```
accessgraphs[field_names: TYPE+]: THEORY
 BEGIN

  node: TYPE+ = [#name: field_names, block: nat, inst: nat#]

  edge: TYPE+ = [#source: node, target: node#]

  access_path: TYPE = finseq[field_names]

  set_nodes: TYPE+ = setof[node]

  ag: TYPE+ =
        [#inter: set_nodes,
          final: set_nodes,
          edges: setof[edge],
          n_0: node#]

  sequence_nodes: TYPE = finseq[node]

  set_ag: TYPE = setof[ag]

  empty_node: node

  e_g: ag =
      (#inter := emptyset,
        final := emptyset,
        edges := emptyset,
        n_0 := empty_node#)

  i, j: VAR nat

  n1, n2: VAR node

  e, e1, e2: VAR edge

  ρ, σ, φ: VAR access_path

  a1, a2, a3, ag_1: VAR ag

  seq_nodes, sq1, sq2: VAR sequence_nodes

  del: VAR setof[edge]

  cn: VAR set_nodes
```

```
sa, sb: VAR set_ag

same_root(a1, a2): bool = a1`n_0 = a2`n_0

path_described(seq_nodes): access_path =
    (LET len: nat = seq_nodes`length IN
        (#length := len,
          seq
            := (λ (i: below[len]): seq_nodes`seq(i)`name)#))

first_node(seq_nodes): node =
    (IF seq_nodes`length = 0
        THEN empty_node
      ELSE seq_nodes`seq(0)
      ENDIF)

belongs(e, seq_nodes): bool =
    (∃ i:
        i < (seq_nodes`length − 1) ∧
         seq_nodes`seq(i) = source(e) ∧
          seq_nodes`seq(i + 1) = target(e))

accepts_n(seq_nodes, a1): bool =
    (IF seq_nodes`length = 0
        THEN TRUE
      ELSIF seq_nodes`length = 1 THEN a1`n_0 = seq_nodes`seq(0)
      ELSE a1`n_0 = seq_nodes`seq(0) ∧
            (∀ e: belongs(e, seq_nodes) ⇒ (e ∈ a1`edges))
      ENDIF)

accepts_nodes(seq_nodes, a1): bool =
    (∀ e: belongs(e, seq_nodes) ⇒ (e ∈ a1`edges))

last_node(seq_nodes): node =
    (IF seq_nodes`length = 0
        THEN empty_node
      ELSE seq_nodes`seq(seq_nodes`length − 1)
      ENDIF)

last_final(a1, seq_nodes): bool =
    (last_node(seq_nodes) ∈ a1`final)

accepts(ρ, a1): bool =
```

```
    (IF ρ‘length = 0 ∧ a_1 = e_g
        THEN TRUE
      ELSE ∃ seq_nodes:
               path_described(seq_nodes) = ρ ∧
                first_node(seq_nodes) = a_1‘n_0 ∧
                 accepts_n(seq_nodes, a_1) ∧ last_final(a_1, seq_nodes)
      ENDIF)

path_exists(a_1, n_1, n_2): bool =
    (IF n_1 = n_2
        THEN TRUE
      ELSE ∃ seq_nodes:
               first_node(seq_nodes) = n_1 ∧
                last_node(seq_nodes) = n_2 ∧ accepts_nodes(seq_nodes, a_1)
      ENDIF)

reachable_entry(a_1, n_1): bool = path_exists(a_1, a_1‘n_0, n_1)

path_to_final(a_1, n_1): bool =
    ∃ n_2: (n_2 ∈ a_1‘final) ∧ path_exists(a_1, n_1, n_2)

clean_nodes(a_1): ag =
    a_1
      WITH [(inter)
                 := (a_1‘inter \ {n_1 : node | ¬ (reachable_entry(a_1, n_1) ∧ path_to_final(a_1, n_1))}),
               (final)
                 := (a_1‘final \ {n_1 : node | ¬ (reachable_entry(a_1, n_1))})]

clean_edges(a_1): ag =
    a_1
      WITH [(edges)
                 := (a_1‘edges \ {e : edge | ¬ ((source(e) ∈ (a_1‘inter ∪ a_1‘final)) ∧ (target(e) ∈ (a_1‘inter ∪ a_1‘final)))})]

cleanup(a_1): ag = clean_edges(clean_nodes(a_1))

cnodes(a_1, a_2): set_nodes =
    (IF same_root(a_1, a_2)
        THEN ({n_1 : node | (n_1 ∈ (a_1‘inter ∪ a_1‘final)) ∧ (∃ ρ, seq_nodes : accepts(ρ, a_2) ∧ path_described(seq_nodes) = ρ ∧ first_node(seq_nodes) = a_1‘n_0 ∧ accepts_n(seq_nodes, a_1) ∧ last_node(seq_nodes) = n_1)} \ singleton(empty_node))
      ELSE emptyset
      ENDIF)

cnodes(a_1, ρ): set_nodes =
    (IF ρ‘length = 0
        THEN emptyset
```

```
    ELSE {n1: node |
             (n1 ∈ (a1`inter ∪ a1`final)) ∧
              (∃ seq_nodes:
                  path_described(seq_nodes) = ρ ∧
                   first_node(seq_nodes) = a1`n_0 ∧
                    accepts_n(seq_nodes, a1) ∧ last_node(seq_nodes) = n1)}
    ENDIF)

ag_union(a1, a2): ag =
