moreover
have *ln'*: ‹$l' \leq n'$› **using** *cs-order-le*[*OF ip csli csn nreti ilen*] **.**
ultimately
have *lles'*: ‹*Suc* l' ≤ (*LEAST* i'. $n' < i'$ ∧ (∃ i. cs^{π} i = $cs^{\pi'}$ i'))› **by** *auto*

have *nwcseq'*: ‹∀ j'∈{(*LEAST* i'. $n' < i'$ ∧ (∃ i. cs^{π} i = $cs^{\pi'}$ i'))..<k'}. $v \notin$ *writes* (π' j')› **proof**
fix j' **assume** ∗: ‹j' ∈ {(*LEAST* i'. $n' < i'$ ∧ (∃ i. cs^{π} i = $cs^{\pi'}$ i'))..<k'}›
hence ‹(*LEAST* i'. $n' < i'$ ∧ (∃ i. cs^{π} i = $cs^{\pi'}$ i')) ≤ j'› **by** (*metis* (*poly-guards-query*) *atLeastLessThan-iff*)
hence ‹*Suc* l' ≤ j'› **using** *lles'* **by** *auto*
moreover
have ‹$j' < k'$› **using** ∗ **by** (*metis* (*poly-guards-query*) *atLeastLessThan-iff*)
ultimately have ‹j'∈ {*Suc* l'..<k'}› **by** (*metis* (*poly-guards-query*) *atLeastLessThan-iff*)
thus ‹v∉*writes* (π' j')› **using** *nwb'* **by** *auto*
qed

from *cp.intros*(*3*)[*OF ncp,folded path*, *OF kddl lcdn cseq*[*symmetric*] *sucn vneq nwcseq'*]

show ‹((σ, k), (σ', k')) ∈ *cp*› **.**
next
assume *l'notinπ*: ‹¬ (∃ i. cs^{π} i = $cs^{\pi'}$ l')›
define *m* **where** ‹*m* ≡ *0*::*nat*›
define *m'* **where** ‹*m'* ≡ *0*::*nat*›
have *csm*: ‹cs^{π} m = $cs^{\pi'}$ m'› **unfolding** *m-def m'-def cs-0* **by** (*metis π0*)
have *ml*: ‹m<l ∨ m'<l'› **using** *csm csl* **unfolding** *m-def m'-def* **by** (*metis neq0-conv*)
have ‹∃ n n'. cs^{π} n = $cs^{\pi'}$ n' ∧ π (*Suc n*) $\neq$ π' (*Suc n'*) ∧
(l $cd^{\pi}\rightarrow$ n ∧ (∀ j'∈{(*LEAST* i'. $n' < i'$ ∧ (∃ i. cs^{π} i = $cs^{\pi'}$ i'))..<k'}. v∉*writes* (π' j'))
∨ l' $cd^{\pi'}\rightarrow$ n' ∧ (∀ j∈{(*LEAST* i. $n < i$ ∧ (∃ i'. $cs^{\pi'}$ i' = cs^{π} i))..<k}. v∉*writes* (π j)))›
using *csm ml* **proof** (*induction* ‹k+k'−(m+m')› *arbitrary*: ‹*m*› ‹*m'*› *rule*: *less-induct*)
case (*less m m'*)

note $csm = \langle cs^{\pi}\ m = cs^{\pi'}\ m' \rangle$
note $lm = \langle m < l \lor m' < l' \rangle$
note $IH = \langle \bigwedge n\ n'.$
$k + k' - (n + n') < k + k' - (m + m') \Longrightarrow$
$cs^{\pi}\ n = cs^{\pi'}\ n' \Longrightarrow$
$n < l \lor n' < l' \Longrightarrow ?thesis \rangle$
show $\langle ?thesis \rangle$ **using** *lm* **proof**
assume ml: $\langle m < l \rangle$
obtain $n\ n'$ **where** mn: $\langle m \leq n \rangle$ **and** csn: $\langle\ cs^{\pi}\ n = cs^{\pi'}\ n' \rangle$ **and** $lcdn$: $\langle l\ cd^{\pi}\!\rightarrow n \rangle$ **and** suc: $\langle \pi\ (Suc\ n) \neq \pi'\ (Suc\ n') \rangle$
using *converged-cd-diverge-cs*[*OF ip csm ml lnotin*π' *lk cseq*[*symmetric*]] **.**
have nl: $\langle n < l \rangle$ **using** *lcdn is-cdi-def* **by** *auto*
hence nk: $\langle n{<}k \rangle$ **using** *lk* **by** *auto*
have $nretn$: $\langle \pi\ n \neq return \rangle$ **using** *lcdn* **by** (*metis cd-not-ret*)
have nk': $\langle n'{<}k' \rangle$ **using** *cs-order*[*OF ip csn cseq*[*symmetric*] *nretn nk*] **.**
show $\langle ?thesis \rangle$ **proof** *cases*
assume $\langle \forall j'{\in}\{(LEAST\ i'.\ n' < i' \land (\exists i.\ cs^{\pi}\ i = cs^{\pi'}\ i'))..{<}k'\}.\ v{\notin}writes\ (\pi'\ j') \rangle$
thus $\langle ?thesis \rangle$ **using** *lcdn csn suc* **by** *blast*
next
assume $\langle \neg(\forall j'{\in}\{(LEAST\ i'.\ n' < i' \land (\exists i.\ cs^{\pi}\ i = cs^{\pi'}\ i'))..{<}k'\}.\ v{\notin}writes\ (\pi'\ j')) \rangle$
then obtain j' **where** jin': $\langle j'{\in}\{(LEAST\ i'.\ n' < i' \land (\exists i.\ cs^{\pi}\ i = cs^{\pi'}\ i'))..{<}k'\} \rangle$ **and** $vwrite$: $\langle v{\in}writes\ (\pi'\ j') \rangle$ **by** *blast*
define i' **where** $\langle i' \equiv LEAST\ i'.\ n' < i' \land (\exists i.\ cs^{\pi}\ i = cs^{\pi'}\ i') \rangle$
have Pk': $\langle n' < k' \land (\exists\ k.\ cs^{\pi}\ k = cs^{\pi'}\ k') \rangle$ (**is** $\langle ?P\ k' \rangle$) **using** nk' *cseq*[*symmetric*] **by** *blast*
have ni': $\langle n' < i' \rangle$ **using** *LeastI*[*of* $\langle ?P \rangle$, *OF* Pk'] i'*-def* **by** *auto*
obtain i **where** csi: $\langle cs^{\pi}\ i = cs^{\pi'}\ i' \rangle$ **using** *LeastI*[*of* $\langle ?P \rangle$, *OF* Pk'] i'*-def* **by** *blast*
have ij': $\langle i'{\leq}j' \rangle$ **using** jin'[*folded* i'*-def*] **by** *auto*
have jk': $\langle j'{<}k' \rangle$ **using** jin'[*folded* i'*-def*] **by** *auto*
have jl': $\langle j' \leq l' \rangle$ **using** $kddl'$ jk' *vwrite* **unfolding** *is-ddi-def* **by** *auto*
have $nretn'$: $\langle \pi'\ n' \neq return \rangle$ **using** *nretn csn last-cs* **by** *metis*
have iln: $\langle n{<}i \rangle$ **using** *cs-order*[*OF ip*(*2*,*1*) *csn*[*symmetric*] *csi*[*symmetric*] $nretn'$ ni'] **.**
hence mi: $\langle m < i \rangle$ **using** *mn* **by** *auto*
have $nretm$: $\langle \pi\ m \neq return \rangle$ **by** (*metis ip*(*1*) *mn nretn term-path-stable*)
have mi': $\langle m'{<}i' \rangle$ **using** *cs-order*[*OF ip csm csi nretm mi*] **.**
have ik': $\langle i' < k' \rangle$ **using** ij' jk' **by** *auto*
have $nreti'$: $\langle \pi'\ i' \neq return \rangle$ **by** (*metis* ij' jl' $nretl'$ *ip*(*2*) *term-path-stable*)
have ik: $\langle i < k \rangle$ **using** *cs-order*[*OF ip*(*2*,*1*) *csi*[*symmetric*] *cseq* $nreti'$ ik'] **.**
show $\langle ?thesis \rangle$ **proof** *cases*
assume il:$\langle i < l \rangle$
have le: $\langle k + k' - (i + i') < k{+}k' - (m{+}m') \rangle$ **using** *mi* mi' *ik* ik' **by** *auto*
show $\langle ?thesis \rangle$ **using** *IH*[*OF le*] **using** *csi il* **by** *blast*
next
assume $\langle \neg\ i < l \rangle$
hence li: $\langle l \leq i \rangle$ **by** *auto*
have $\langle i' \leq l' \rangle$ **using** ij' jl' **by** *auto*
hence il': $\langle i' < l' \rangle$ **using** *csi* $l'notin\pi$ **by** *fastforce*
obtain $n\ n'$ **where** in': $\langle i' \leq n' \rangle$ **and** csn: $\langle\ cs^{\pi}\ n = cs^{\pi'}\ n' \rangle$ **and** $lcdn'$: $\langle l'\ cd^{\pi'}\!\rightarrow n' \rangle$ **and** suc: $\langle \pi\ (Suc\ n) \neq \pi'\ (Suc\ n') \rangle$
using *converged-cd-diverge-cs*[*OF ip*(*2*,*1*) *csi*[*symmetric*] il' - lk' *cseq*] $l'notin\pi$ **by** *metis*
have nk': $\langle n' < k' \rangle$ **using** $lcdn'$ *is-cdi-def* lk' **by** *auto*
have $nretn'$: $\langle \pi'\ n' \neq return \rangle$ **by** (*metis cd-not-ret* $lcdn'$)
have nk: $\langle n < k \rangle$ **using** *cs-order*[*OF ip*(*2*,*1*) *csn*[*symmetric*] *cseq* $nretn'$ nk'] **.**
define j **where** $\langle j \equiv LEAST\ j.\ n < j \land (\exists j'.\ cs^{\pi'}\ j' = cs^{\pi}\ j) \rangle$
have Pk: $\langle n < k \land (\exists j'.\ cs^{\pi'}\ j' = cs^{\pi}\ k) \rangle$ (**is** $\langle ?P\ k \rangle$) **using** *nk cseq* **by** *blast*

