

Size-Change Termination

Alexander Krauss

June 8, 2008

1 Miscellaneous Tools for Size-Change Termination

```
theory Misc-Tools
imports Main
begin
```

1.1 Searching in lists

```
fun index-of :: 'a list  $\Rightarrow$  'a  $\Rightarrow$  nat
where
  index-of [] c = 0
| index-of (x#xs) c = (if x = c then 0 else Suc (index-of xs c))
```

```
lemma index-of-member:
  (x  $\in$  set l)  $\implies$  (l ! index-of l x = x)
by (induct l) auto
```

```
lemma index-of-length:
  (x  $\in$  set l) = (index-of l x < length l)
by (induct l) auto
```

1.2 Some reasoning tools

```
lemma three-cases:
  assumes a1  $\implies$  thesis
  assumes a2  $\implies$  thesis
  assumes a3  $\implies$  thesis
  assumes  $\bigwedge R. \llbracket a1 \implies R; a2 \implies R; a3 \implies R \rrbracket \implies R$ 
  shows thesis
  using assms
  by auto
```

1.3 Sequences

```
types
  'a sequence = nat  $\Rightarrow$  'a
```

1.3.1 Increasing sequences

definition

$increasing :: (nat \Rightarrow nat) \Rightarrow bool$ **where**
 $increasing\ s = (\forall i\ j. i < j \longrightarrow s\ i < s\ j)$

lemma *increasing-strict*:

assumes *increasing s*
assumes $i < j$
shows $s\ i < s\ j$
using *assms*
unfolding *increasing-def* **by** *simp*

lemma *increasing-weak*:

assumes *increasing s*
assumes $i \leq j$
shows $s\ i \leq s\ j$
using *assms increasing-strict* [*of s i j*]
by (*cases i < j*) *auto*

lemma *increasing-inc*:

assumes *increasing s*
shows $n \leq s\ n$
proof (*induct n*)
 case 0 **then show** ?*case* **by** *simp*
next
 case (*Suc n*)
 with *increasing-strict* [*OF increasing s*, *of n Suc n*]
 show ?*case* **by** *auto*
qed

lemma *increasing-bij*:

assumes [*simp*]: *increasing s*
shows $(s\ i < s\ j) = (i < j)$
proof
 assume $s\ i < s\ j$
 show $i < j$
 proof (*rule classical*)
 assume $\neg ?thesis$
 hence $j \leq i$ **by** *arith*
 with *increasing-weak* **have** $s\ j \leq s\ i$ **by** *simp*
 with $(s\ i < s\ j)$ **show** ?*thesis* **by** *simp*
 qed
qed (*simp add: increasing-strict*)

1.3.2 Sections induced by an increasing sequence

abbreviation

$section\ s\ i == \{s\ i ..< s\ (Suc\ i)\}$

definition

$\text{section-of } s \ n = (\text{LEAST } i. \ n < s \ (\text{Suc } i))$

lemma *section-help*:

assumes *increasing s*

shows $\exists i. \ n < s \ (\text{Suc } i)$

proof –

have $n \leq s \ n$

using $\langle \text{increasing } s \rangle$ **by** (rule *increasing-inc*)

also have $\dots < s \ (\text{Suc } n)$

using $\langle \text{increasing } s \rangle$ *increasing-strict* **by** *simp*

finally show *?thesis* ..

qed

lemma *section-of2*:

assumes *increasing s*

shows $n < s \ (\text{Suc } (\text{section-of } s \ n))$

unfolding *section-of-def*

by (rule *LeastI-ex*) (rule *section-help* [*OF* $\langle \text{increasing } s \rangle$])

lemma *section-of1*:

assumes [*simp*, *intro*]: *increasing s*

assumes $s \ i \leq n$

shows $s \ (\text{section-of } s \ n) \leq n$

proof (rule *classical*)

let $?m = \text{section-of } s \ n$

assume $\neg ?thesis$

hence $a: n < s \ ?m$ **by** *simp*

have *nonzero*: $?m \neq 0$

proof

assume $?m = 0$

from *increasing-weak* **have** $s \ 0 \leq s \ i$ **by** *simp*

also note $\langle \dots \leq n \rangle$

finally show *False* **using** $\langle ?m = 0 \rangle \langle n < s \ ?m \rangle$ **by** *simp*

qed

with *a* **have** $n < s \ (\text{Suc } (?m - 1))$ **by** *simp*

with *Least-le* **have** $?m \leq ?m - 1$

unfolding *section-of-def* .