    (IF a1`n_0 = a2`n_0 ∧ ¬ a1 = e_g ∧ ¬ a2 = e_g
      THEN (#inter := ((a1`inter ∪ a2`inter) \ (a1`final ∪ a2`final)),
             final := (a1`final ∪ a2`final),
             edges := (a1`edges ∪ a2`edges),
             n_0 := a1`n_0#)
    ELSE e_g
    ENDIF)

unique_path(a1, n1, n2): bool =
    ∃ ρ:
     (∀ seq_nodes:
         (first_node(seq_nodes) = n1 ∧
            last_node(seq_nodes) = n2 ∧ accepts_n(seq_nodes, a1))
          ⇒ path_described(seq_nodes) = ρ)

compute_delete(a1, ρ): setof[edge] =
    (IF ρ`length > 1
      THEN {e: edge |
               (e ∈ a1`edges) ∧
                target(e)`name = ρ`seq(ρ`length − 1) ∧
                 unique_path(a1, a1`n_0, source(e)) ∧
                  (source(e) ∈ cnodes(a1, ^(ρ, (0, ρ`length − 2))))}
    ELSE emptyset
    ENDIF)

compute_belong(sq1): setof[edge] = {e: edge | belongs(e, sq1)}

pathremove(a1, ρ): ag =
    (IF ρ`length = 0
      THEN a1
    ELSIF ¬ a1`n_0`name = ρ`seq(0) THEN a1
    ELSIF ρ`length = 1 THEN e_g
    ELSE a1 WITH [edges := (a1`edges \ compute_delete(a1, ρ))]
    ENDIF)
```

```
prefix(sq1, i): sequence_nodes =
    (LET len: nat = (IF i < sq1`length THEN i ELSE 0 ENDIF) IN
        (IF sq1`length = 0
            THEN empty_seq
          ELSE (#length := len,
                  seq := (λ (k: below[len]): finseq_appl[node](sq1)(k))#)
          ENDIF))

accepts(σ, sa): bool = ∃ a_1: (a_1 ∈ sa) ∧ accepts(σ, a_1)

compute_entry(a_1, cn): set_nodes =
    {n_1: node |
        (n_1 ∈ (a_1`inter ∪ a_1`final)) ∧
         (∃ e:
             (e ∈ a_1`edges) ∧
              (source(e) ∈ cn) ∧ target(e) = n_1)}

compute_subgraph(a_1, n_1): ag =
    (LET ir: set_nodes = {n_2: node | (n_2 ∈ a_1`inter) ∧ path_exists(a_1, n_1, n_2)},
          fl: set_nodes = {n_2: node | (n_2 ∈ a_1`final) ∧ path_exists(a_1, n_1, n_2)}
        IN
        (#n_0 := n_1,
          inter := ir,
          final := fl,
          edges
            := {e: edge |
                   (e ∈ a_1`edges) ∧
                    (source(e) ∈ (ir ∪ fl)) ∧
                     (target(e) ∈ (ir ∪ fl))}#))

factorise(a_1, a_2): set_ag =
    (LET cn: set_nodes = cnodes(a_1, a_2) IN
        (IF empty?(cn)
            THEN emptyset
          ELSE (LET entry: set_nodes = compute_entry(a_1, cn) IN
                   {a_3: ag | ∃ n_1: a_3 = compute_subgraph(a_1, n_1) ∧ (n_1 ∈ entry)})
          ENDIF))

rev_factorise(a_1, a_2): set_ag =
    (LET sa: set_ag = factorise(a_1, a_2), cn: set_nodes = cnodes(a_1, a_2) IN
        (IF empty?((cn ∩ a_1`final))
            THEN sa
          ELSE (singleton(e_g) ∪ sa)
          ENDIF))
```

```
extend(a_1, a_2): ag =
  (IF a_2 = e_g ∨ a_1 = e_g
     THEN a_1
   ELSE a_1
          WITH [final := a_2`final,
                inter := (((a_1`inter ∪ a_1`final) ∪ a_2`inter) \ a_2`final),
                edges
                  := ((a_1`edges ∪ a_2`edges) ∪ {e: edge | (source(e) ∈ a_1`final) ∧ target(e) = a_2`n_0})]
   ENDIF)

meaningful_graph(a_1): bool = (a_1`inter ∩ a_1`final) = emptyset

meaningful_nodes(a_1): bool =
  (IF ¬ a_1 = e_g
     THEN (∀ n_1: (n_1 ∈ (a_1`inter ∪ a_1`final)) ⇒ ¬ n_1 = empty_node) ∧
           ¬ a_1`n_0 = empty_node
   ELSE TRUE
   ENDIF)

meaningful_edges(a_1): bool =
  (∀ e:
     (e ∈ a_1`edges) ⇒
      (target(e) ∈ (a_1`inter ∪ a_1`final)) ∧
       (source(e) ∈ (a_1`inter ∪ a_1`final)))

meaningful_start(a_1): bool =
  (IF ¬ a_1`n_0 = empty_node
     THEN (a_1`n_0 ∈ (a_1`inter ∪ a_1`final))
   ELSE TRUE
   ENDIF)

at_least_one_final(a_1): bool =
  IF ¬ a_1 = e_g THEN ¬ empty?(a_1`final) ELSE TRUE ENDIF

no_in_edges_start(a_1): bool =
  (IF ∃ e: (e ∈ a_1`edges) ∧ target(e) = a_1`n_0
     THEN FALSE
   ELSE TRUE
   ENDIF)

no_dangling_nodes(a_1): bool =
  (∀ n_1:
     (n_1 ∈ (a_1`inter ∪ a_1`final)) ⇒
```