have *nj*: ‹$n<j$› **using** *LeastI*[*of* ‹*?P*›, *OF Pk*] *j-def* **by** *auto*
have *ilen*: ‹$i \leq n$› **using** *cs-order-le*[*OF ip*(*2*,*1*) *csi*[*symmetric*] *csn*[*symmetric*] *nreti′ in′*] **.**
hence *lj*: ‹$l<j$› **using** *li nj* **by** *simp*
have ‹$\forall l \in \{l<..<k\}$. $v \notin$ *writes* $(\pi\ l)$› **using** *kddl* **unfolding** *is-ddi-def* **by** *simp*
hence *nw*: ‹$\forall l \in \{j..<k\}$. $v \notin$ *writes* $(\pi\ l)$› **using** *lj* **by** *auto*
show ‹*?thesis*› **using** *csn lcdn′ suc nw*[*unfolded j-def*] **by** *blast*
qed
qed
next
assume *ml′*: ‹$m' < l'$›
obtain *n n′* **where** *mn′*: ‹$m' \leq n'$› **and** *csn*: ‹ $cs^{\pi}\ n = cs^{\pi'}\ n'$› **and** *lcdn′*: ‹$l'\ cd^{\pi'}\!\rightarrow n'$› **and** *suc*: ‹$\pi\ (Suc\ n) \neq \pi'\ (Suc\ n')$›
using *converged-cd-diverge-cs*[*OF ip*(*2*,*1*) *csm*[*symmetric*] *ml′ - lk′ cseq*] *l′notinπ* **by** *metis*
have *nl′*: ‹$n' < l'$› **using** *lcdn′ is-cdi-def* **by** *auto*
hence *nk′*: ‹$n'<k'$› **using** *lk′* **by** *auto*
have *nretn′*: ‹$\pi'\ n' \neq$ *return*› **using** *lcdn′* **by** (*metis cd-not-ret*)
have *nk*: ‹$n<k$› **using** *cs-order*[*OF ip*(*2*,*1*) *csn*[*symmetric*] *cseq nretn′ nk′*] **.**
show ‹*?thesis*› **proof** *cases*
assume ‹$\forall j \in \{(LEAST\ i.\ n < i \wedge (\exists i'.\ cs^{\pi'}\ i' = cs^{\pi}\ i))..<k\}$. $v \notin$*writes* $(\pi\ j)$›
thus ‹*?thesis*› **using** *lcdn′ csn suc* **by** *blast*
next
assume ‹$\neg(\forall j \in \{(LEAST\ i.\ n < i \wedge (\exists i'.\ cs^{\pi'}\ i' = cs^{\pi}\ i))..<k\}$. $v \notin$*writes* $(\pi\ j))$›
then obtain *j* **where** *jin*: ‹$j \in \{(LEAST\ i.\ n < i \wedge (\exists i'.\ cs^{\pi'}\ i' = cs^{\pi}\ i))..<k\}$› **and** *vwrite*: ‹$v \in$*writes* $(\pi\ j)$› **by** *blast*
define *i* **where** ‹$i \equiv LEAST\ i.\ n < i \wedge (\exists i'.\ cs^{\pi'}\ i' = cs^{\pi}\ i)$›
have *Pk*: ‹$n < k \wedge (\exists\ k'.\ cs^{\pi'}\ k' = cs^{\pi}\ k)$› (**is** ‹*?P k*›) **using** *nk cseq* **by** *blast*
have *ni*: ‹$n < i$› **using** *LeastI*[*of* ‹*?P*›, *OF Pk*] *i-def* **by** *auto*
obtain *i′* **where** *csi*: ‹$cs^{\pi}\ i = cs^{\pi'}\ i'$› **using** *LeastI*[*of* ‹*?P*›, *OF Pk*] *i-def* **by** *metis*
have *ij*: ‹$i \leq j$› **using** *jin*[*folded i-def*] **by** *auto*
have *jk*: ‹$j<k$› **using** *jin*[*folded i-def*] **by** *auto*
have *jl*: ‹$j \leq l$› **using** *kddl jk vwrite* **unfolding** *is-ddi-def* **by** *auto*
have *nretn*: ‹$\pi\ n \neq$ *return*› **using** *nretn′ csn last-cs* **by** *metis*
have *iln′*: ‹$n'<i'$› **using** *cs-order*[*OF ip csn csi nretn ni*] **.**
hence *mi′*: ‹$m' < i'$› **using** *mn′* **by** *auto*
have *nretm′*: ‹$\pi'\ m' \neq$ *return*› **by** (*metis ip*(*2*) *mn′ nretn′ term-path-stable*)
have *mi*: ‹$m<i$› **using** *cs-order*[*OF ip*(*2*,*1*) *csm*[*symmetric*] *csi*[*symmetric*] *nretm′ mi′*] **.**
have *ik*: ‹$i < k$› **using** *ij jk* **by** *auto*
have *nreti*: ‹$\pi\ i \neq$ *return*› **by** (*metis ij ip*(*1*) *jl nret term-path-stable*)
have *ik′*: ‹$i' < k'$› **using** *cs-order*[*OF ip csi cseq*[*symmetric*] *nreti ik*] **.**
show ‹*?thesis*› **proof** *cases*
assume *il′*:‹$i' < l'$›
have *le*: ‹$k + k' - (i + i') < k + k' - (m + m')$› **using** *mi mi′ ik ik′* **by** *auto*
show ‹*?thesis*› **using** *IH*[*OF le*] **using** *csi il′* **by** *blast*
next
assume ‹$\neg\ i' < l'$›
hence *li′*: ‹$l' \leq i'$› **by** *auto*
have ‹$i \leq l$› **using** *ij jl* **by** *auto*
hence *il*: ‹$i < l$› **using** *csi lnotinπ′* **by** *fastforce*
obtain *n n′* **where** *ilen*: ‹$i \leq n$› **and** *csn*: ‹ $cs^{\pi}\ n = cs^{\pi'}\ n'$› **and** *lcdn*: ‹$l\ cd^{\pi}\!\rightarrow n$› **and** *suc*: ‹$\pi\ (Suc\ n) \neq \pi'\ (Suc\ n')$›
using *converged-cd-diverge-cs*[*OF ip csi il - lk cseq*[*symmetric*]] *lnotinπ′* **by** *metis*
have *nk*: ‹$n < k$› **using** *lcdn is-cdi-def lk* **by** *auto*
have *nretn*: ‹$\pi\ n \neq$ *return*› **by** (*metis cd-not-ret lcdn*)
have *nk′*: ‹$n' < k'$› **using** *cs-order*[*OF ip csn cseq*[*symmetric*] *nretn nk*] **.**
define *j′* **where** ‹$j' \equiv LEAST\ j'.\ n' < j' \wedge (\exists j.\ cs^{\pi}\ j = cs^{\pi'}\ j')$›

have Pk': ‹$n' < k' \wedge (\exists j.\ cs^{\pi}\ j = cs^{\pi'}\ k')$› (**is** ‹$?P\ k'$›) **using** nk' $cseq[symmetric]$ **by** $blast$
have nj': ‹$n'<j'$› **using** $LeastI[of$ ‹$?P$›, $OF\ Pk']$ j'-def **by** $auto$
have in': ‹$i' \leq n'$› **using** cs-$order$-$le[OF\ ip\ csi\ csn\ nreti\ ilen]$ **.**
hence lj': ‹$l'<j'$› **using** li' nj' **by** $simp$
have ‹$\forall l\in\{l'<..<k'\}.\ v \notin writes\ (\pi'\ l)$› **using** $kddl'$ **unfolding** is-ddi-def **by** $simp$
hence nw': ‹$\forall l\in\{j'..<k'\}.\ v \notin writes\ (\pi'\ l)$› **using** lj' **by** $auto$
show ‹$?thesis$› **using** $csn\ lcdn\ suc\ nw'[unfolded\ j'$-$def]$ **by** $blast$
qed
qed
qed
qed
then obtain $n\ n'$ **where** csn: ‹ $cs^{\pi}\ n = cs^{\pi'}\ n'$› **and** suc: ‹$\pi\ (Suc\ n) \neq \pi'\ (Suc\ n')$›
and $cdor$:
‹$(l\ cd^{\pi}\rightarrow\ n \wedge (\forall j'\in\{(LEAST\ i'.\ n' < i' \wedge (\exists i.\ cs^{\pi}\ i = cs^{\pi'}\ i'))..<k'\}.\ v\notin writes\ (\pi'\ j'))$
$\vee\ l'\ cd^{\pi'}\rightarrow\ n' \wedge (\forall j\in\{(LEAST\ i.\ n < i \wedge (\exists i'.\ cs^{\pi'}\ i' = cs^{\pi}\ i))..<k\}.\ v\notin writes\ (\pi\ j)))$›
by $blast$
show ‹$?thesis$› **using** $cdor$ **proof**
assume $*$: ‹$l\ cd^{\pi}\rightarrow\ n \wedge (\forall j'\in\{LEAST\ i'.\ n' < i' \wedge (\exists i.\ cs^{\pi}\ i = cs^{\pi'}\ i')..<k'\}.\ v \notin local.writes$ $(\pi'\ j'))$›
hence $lcdn$: ‹$l\ cd^{\pi}\rightarrow\ n$› **by** $blast$
have $nowrite$: ‹$\forall j'\in\{LEAST\ i'.\ n' < i' \wedge (\exists i.\ cs^{\pi}\ i = cs^{\pi'}\ i')..<k'\}.\ v \notin local.writes\ (\pi'\ j')$› **using** $*$ **by** $blast$
show ‹$?thesis$› **proof** ($rule\ cp.intros(3)[of$ ‹σ› ‹n› ‹σ'› ‹n'›,$folded\ path]$)
show ‹$l\ cd^{\pi}\rightarrow\ n$› **using** $lcdn$ **.**
show ‹$k\ dd^{\pi,v}\rightarrow\ l$› **using** $kddl$ **.**
show ‹$cs^{\pi}\ k = cs^{\pi'}\ k'$› **using** $cseq$ **by** $simp$
show ‹$\pi\ (Suc\ n) \neq \pi'\ (Suc\ n')$› **using** suc **by** $simp$
show ‹$\forall j'\in\{LEAST\ i'.\ n' < i' \wedge (\exists i.\ cs^{\pi}\ i = cs^{\pi'}\ i')..<k'\}.\ v \notin local.writes\ (\pi'\ j')$› **using** $nowrite$ **.**
show ‹$(\sigma^{k})\ v \neq (\sigma'^{k'})\ v$› **using** $vneq$ **.**
have nk: ‹$n < k$› **using** $lcdn\ lk\ is$-cdi-def **by** $auto$
have $nretn$: ‹$\pi\ n \neq return$› **using** cd-not-$ret\ lcdn$ **by** $metis$
have nk': ‹$n' < k'$› **using** cs-$order[OF\ ip\ csn\ cseq[symmetric]\ nretn\ nk]$ **.**
hence le: ‹$n + n' < k + k'$› **using** nk **by** $auto$
moreover
have 1: ‹$\pi\ (Suc\ n) = suc\ (\pi\ n)\ (\sigma^{n})$› **by** ($metis\ step$-$suc$-$sem\ fst$-$conv\ path(1)\ path$-$suc$)
have 2: ‹$\pi'\ (Suc\ n') = suc\ (\pi'\ n')\ (\sigma'^{n'})$› **by** ($metis\ step$-$suc$-$sem\ fst$-$conv\ path(2)\ path$-$suc$)
have 3: ‹$\pi'\ n' = \pi\ n$› **using** $csn\ last$-cs **by** $metis$
have $nreads$: ‹$\sigma^{n} \restriction reads\ (\pi\ n) \neq \sigma'^{n'} \restriction reads\ (\pi\ n)$› **by** ($metis\ 1\ 2\ 3\ suc\ reads$-$restr$-$suc$)
then obtain v' **where** $v'read$: ‹$v'\in reads\ (path\ \sigma\ n)$› ‹$(\sigma^{n})\ v' \neq (\sigma'^{n'})\ v'$› **by** ($metis\ path(1)$ $reads$-$restrict$)
ultimately
show ‹$((\sigma, n), (\sigma', n')) \in cp$› **using** $IH\ csn\ path$ **by** $auto$
qed
next
assume $*$: ‹$l'\ cd^{\pi'}\rightarrow\ n' \wedge (\forall j\in\{(LEAST\ i.\ n < i \wedge (\exists i'.\ cs^{\pi'}\ i' = cs^{\pi}\ i))..<k\}.\ v\notin writes\ (\pi\ j))$›
hence $lcdn'$: ‹$l'\ cd^{\pi'}\rightarrow\ n'$› **by** $blast$
have $nowrite$: ‹$\forall j\in\{(LEAST\ i.\ n < i \wedge (\exists i'.\ cs^{\pi'}\ i' = cs^{\pi}\ i))..<k\}.\ v\notin writes\ (\pi\ j)$› **using** $*$ **by** $blast$
show ‹$?thesis$› **proof** ($rule\ cp.intros(4)$, $rule\ cp.intros(3)[of$ ‹σ'› ‹n'› ‹σ› ‹n›,$folded\ path]$)
show ‹$l'\ cd^{\pi'}\rightarrow\ n'$› **using** $lcdn'$ **.**
show ‹$k'\ dd^{\pi',v}\rightarrow\ l'$› **using** $kddl'$ **.**
show ‹$cs^{\pi'}\ k' = cs^{\pi}\ k$› **using** $cseq$ **.**