reachable_entry(a_1, n_1) $\wedge$ path_to_final(a_1, n_1))

commutative_union: LEMMA ag_union(a_1, a_2) $=$ ag_union(a_2, a_1)

union_lemma1: LEMMA
same_root(a_1, a_2) $\wedge\ \neg\ a_1 =$ e_g $\wedge\ \neg\ a_2 =$ e_g $\wedge$ accepts(ρ, a_1) $\Rightarrow$
accepts(ρ, ag_union(a_1, a_2))

correctness_union: THEOREM
same_root(a_1, a_2) $\wedge$
$\neg\ a_1 =$ e_g $\wedge\ \neg\ a_2 =$ e_g $\wedge$ (accepts(ρ, a_1) $\vee$ accepts(ρ, a_2))
$\Rightarrow$ accepts(ρ, ag_union(a_1, a_2))

pathremoval1: THEOREM
accepts_n(sq1, a_1) $\wedge$ (compute_belong(sq1) $\cap$ del) $=$ emptyset $\Rightarrow$
accepts_n(sq1, a_1 WITH [edges := (a_1‘edges $\setminus$ del)])

pathremoval2_1: THEOREM
ϕ‘length $\geq 1\ \wedge\ \rho$‘length $\geq 1\ \wedge\ \hat{\ }_{\wedge}(\phi,\ (0,\ \rho$‘length $- 1)) = \rho \Rightarrow$
$\phi = \rho \circ \hat{\ }_{\wedge}(\phi,\ (\rho$‘length, ϕ‘length $- 1))$

pathremoval2_2: THEOREM
path_described(sq1) $= \phi\ \wedge\ i < \phi$‘length $\Rightarrow$
path_described($\hat{\ }_{\wedge}$(sq1, $(0,\ i)$)) $= \hat{\ }_{\wedge}(\phi,\ (0,\ i))$

pathremoval2_3: THEOREM
accepts_n(sq1, a_1) $\wedge\ i <$ sq1‘length $\Rightarrow$
accepts_n($\hat{\ }_{\wedge}$(sq1, $(0,\ i)$), a_1)