show ‹π' (*Suc* n') $\neq$ π (*Suc* n)› **using** *suc* **by** *simp*
show ‹$\forall j \in \{(LEAST\ i.\ n < i \wedge (\exists i'.\ cs^{\pi'}\ i' = cs^{\pi}\ i))..<k\}.\ v \notin writes\ (\pi\ j)$› **using** *nowrite* **.**
show ‹$(\sigma'^{k'})\ v \neq (\sigma^{k})\ v$› **using** *vneq* **by** *simp*
have *nk'*: ‹$n' < k'$› **using** *lcdn' lk' is-cdi-def* **by** *auto*
have *nretn'*: ‹$\pi'\ n' \neq return$› **using** *cd-not-ret lcdn'* **by** *metis*
have *nk*: ‹$n < k$› **using** *cs-order*[*OF ip*(*2*,*1*) *csn*[*symmetric*] *cseq nretn' nk'*] **.**
hence *le*: ‹$n + n' < k + k'$› **using** *nk'* **by** *auto*
moreover
have *1*: ‹π (*Suc* n) = *suc* (π n) (σ^{n})› **by** (*metis step-suc-sem fst-conv path*(*1*) *path-suc*)
have *2*: ‹π' (*Suc* n') = *suc* (π' n') ($\sigma'^{m'}$)› **by** (*metis step-suc-sem fst-conv path*(*2*) *path-suc*)
have *3*: ‹$\pi'\ n' = \pi\ n$› **using** *csn last-cs* **by** *metis*
have *nreads*: ‹$\sigma^{n} \upharpoonright reads\ (\pi\ n) \neq \sigma'^{m'} \upharpoonright reads\ (\pi\ n)$› **by** (*metis 1 2 3 suc reads-restr-suc*)
then obtain v' **where** *v'read*: ‹$v' \in reads\ (path\ \sigma\ n)$› ‹$(\sigma^{n})\ v' \neq (\sigma'^{m'})\ v'$› **by** (*metis path*(*1*) *reads-restrict*)
ultimately
have ‹((σ, n), (σ', n')) $\in$ *cp*› **using** *IH csn path* **by** *auto*
thus ‹((σ', n'), σ, n) $\in$ *cp*› **using** *cp.intros*(*4*) **by** *simp*
qed
qed
qed
qed
qed
qed
qed
qed

theorem *contradicting-in-cop*: **assumes** ‹$\sigma =_L \sigma'$› **and** ‹$(\sigma',k')\ \mathfrak{c}\ (\sigma,k)$› **and** ‹*path* σ k $\in$ *dom att*›
shows ‹$((\sigma,k),\sigma',k') \in cop$› **using** *assms*(*2*) **proof**(*cases*)
case (*1* π' π)
define j **where** ‹$j \equiv \pi$ ¡ $cs^{\pi'}\ k'$›
have *csj*: ‹$cs^{\pi}\ j = cs^{\pi'}\ k'$› **unfolding** *j-def* **using** *1* **by** (*metis cs-not-nil cs-select-is-cs*(*1*) *path-is-path*)
have *suc*: ‹π (*Suc* j) $\neq$ π' (*Suc* k')› **using** *1 j-def* **by** *simp*
have *kcdj*: ‹$k\ cd^{\pi} \rightarrow j$› **by** (*metis cs-not-nil cs-select-is-cs*(*2*) *1*(*1*,*2*) *j-def path-is-path*)
obtain v **where** *readv*: ‹$v \in reads(path\ \sigma\ j)$› **and** *vneq*: ‹$(\sigma^{j})\ v \neq (\sigma'^{k'})\ v$› **using** *suc csj* **unfolding** *1* **by** (*metis IFC-def.suc-def 1*(*2*) *1*(*3*) *last-cs path-suc reads-restr-suc reads-restrict*)
have ‹$((\sigma,j),\sigma',k') \in cp$› **apply** (*rule contradicting-in-cp*[*OF assms*(*1*)]) **using** *readv vneq csj 1* **by** *auto*
thus ‹$((\sigma,k),\sigma',k') \in cop$› **using** *kcdj suc assms*(*3*) *cop.intros*(*2*) **unfolding** *1* **by** *auto*
next
case (*2* π' π)
obtain v **where** *readv*: ‹$v \in reads(path\ \sigma\ k)$› **and** *vneq*: ‹$(\sigma^{k})\ v \neq (\sigma'^{k'})\ v$› **using** *2*(*2*−*4*) **by** (*metis reads-restrict*)
have ‹$((\sigma,k),\sigma',k') \in cp$› **apply** (*rule contradicting-in-cp*[*OF assms*(*1*)]) **using** *readv vneq 2* **by** *auto*
thus ‹$((\sigma,k),\sigma',k') \in cop$› **using** *assms*(*3*) *cop.intros*(*1*) **unfolding** *2* **by** *auto*
qed

theorem *cop-correct-term*: **fixes** σ σ' **defines** π: ‹$\pi \equiv path\ \sigma$› **and** π': ‹$\pi' \equiv path\ \sigma'$›
assumes *ret*: ‹$\pi\ n = return$› ‹$\pi'\ n' = return$› **and** *obsne*: ‹$obs\ \sigma \neq obs\ \sigma'$› **and** *leq*: ‹$\sigma =_L \sigma'$›
shows ‹$\exists\ k\ k'.\ ((\sigma,k),\sigma',k') \in cop \vee ((\sigma',k'),\sigma,k) \in cop$›
proof −
have $*$: ‹$\exists\ k\ k'.\ ((\sigma',\ k')\ \mathfrak{c}\ (\sigma\ ,k) \wedge \pi\ k \in dom\ (att)) \vee ((\sigma,\ k)\ \mathfrak{c}\ (\sigma'\ ,k') \wedge \pi'\ k' \in dom\ (att))$› **using** *obs-neq-contradicts-term ret obsne* π π' **by** *auto*
have *leq'* :‹$\sigma' =_L \sigma$› **using** *leq* **unfolding** *loweq-def* **by** *auto*

from $*$ *contradicting-in-cop*[*OF leq*] *contradicting-in-cop*[*OF leq'*] **show** ‹*?thesis*› **unfolding** π π' **by** *metis*
qed

theorem *cop-correct-ret*: **fixes** σ σ' **defines** π: ‹$\pi \equiv path\ \sigma$› **and** π': ‹$\pi' \equiv path\ \sigma'$›
assumes *ret*: ‹$\pi\ n = return$› **and** *obsne*: ‹$obs\ \sigma\ i \neq obs\ \sigma'\ i$› **and** *obs*: ‹$obs\ \sigma'\ i \neq None$› **and** *leq*: ‹$\sigma =_L \sigma'$›
shows ‹$\exists\ k\ k'.\ ((\sigma,k),\sigma',k') \in cop \lor ((\sigma',k'),\sigma,k) \in cop$›
proof −
  have $*$: ‹$\exists\ k\ k'.\ ((\sigma',\ k')\ \mathfrak{c}\ (\sigma\ ,k) \land \pi\ k \in dom\ (att)) \lor ((\sigma,\ k)\ \mathfrak{c}\ (\sigma'\ ,k') \land \pi'\ k' \in dom\ (att))$›
    by (*metis* (*no-types*, *lifting*) π π' *obs obs-neq-ret-contradicts obsne ret*)
  have *leq'* :‹$\sigma' =_L \sigma$› **using** *leq* **unfolding** *loweq-def* **by** *auto*
  from $*$ *contradicting-in-cop*[*OF leq*] *contradicting-in-cop*[*OF leq'*] **show** ‹*?thesis*› **unfolding** π π' **by** *metis*
qed

theorem *cop-correct-nterm*: **assumes** *obsne*: ‹$obs\ \sigma\ k \neq obs\ \sigma'\ k$› ‹$obs\ \sigma\ k \neq None$› ‹$obs\ \sigma'\ k \neq None$›
and *leq*: ‹$\sigma =_L \sigma'$›
shows ‹$\exists\ k\ k'.\ ((\sigma,k),\sigma',k') \in cop \lor ((\sigma',k'),\sigma,k) \in cop$›
proof −
  obtain k k' **where** ‹$((\sigma',\ k')\ \mathfrak{c}\ (\sigma\ ,k) \land path\ \sigma\ k \in dom\ att) \lor ((\sigma,\ k)\ \mathfrak{c}\ (\sigma'\ ,k') \land path\ \sigma'\ k' \in dom\ att)$›
  using *obs-neq-some-contradicts*[*OF obsne*] **by** *metis*
  thus ‹*?thesis*› **proof**
    assume $*$: ‹$(\sigma',\ k')\ \mathfrak{c}\ (\sigma\ ,k) \land path\ \sigma\ k \in dom\ att$›
    hence ‹$((\sigma,k),\sigma',k') \in cop$› **using** *leq* **by** (*metis contradicting-in-cop*)
    thus ‹*?thesis*› **using** $*$ **by** *blast*
  next
    assume $*$: ‹$(\sigma,\ k)\ \mathfrak{c}\ (\sigma'\ ,k') \land path\ \sigma'\ k' \in dom\ att$›
    hence ‹$((\sigma',k'),\sigma,k) \in cop$› **using** *leq* **by** (*metis contradicting-in-cop loweq-def*)
    thus ‹*?thesis*› **using** $*$ **by** *blast*
  qed
qed

2.10 Correctness of the Characterisation

The following is our main correctness result. If there exist no critical observable paths, then the program is secure.

theorem *cop-correct*: **assumes** ‹$cop = empty$› **shows** ‹*secure*› **proof** (*rule ccontr*)
  assume ‹$\neg$ *secure*›
  then obtain σ σ' **where** *leq*: ‹ $\sigma =_L \sigma'$›
    and $**$: ‹$\neg\ obs\ \sigma \approx obs\ \sigma' \lor (terminates\ \sigma \land \neg\ obs\ \sigma' \lesssim obs\ \sigma)$›
    unfolding *secure-def* **by** *blast*
  show ‹*False*› **using** $**$ **proof**
    assume ‹$\neg\ obs\ \sigma \approx obs\ \sigma'$›
    then obtain k **where** ‹$obs\ \sigma\ k \neq obs\ \sigma'\ k \land obs\ \sigma\ k \neq None \land obs\ \sigma'\ k \neq None$›
      unfolding *obs-comp-def obs-prefix-def*
      by (*metis kth-obs-stable linorder-neqE-nat obs-none-no-kth-obs obs-some-kth-obs*)
    thus ‹*False*› **using** *cop-correct-nterm leq assms* **by** *auto*
  next
    assume $*$: ‹$terminates\ \sigma \land \neg\ obs\ \sigma' \lesssim obs\ \sigma$›
    then obtain n **where** *ret*: ‹$path\ \sigma\ n = return$›
      unfolding *terminates-def* **by** *auto*
    obtain k **where** ‹$obs\ \sigma\ k \neq obs\ \sigma'\ k \land obs\ \sigma'\ k \neq None$› **using** $*$ **unfolding** *obs-prefix-def* **by** *metis*
    thus ‹*False*› **using** *cop-correct-ret ret leq assms* **by** (*metis empty-iff*)
  qed
qed

Our characterisation is not only correct, it is also precise in the way that *cp* characterises exactly the matching indices in executions for low equivalent input states where diverging data is read. This follows easily as the inverse implication to lemma *contradicting-in-cp* can be shown by simple induction.