pathremoval2_4: THEOREM
first_node(sq1) $= n_1\ \wedge\ i <$ sq1‘length $\Rightarrow$
first_node($\hat{\ }_{\wedge}$(sq1, $(0,\ i)$)) $= n_1$

pathremoval2_5: THEOREM
$i <$ sq1‘length $\Rightarrow\ \neg\ \hat{\ }_{\wedge}$(sq1, $(0,\ i)$)‘length $= 0$

pathremoval2_6: THEOREM
$i <$ sq1‘length $\Rightarrow$
last_node($\hat{\ }_{\wedge}$(sq1, $(0,\ i)$)) $=$ sq1‘seq(i)

pathremoval2_7: THEOREM
ρ‘length $> 1\ \wedge$
$i <$ sq1‘length $- 1\ \wedge$
path_described($\hat{\ }_{\wedge}$(sq1, $(0,\ i)$)) $= \hat{\ }_{\wedge}(\rho,\ (0,\ \rho$‘length $- 2))\ \wedge$

sq1 ‘seq$(1+i)$ ‘name = ρ ‘seq(ρ‘length − 1)
⇒ path_described($\hat{}_{\wedge}$(sq1, (0, $1+i$))) = ρ

pathremoval2: THEOREM
path_described(sq1) = ϕ ∧
first_node(sq1) = a_1 ‘n_0 ∧
accepts_n(sq1, a_1) ∧ ρ ‘length < ϕ ‘length ∧ ¬ (∃ σ: ϕ = $\rho \circ \sigma$)
⇒
((compute_belong(sq1) ∩ compute_delete(a_1, ρ)) =
emptyset)

pathremoval: THEOREM
accepts(ϕ, a_1) ∧
¬ a_1 = e_g ∧
ρ ‘length > 1 ∧ ρ ‘length < ϕ ‘length ∧ ¬ (∃ σ: ϕ = $\rho \circ \sigma$)
⇒ accepts(ϕ, pathremove(a_1, ρ))

factorisation_lemma5: LEMMA
sq1 ‘length ≥ 2 ∧ i > 0 ∧ i < sq1 ‘length ⇒
belongs((#source := sq1 ‘seq($i-1$), target := sq1 ‘seq(i)#),
sq1)

factorisation_axiom: AXIOM
source(e) = n_1 ∧ target(e) = n_2 ⇒
e = (#source := n_1, target := n_2#)

factorisation_axiom2: AXIOM
source(e_1) = source(e_2) ∧ target(e_1) = target(e_2) ⇒
e_1 = e_2

factorisation_axiom3: AXIOM
($e \in a_1$‘edges) ⇒
(source(e) ∈ (a_1‘inter ∪ a_1‘final)) ∧
(target(e) ∈ (a_1‘inter ∪ a_1‘final))

factorisation_lemma1_2: LEMMA
i ≥ 0 ∧
i < sq1‘length − 1 ∧
j > 0 ∧ j < sq1 ‘length ∧ j > i ∧ accepts_n(sq1, a_1)
⇒ path_exists(a_1, sq1 ‘seq(i), sq1 ‘seq(j))

factorisation_lemma1_1_1: LEMMA
¬ σ ‘length = 0 ∧
path_described(sq1) = ($\rho \circ \sigma$) ∧

belongs(e, ^(sq1, (ρ‘length, sq1‘length $-$ 1)))
$\Rightarrow$ belongs(e, sq1)

factorisation_lemma1_1: LEMMA
path_described(sq1) = ($\rho \circ \sigma$) $\wedge$
accepts_nodes(sq1, a_1) $\wedge$ last_final(a_1, sq1) $\wedge$ $\neg$ σ‘length = 0
$\Rightarrow$
path_described(^(sq1, (ρ‘length, sq1‘length $-$ 1))) = σ $\wedge$
accepts_nodes(^(sq1, (ρ‘length, sq1‘length $-$ 1)), a_1) $\wedge$
last_final(a_1, ^(sq1, (ρ‘length, sq1‘length $-$ 1)))

factorisation_lemma1_3: LEMMA
accepts_n(sq1, a_1) $\Rightarrow$ accepts_nodes(sq1, a_1)

factorisation_lemma1: LEMMA
accepts(ρ, a_2) $\wedge$
path_described(sq1) = ($\rho \circ \sigma$) $\wedge$
first_node(sq1) = a_1‘n_0 $\wedge$
accepts_n(sq1, a_1) $\wedge$ last_final(a_1, sq1) $\wedge$ $\neg$ σ‘length = 0
$\Rightarrow$ accepts(σ, compute_subgraph(a_1, sq1‘seq(ρ‘length)))