theorem *cp-iff-reads-contradict*: ‹$((\sigma,k),(\sigma',k')) \in cp \longleftrightarrow \sigma =_L \sigma' \wedge cs^{path\ \sigma}\ k = cs^{path\ \sigma'}\ k' \wedge (\exists\ v{\in}reads(path\ \sigma\ k).\ (\sigma^k)\ v \neq (\sigma'^{k'})\ v)$›
proof
  assume ‹$\sigma =_L \sigma' \wedge cs^{path\ \sigma}\ k = cs^{path\ \sigma'}\ k' \wedge (\exists\ v{\in}reads\ (path\ \sigma\ k).\ (\sigma^k)\ v \neq (\sigma'^{k'})\ v)$›
  thus ‹$((\sigma,\ k),\ \sigma',\ k') \in cp$› **using** *contradicting-in-cp* **by** *blast*
next
  assume ‹$((\sigma,\ k),\ \sigma',\ k') \in cp$›
  thus ‹$\sigma =_L \sigma' \wedge cs^{path\ \sigma}\ k = cs^{path\ \sigma'}\ k' \wedge (\exists\ v{\in}reads\ (path\ \sigma\ k).\ (\sigma^k)\ v \neq (\sigma'^{k'})\ v)$›
  proof (*induction*)
    case (*1 σ σ′ n n′ h*)
    then show ‹*?case*› **by** *blast*
  next
    case (*2 σ k σ′ k′ n v n′*)
    have ‹$v{\in}reads\ (path\ \sigma\ n)$› **using** *2(2)* **unfolding** *is-ddi-def* **by** *auto*
    then show ‹*?case*› **using** *2* **by** *auto*
  next
    case (*3 σ k σ′ k′ n v l n′*)
    have ‹$v{\in}reads\ (path\ \sigma\ n)$› **using** *3(2)* **unfolding** *is-ddi-def* **by** *auto*
    then show ‹*?case*› **using** *3(4,6,8)* **by** *auto*
  next
    case (*4 σ k σ′ k′*)
    hence ‹$cs^{path\ \sigma}\ k = cs^{path\ \sigma'}\ k'$› **by** *simp*
    hence ‹$path\ \sigma'\ k' = path\ \sigma\ k$› **by** (*metis last-cs*)
    moreover have ‹$\sigma' =_L \sigma$› **using** *4(2)* **unfolding** *loweq-def* **by** *simp*
    ultimately show ‹*?case*› **using** *4* **by** *metis*
  qed
qed

In the same way the inverse implication to *contradicting-in-cop* follows easily such that we obtain the following characterisation of *cop*.

theorem *cop-iff-contradicting*: ‹$((\sigma,k),(\sigma',k')) \in cop \longleftrightarrow \sigma =_L \sigma' \wedge (\sigma',k')\ \mathfrak{c}\ (\sigma,k) \wedge path\ \sigma\ k \in dom\ att$›
proof
  assume ‹$\sigma =_L \sigma' \wedge (\sigma',\ k')\ \mathfrak{c}\ (\sigma,\ k) \wedge path\ \sigma\ k \in dom\ att$› **thus** ‹$((\sigma,k),(\sigma',k')) \in cop$› **using** *contradicting-in-cop* **by** *simp*
next
  assume ‹$((\sigma,k),(\sigma',k')) \in cop$›
  thus ‹ $\sigma =_L \sigma' \wedge (\sigma',k')\ \mathfrak{c}\ (\sigma,k) \wedge path\ \sigma\ k \in dom\ att$› **proof** (*cases rule*: *cop.cases*)
    case *1*
    then show ‹*?thesis*› **using** *cp-iff-reads-contradict contradicts.simps* **by** (*metis* (*full-types*) *reads-restrict1*)
  next
    case (*2 k*)
    then show ‹*?thesis*› **using** *cp-iff-reads-contradict contradicts.simps*
      by (*metis cd-is-cs-less cd-not-ret contradicts.intros*(*1*) *cs-select-id path-is-path*)
  qed
qed

2.11 Correctness of the Single Path Approximation

theorem *cp-in-scp*: **assumes** ‹$((\sigma,k),(\sigma',k')){\in}cp$› **shows** ‹$(path\ \sigma,k){\in}scp \wedge (path\ \sigma',k'){\in}scp$›
using *assms* **proof**(*induction* ‹σ› ‹k› ‹σ′› ‹k′› *rule*:*cp.induct*[*case-names read-high dd dcd sym*])
  case (*read-high σ σ′ k k′ h*)