factorisation_lemma2_1: LEMMA
path_described(sq1) = $\rho \circ \sigma$ $\wedge$ $\neg$ σ‘length = 0 $\Rightarrow$
path_described(prefix(sq1, ρ‘length)) = ρ

factorisation_lemma2_2: LEMMA
meaningful_nodes(a_1) $\wedge$
accepts(ρ, a_2) $\wedge$
path_described(sq1) = ($\rho \circ \sigma$) $\wedge$
first_node(sq1) = a_1‘n_0 $\wedge$
accepts_n(sq1, a_1) $\wedge$ same_root(a_1, a_2) $\wedge$ $\neg$ σ‘length = 0 $\wedge$ $\neg$ ρ‘length = 0
$\Rightarrow$ (sq1‘seq(ρ‘length $-$ 1) $\in$ cnodes(a_1, a_2))

factorisation_lemma2: LEMMA
meaningful_nodes(a_1) $\wedge$
accepts(ρ, a_2) $\wedge$
path_described(sq1) = ($\rho \circ \sigma$) $\wedge$
first_node(sq1) = a_1‘n_0 $\wedge$
accepts_n(sq1, a_1) $\wedge$ same_root(a_1, a_2) $\wedge$ $\neg$ σ‘length = 0 $\wedge$ $\neg$ ρ‘length = 0
$\Rightarrow$ $\neg$ empty?(cnodes(a_1, a_2))

factorisation_lemma3_1: LEMMA
meaningful_nodes(a_1) $\wedge$
accepts(ρ, a_2) $\wedge$

```
      path_described(sq1) = (ρ ∘ σ) ∧
        first_node(sq1) = a1‘n_0 ∧
          accepts_n(sq1, a1) ∧ same_root(a1, a2) ∧ ¬ σ‘length = 0 ∧ ¬ ρ‘length = 0
    ⇒
    (sq1‘seq(ρ‘length) ∈ compute_entry(a1, cnodes(a1, a2)))

factorisation_lemma3: LEMMA
  meaningful_nodes(a1) ∧
    accepts(ρ, a2) ∧
      path_described(sq1) = (ρ ∘ σ) ∧
        first_node(sq1) = a1‘n_0 ∧
          accepts_n(sq1, a1) ∧ same_root(a1, a2) ∧ ¬ σ‘length = 0 ∧ ¬ ρ‘length = 0
    ⇒ ¬ empty?(compute_entry(a1, cnodes(a1, a2)))

factorisation_lemma6: LEMMA
  path_described(sq1) = (ρ ∘ σ) ∧
    first_node(sq1) = a1‘n_0 ∧
      accepts_n(sq1, a1) ∧ last_final(a1, sq1) ∧ ¬ σ‘length = 0 ∧ ¬ ρ‘length = 0
    ⇒
    first_node(prefix(sq1, ρ‘length)) = a1‘n_0 ∧
      accepts_n(prefix(sq1, ρ‘length), a1)

correctness_factorisation: THEOREM
  same_root(a1, a2) ∧
    meaningful_nodes(a1) ∧
      meaningful_nodes(a2) ∧
        accepts(ρ, a2) ∧
          accepts(ρ ∘ σ, a1) ∧ ¬ σ‘length = 0 ∧ ¬ ρ‘length = 0
    ⇒ accepts(σ, factorise(a1, a2))

conv_factorisation_lemma1: LEMMA
  path_described(sq1) = ρ ∧ path_described(sq2) = σ ⇒
    path_described(sq1 ∘ sq2) = ρ ∘ σ

conv_factorisation_lemma2: LEMMA
  first_node(sq1) = n1 ∧ ¬ sq1‘length = 0 ⇒
    first_node(sq1 ∘ sq2) = n1

conv_factorisation_lemma3: LEMMA
  belongs(e, sq1 ∘ sq2) ⇒
    belongs(e, sq1) ∨
      belongs(e, sq2) ∨
        (source(e) = last_node(sq1) ∧
            target(e) = first_node(sq2))
```

conv_factorisation_lemma4: LEMMA
accepts_n(sq1, a_1) $\wedge$ $i \geq 0$ $\wedge$ $i <$ sq1‘length $\wedge$ sq1‘length $\geq 2 \Rightarrow$
(sq1‘seq(i) $\in$ (a_1‘inter $\cup$ a_1‘final))

conv_factorisation_lemma5: LEMMA
($e \in$ compute_subgraph(a_1, n_1)‘edges) $\Rightarrow$ ($e \in a_1$‘edges)

conv_factorisation_lemma6: LEMMA
($n_2 \in$ (a_2‘inter $\cup$ a_2‘final)) $\wedge$ a_2 = compute_subgraph(a_1, n_1) $\Rightarrow$
($n_2 \in$ (a_1‘inter $\cup$ a_1‘final))

conv_factorisation_lemma7: LEMMA
meaningful_start(a_1) $\wedge$ ($n_1 \in$ (a_1‘inter $\cup$ a_1‘final)) $\Rightarrow$
meaningful_start(compute_subgraph(a_1, n_1))

conv_factorisation_lemma8: LEMMA accepts(σ, e_g) $\Rightarrow$ σ‘length = 0

conv_factorisation_lemma9: LEMMA
a_2 = compute_subgraph(a_1, n_1) $\wedge$
accepts_n(sq1, a_1) $\wedge$
accepts_n(sq2, a_2) $\wedge$
($e \in a_1$‘edges) $\wedge$
source(e) = last_node(sq1) $\wedge$
target(e) = n_1 $\wedge$
$\neg$ sq2‘length = 0 $\wedge$
meaningful_edges(a_1) $\wedge$ meaningful_nodes(a_1) $\wedge$ first_node(sq2) = a_2‘n_0
$\Rightarrow$ accepts_n(sq1 $\circ$ sq2, a_1)

converse_factorisation: THEOREM
no_dangling_nodes(a_1) $\wedge$
same_root(a_1, a_2) $\wedge$
meaningful_nodes(a_1) $\wedge$
meaningful_nodes(a_2) $\wedge$
meaningful_start(a_1) $\wedge$
meaningful_start(a_2) $\wedge$
$\neg$ σ‘length = 0 $\wedge$ meaningful_edges(a_1) $\wedge$ accepts(σ, factorise(a_1, a_2))
$\Rightarrow$ ($\exists$ ρ: accepts(ρ, a_2) $\wedge$ accepts($\rho \circ \sigma$, a_1))

rev_correctness_factorisation: THEOREM
same_root(a_1, a_2) $\wedge$
meaningful_nodes(a_1) $\wedge$
meaningful_nodes(a_2) $\wedge$
accepts(ρ, a_2) $\wedge$

$\quad\quad$ accepts($\rho \circ \sigma$, a_1) $\wedge$ $\neg$ σ‘length = 0 $\wedge$ $\neg$ ρ‘length = 0
$\quad \Rightarrow$ accepts(σ, rev_factorise(a_1, a_2))

rev_converse_factorisation: THEOREM
no_dangling_nodes(a_1) $\wedge$
same_root(a_1, a_2) $\wedge$
meaningful_nodes(a_1) $\wedge$
meaningful_nodes(a_2) $\wedge$
meaningful_start(a_1) $\wedge$
meaningful_start(a_2) $\wedge$
$\neg$ σ‘length = 0 $\wedge$ meaningful_edges(a_1) $\wedge$ accepts(σ, rev_factorise(a_1, a_2))
$\Rightarrow$ ($\exists$ ρ: accepts(ρ, a_2) $\wedge$ accepts($\rho \circ \sigma$, a_1))

correctness_extension_lemma1: THEOREM
accepts(σ, a_2) $\wedge$ a_2 = e_g $\Rightarrow$ σ‘length = 0

correctness_extension_lemma2: THEOREM
accepts(ρ, a_1) $\wedge$ σ‘length = 0 $\Rightarrow$
accepts($\rho \circ \sigma$, a_1)

correctness_extension_lemma3: THEOREM
accepts(σ, a_2) $\wedge$ $\neg$ a_2 = e_g $\wedge$ meaningful_nodes(a_2) $\Rightarrow$
$\neg$ σ‘length = 0

correctness_extension: THEOREM
accepts(ρ, a_1) $\wedge$
accepts(σ, a_2) $\wedge$ $\neg$ a_1 = e_g $\wedge$ meaningful_nodes(a_2) $\wedge$ meaningful_nodes(a_1)
$\Rightarrow$ accepts($\rho \circ \sigma$, extend(a_1, a_2))

END accessgraphs