have ‹$\sigma\ h = (\sigma^k)\ h$› **using** *read-high*(5) **by** (*simp add*: *no-writes-unchanged0*)
moreover have ‹$\sigma'\ h = (\sigma'^{k'})\ h$› **using** *read-high*(6) **by** (*simp add*: *no-writes-unchanged0*)
ultimately have ‹$\sigma\ h \neq \sigma'\ h$› **using** *read-high*(4) **by** *simp*
hence ∗: ‹*h*∈*hvars*› **using** *read-high*(1) **unfolding** *loweq-def* **by** (*metis Compl-iff IFC-def.restrict-def*)
have *1*: ‹(*path* σ,*k*)∈*scp*› **using** *scp.intros*(1) *read-high*(3,5) ∗ **by** *auto*
have ‹*path* σ *k* = *path* σ′ *k*′› **using** *read-high*(2) **by** (*metis last-cs*)
hence ‹(*path* σ′,*k*′)∈*scp*› **using** *scp.intros*(1) *read-high*(3,6) ∗ **by** *auto*
thus ‹*?case*› **using** *1* **by** *auto*
next
case *dd* **show** ‹*?case*› **using** *scp.intros*(3) *dd* **by** *auto*
next
case *sym* **thus** ‹*?case*› **by** *blast*
next
case (*dcd* σ *k* σ′ *k*′ *n v l n*′)
note *scp.intros*(4) *is-dcdi-via-def cd-cs-swap cs-ipd*
have *1*: ‹(*path* σ, *n*)∈*scp*› **using** *dcd.IH dcd.hyps*(2) *dcd.hyps*(3) *scp.intros*(2) *scp.intros*(3) **by** *blast*
have *csk*: ‹$cs^{path\ \sigma}\ k = cs^{path\ \sigma'}\ k'$› **using** *cp-eq-cs*[*OF dcd*(1)] **.**
have *kn*: ‹*k*<*n*› **and** *kl*: ‹*k*<*l*› **and** *ln*: ‹*l*<*n*› **using** *dcd*(2,3) **unfolding** *is-ddi-def is-cdi-def* **by** *auto*
have *nret*: ‹*path* σ *k* ≠ *return*› **using** *cd-not-ret dcd.hyps*(3) **by** *auto*
have ‹*k*′ < *n*′› **using** *kn csk dcd*(4) *cs-order nret path-is-path last-cs* **by** *blast*
have *2*: ‹(*path* σ′, *n*′)∈*scp*› **proof** *cases*
assume *j*′*ex*: ‹∃*j*′∈{*k*′..<*n*′}. *v* ∈ *writes* (*path* σ′ *j*′)›
hence ‹∃*j*′. *j*′∈{*k*′..<*n*′} ∧ *v* ∈ *writes* (*path* σ′ *j*′)› **by** *auto*
note ∗ = *GreatestI-ex-nat*[*OF this*]
define *j*′ **where** ‹*j*′ == *GREATEST j*′. *j*′∈{*k*′..<*n*′} ∧ *v* ∈ *writes* (*path* σ′ *j*′)›
note ∗∗ = ∗[*of* ‹*j*′›,*folded j*′*-def*]
have ‹*k*′ ≤ *j*′› ‹*j*′<*n*′› **and** *j*′*write*: ‹*v* ∈ *writes* (*path* σ′ *j*′)›
using ∗ *atLeastLessThan-iff j*′*-def nat-less-le* **by** *auto*
have *nowrite*: ‹∀ *i*′∈{*j*′<..<*n*′}. *v* ∉ *writes*(*path* σ′ *i*′)› **proof** (*rule*, *rule ccontr*)
fix *i*′ **assume** ‹*i*′ ∈ {*j*′<..<*n*′}› ‹¬ *v* ∉ *local.writes* (*path* σ′ *i*′)›
hence ‹*i*′ ∈ {*k*′..<*n*′} ∧ *v* ∈ *local.writes* (*path* σ′ *i*′)› **using** ‹*k*′ ≤ *j*′› **by** *auto*
hence ‹*i*′ ≤ *j*′› **using** *Greatest-le-nat*
by (*metis* (*no-types*, *lifting*) *atLeastLessThan-iff j*′*-def nat-less-le*)
thus ‹*False*› **using** ‹*i*′ ∈ {*j*′<..<*n*′}› **by** *auto*
qed
have ‹*path* σ′ *n*′ = *path* σ *n*› **using** *dcd*(4) *last-cs* **by** *metis*
hence ‹*v*∈*reads*(*path* σ′ *n*′)› **using** *dcd*(2) **unfolding** *is-ddi-def* **by** *auto*
hence *nddj*′: ‹$n'\ dd^{path\ \sigma',v}\!\rightarrow j'$› **using** *dcd*(2) **unfolding** *is-ddi-def* **using** *nowrite* ‹*j*′<*n*′› *j*′*write* **by** *auto*
show ‹*?thesis*› **proof** *cases*
assume ‹$j'\ cd^{path\ \sigma'}\!\rightarrow k'$›
thus ‹(*path* σ′,*n*′) ∈ *scp*› **using** *scp.intros*(2) *scp.intros*(3) *dcd.IH nddj*′ **by** *fast*
next
assume *jcdk*′: ‹$\neg\ j'\ cd^{path\ \sigma'}\!\rightarrow k'$›
show ‹*?thesis*› **proof** *cases*
assume ‹*j*′ = *k*′›
thus ‹*?thesis*› **using** *scp.intros*(3) *dcd.IH nddj*′ **by** *fastforce*
next
assume ‹*j*′ ≠ *k*′› **hence** ‹*k*′ < *j*′› **using** ‹*k*′ ≤ *j*′› **by** *auto*
have ‹*path* σ′ *j*′ ≠ *return*› **using** *j*′*write writes-return* **by** *auto*
hence *ipdex*′:‹∃*j*. *j* ∈{*k*′..*j*′} ∧ *path* σ′ *j* = *ipd* (*path* σ′ *k*′) › **using** *path-is-path* ‹*k*′ < *j*′› *jcdk*′ *is-cdi-def* **by** *blast*
define *i*′ **where** ‹*i*′ == *LEAST j*. *j*∈ {*k*′..*j*′} ∧ *path* σ′ *j* = *ipd* (*path* σ′ *k*′)›
have *iipd*′: ‹*i*′∈ {*k*′..*j*′}› ‹*path* σ′ *i*′ = *ipd* (*path* σ′ *k*′)› **unfolding** *i*′*-def* **using** *LeastI-ex*[*OF ipdex*′] **by** *simp-all*

have $*$:‹$\forall\ i \in \{k'..<i'\}$. *path* σ' $i \neq$ *ipd* (*path* σ' k')› **proof** (*rule*, *rule ccontr*)
fix i **assume** $*$: ‹$i \in \{k'..<i'\}$› ‹$\neg$ *path* σ' $i \neq$ *ipd* (*path* σ' k')›
hence $**$: ‹$i \in\{k'..j'\} \wedge$ *path* σ' $i =$ *ipd* (*path* σ' k')› (**is** ‹?P i›) **using** *iipd'(1)* **by** *auto*
thus ‹*False*› **using** *Least-le*[*of* ‹?P› ‹i›] *i'-def* $*$ **by** *auto*
qed
have ‹$i' \neq k'$› **using** *iipd'(2)* **by** (*metis csk last-cs nret path-in-nodes ipd-not-self*)
hence ‹$k'<i'$› **using** *iipd'(1)* **by** *simp*
hence *csi'*: ‹$cs^{path\ \sigma'}$ $i' = [n\leftarrow cs^{path\ \sigma'}$ k' . *ipd* $n \neq$ *path* σ' i'] @ [*path* σ' i']›**using** *cs-ipd*[*OF iipd'(2)* $*$] **by** *fast*

have *ncdk'*: ‹$\neg$ n' $cd^{path\ \sigma'}\!\!\rightarrow k'$› **using** ‹$j' < n'$› ‹$k' < j'$› *cdi-prefix jcdk' less-imp-le-nat* **by** *blast*
hence *ncdk*: ‹$\neg$ n $cd^{path\ \sigma}\!\!\rightarrow k$› **using** *cd-cs-swap csk dcd(4)* **by** *blast*
have *ipdex*: ‹$\exists\, i.\ i\in\{k..n\} \wedge$ *path* σ $i =$ *ipd* (*path* σ k)› (**is** ‹$\exists\, i.$?P i›) **proof** *cases*
assume $*$:‹*path* σ $n =$ *return*›
from *path-ret-ipd*[*of* ‹*path* σ› ‹k› ‹n›,*OF path-is-path nret* $*$]
obtain i **where** ‹?P i› **by** *fastforce* **thus** ‹?thesis› **by** *auto*
next
assume $*$:‹*path* σ $n \neq$ *return*›
show ‹?thesis› **using** *not-cd-impl-ipd* [*of* ‹*path* σ› ‹k› ‹n›, *OF path-is-path* ‹$k<n$› *ncdk* $*$] **by** *auto*
qed

define i **where** ‹$i ==$ *LEAST* $j.\ j\in \{k..n\} \wedge$ *path* σ $j =$ *ipd* (*path* σ k)›
have *iipd*: ‹$i\in \{k..n\}$› ‹*path* σ $i =$ *ipd* (*path* σ k)› **unfolding** *i-def* **using** *LeastI-ex*[*OF ipdex*] **by** *simp-all*
have $**$:‹$\forall\ i' \in \{k..<i\}$. *path* σ $i' \neq$ *ipd* (*path* σ k)› **proof** (*rule*, *rule ccontr*)
fix i' **assume** $*$: ‹$i' \in \{k..<i\}$› ‹$\neg$ *path* σ $i' \neq$ *ipd* (*path* σ k)›
hence $**$: ‹$i' \in\{k..n\} \wedge$ *path* σ $i' =$ *ipd* (*path* σ k)› (**is** ‹?P i'›) **using** *iipd(1)* **by** *auto*
thus ‹*False*› **using** *Least-le*[*of* ‹?P› ‹i'›] *i-def* $*$ **by** *auto*
qed
have ‹$i \neq k$› **using** *iipd(2)* **by** (*metis nret path-in-nodes ipd-not-self*)
hence ‹$k<i$› **using** *iipd(1)* **by** *simp*
hence ‹$cs^{path\ \sigma}$ $i = [n\leftarrow cs^{path\ \sigma}$ k . *ipd* $n \neq$ *path* σ i] @ [*path* σ i]›**using** *cs-ipd*[*OF iipd(2)* $**$] **by** *fast*
hence *csi*: ‹$cs^{path\ \sigma}$ $i = cs^{path\ \sigma'}$ i'› **using** *csi' csk* **unfolding** *iipd'(2) iipd(2)* **by** (*metis last-cs*)
hence ‹(*LEAST* $i'.\ k' < i' \wedge (\exists\, i.\ cs^{path\ \sigma}$ $i = cs^{path\ \sigma'}$ i')) $\leq i'$› (**is** ‹(*LEAST* x. ?P x) $\leq$ -›)
using ‹$k' < i'$› *Least-le*[*of* ‹?P› ‹i'›] **by** *blast*
hence *nw*: ‹$\forall\, j'\in\{i'..<n'\}$. $v \notin$ *writes* (*path* σ' j')› **using** *dcd(7) allB-atLeastLessThan-lower* **by** *blast*
moreover have ‹$v \in$ *writes* (*path* σ' j')› **using** *nddj'* **unfolding** *is-ddi-def* **by** *auto*
moreover have ‹$i' \leq j'$› **using** *iipd'(1)* **by** *auto*
ultimately have ‹*False*› **using** ‹$j' < n'$› **by** *auto*
thus ‹?thesis› **..**
qed
qed
next
assume ‹$\neg$ ($\exists\, j'\in\{k'..<n'\}$. $v \in$ *writes* (*path* σ' j'))›

hence ‹n' $dcd^{path\ \sigma',v}\!\!\rightarrow$ k' *via* (*path* σ) k› **unfolding** *is-dcdi-via-def* **using** *dcd(2−4) csk* ‹$k'<n'$› *path-is-path* **by** *metis*
thus ‹?thesis› **using** *dcd.IH scp.intros(4)* **by** *blast*
qed
with *1* **show** ‹?case› **..**
qed

theorem *cop-in-scop*: **assumes** ‹$((\sigma,k),(\sigma',k'))\in cop$› **shows** ‹(*path* σ,k)$\in$*scop* $\wedge$ (*path* σ',k')$\in$*scp*›
using *assms*

apply (*induct rule*: *cop.induct*)
 apply (*simp add*: *cp-in-scp*)
using *cp-in-scp scop.intros scp.intros*(*2*)
 apply *blast*
using *cp-in-scp scop.intros scp.intros*(*2*)
apply *blast*
done

The main correctness result for out single execution approximation follows directly.

theorem *scop-correct*: **assumes** ‹*scop = empty*› **shows** ‹*secure*›
using *cop-correct assms cop-in-scop* **by** *fast*

end

end

3 Example: Program Dependence Graphs

Program dependence graph (PDG) based slicing provides a very crude but direct approximation of our characterisation. As such we can easily derive a corresponding correctness result.

theory *PDG* **imports** *IFC*
begin

context *IFC*
begin

We utilise our established dependencies on program paths to define the PDG. Note that PDGs usually only contain immediate control dependencies instead of the transitive ones we use here. However as slicing is considering reachability questions this does not affect the result.

inductive-set *pdg* **where**
‹$[\![i\ cd^{\pi}\!\rightarrow k]\!] \Longrightarrow (\pi\ k,\ \pi\ i) \in pdg$› |
‹$[\![i\ dd^{\pi,v}\!\rightarrow k]\!] \Longrightarrow (\pi\ k,\ \pi\ i) \in pdg$›

The set of sources is the set of nodes reading high variables.

inductive-set *sources* **where**
‹$n \in nodes \Longrightarrow h \in hvars \Longrightarrow h \in reads\ n \Longrightarrow n \in sources$›

The forward slice is the set of nodes reachable in the PDG from the set of sources. To ensure security slicing aims to prove that no observable node is contained in the

inductive-set *slice* **where**
‹$n \in sources \Longrightarrow n \in slice$› |
‹$m \in slice \Longrightarrow (m,n) \in pdg \Longrightarrow n \in slice$›

As the PDG does not contain data control dependencies themselves we have to decompose these.

lemma *dcd-pdg*: **assumes** ‹$n\ dcd^{\pi,v}\!\rightarrow m\ via\ \pi'\ m'$› **obtains** l **where** ‹$(\pi\ m,l) \in pdg$› **and** ‹$(l,\pi\ n) \in pdg$›
proof −
assume *r*: ‹$(\bigwedge l.\ (\pi\ m,\ l) \in pdg \Longrightarrow (l,\ \pi\ n) \in pdg \Longrightarrow thesis)$›
obtain l' n' **where** *ln*: ‹$cs^{\pi}\ m = cs^{\pi'}\ m' \wedge cs^{\pi}\ n = cs^{\pi'}\ n' \wedge n'\ dd^{\pi',v}\!\rightarrow l' \wedge l'\ cd^{\pi'}\!\rightarrow m'$› **using** *assms* **unfolding** *is-dcdi-via-def* **by** *metis*
hence *mn*: ‹$\pi'\ m' = \pi\ m \wedge \pi'\ n' = \pi\ n$› **by** (*metis last-cs ln*)
have *1*: ‹$(\pi\ m,\ \pi'\ l') \in pdg$› **by** (*metis ln mn pdg.intros*(*1*))
have *2*: ‹$(\pi'\ l',\ \pi\ n) \in pdg$› **by** (*metis ln mn pdg.intros*(*2*))

show *thesis* **using** *1 2 r* **by** *auto*
qed

By induction it directly follows that the slice is an approximation of the single critical paths.

lemma *scp-slice*: ‹$(\pi,\ i) \in scp \Longrightarrow \pi\ i \in slice$›
apply (*induction rule*: *scp.induct*)
apply (*simp add*: *path-in-nodes slice.intros*(*1*) *sources.intros*)
using *pdg.intros*(*1*) *slice.intros*(*2*) **apply** *blast*
using *pdg.intros*(*2*) *slice.intros*(*2*) **apply** *blast*
by (*metis dcd-pdg slice.intros*(*2*))

lemma *scop-slice*: ‹$(\pi,\ i) \in scop \Longrightarrow \pi\ i \in slice \cap dom(att)$› **by** (*metis IntI scop.cases scp-slice*)

The requirement targeted by slicing, that no observable node is contained in the slice, is thereby a sound criteria for security.

lemma *pdg-correct*: **assumes** ‹$slice \cap dom(att) = \{\}$› **shows** ‹*secure*›
proof (*rule ccontr*)
assume ‹$\neg$ *secure*›
then obtain π i **where** ‹$(\pi,\ i) \in scop$› **using** *scop-correct* **by** *force*
thus ‹*False*› **using** *scop-slice assms* **by** *auto*
qed

end

end

References

[1] A. Bohannon, B. C. Pierce, V. Sjöberg, S. Weirich, and S. Zdancewic. Reactive noninterference. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 79–90, New York, NY, USA, 2009. ACM.