with *nonzero* **show** *?thesis* **by** *simp*

qed

lemma *section-of-known*:

assumes [*simp*]: *increasing s*

assumes *in-sect*: $k \in \text{section } s \ i$

shows $\text{section-of } s \ k = i$ (**is** $?s = i$)

proof (rule *classical*)

assume $\neg ?thesis$

```

hence ?s < i ∨ ?s > i by arith
thus ?thesis
proof
  assume ?s < i
  hence Suc ?s ≤ i by simp
  with increasing-weak have s (Suc ?s) ≤ s i by simp
  moreover have k < s (Suc ?s) using section-of2 by simp
  moreover from in-sect have s i ≤ k by simp
  ultimately show ?thesis by simp
next
  assume i < ?s hence Suc i ≤ ?s by simp
  with increasing-weak have s (Suc i) ≤ s ?s by simp
  moreover
  from in-sect have s i ≤ k by simp
  with section-of1 have s ?s ≤ k by simp
  moreover from in-sect have k < s (Suc i) by simp
  ultimately show ?thesis by simp
qed
qed

lemma in-section-of:
  assumes increasing s
  assumes s i ≤ k
  shows k ∈ section s (section-of s k)
  using assms
  by (auto intro:section-of1 section-of2)

end

```

2 Kleene Algebras

```

theory Kleene-Algebras
imports Main
begin

```

A type class of kleene algebras

```

class star = type +
  fixes star :: 'a ⇒ 'a

```

```

class idem-add = ab-semigroup-add +
  assumes add-idem [simp]: x + x = x

```

```

lemma add-idem2[simp]: (x::'a::idem-add) + (x + y) = x + y
  unfolding add-assoc[symmetric]
  by simp

```

```

class order-by-add = idem-add + ord +

```

```

assumes order-def:  $a \leq b \iff a + b = b$ 
assumes strict-order-def:  $a < b \iff a \leq b \wedge a \neq b$ 

lemma ord-simp1[simp]:  $(x::'a::\text{order-by-add}) \leq y \implies x + y = y$ 
unfolding order-def .
lemma ord-simp2[simp]:  $(x::'a::\text{order-by-add}) \leq y \implies y + x = y$ 
unfolding order-def add-commute .
lemma ord-intro:  $(x::'a::\text{order-by-add}) + y = y \implies x \leq y$ 
unfolding order-def .

instance order-by-add  $\subseteq$  order
proof
  fix  $x\ y\ z :: 'a$ 
  show  $x \leq x$  unfolding order-def by simp

  show  $\llbracket x \leq y; y \leq z \rrbracket \implies x \leq z$ 
  proof (rule ord-intro)
    assume  $x \leq y\ y \leq z$ 

    have  $x + z = x + y + z$  by (simp add: $\langle y \leq z \rangle$  add-assoc)
    also have  $\dots = y + z$  by (simp add: $\langle x \leq y \rangle$ )
    also have  $\dots = z$  by (simp add: $\langle y \leq z \rangle$ )
    finally show  $x + z = z$  .
  qed

  show  $\llbracket x \leq y; y \leq x \rrbracket \implies x = y$  unfolding order-def
    by (simp add:add-commute)
  show  $x < y \iff x \leq y \wedge x \neq y$  by (fact strict-order-def)
qed

class pre-kleene = semiring-1 + order-by-add

instance pre-kleene  $\subseteq$  pordered-semiring
proof
  fix  $x\ y\ z :: 'a$ 

  assume  $x \leq y$ 

  show  $z + x \leq z + y$ 
  proof (rule ord-intro)
    have  $z + x + (z + y) = x + y + z$  by (simp add:add-ac)
    also have  $\dots = z + y$  by (simp add: $\langle x \leq y \rangle$  add-ac)
    finally show  $z + x + (z + y) = z + y$  .
  qed

  show  $z * x \leq z * y$ 
  proof (rule ord-intro)
    from  $\langle x \leq y \rangle$  have  $z * (x + y) = z * y$  by simp

```

```

    thus  $z * x + z * y = z * y$  by (simp add:right-distrib)
  qed

  show  $x * z \leq y * z$ 
  proof (rule ord-intro)
    from  $\langle x \leq y \rangle$  have  $(x + y) * z = y * z$  by simp
    thus  $x * z + y * z = y * z$  by (simp add:left-distrib)
  qed
qed

class kleene = pre-kleene + star +
  assumes star1:  $1 + a * star\ a \leq star\ a$ 
  and star2:  $1 + star\ a * a \leq star\ a$ 
  and star3:  $a * x \leq x \implies star\ a * x \leq x$ 
  and star4:  $x * a \leq x \implies x * star\ a \leq x$ 

class kleene-by-complete-lattice = pre-kleene
  + complete-lattice + recpower + star +
  assumes star-cont:  $a * star\ b * c = SUPR\ UNIV\ (\lambda n. a * b ^ n * c)$ 

lemma plus-leI:
  fixes  $x :: 'a :: order-by-add$ 
  shows  $x \leq z \implies y \leq z \implies x + y \leq z$ 
  unfolding order-def by (simp add:add-assoc)

lemma le-SUPI':
  fixes  $l :: 'a :: complete-lattice$ 
  assumes  $l \leq M\ i$ 
  shows  $l \leq (SUP\ i. M\ i)$ 
  using assms by (rule order-trans) (rule le-SUPI [OF UNIV-I])

lemma zero-minimum[simp]:  $(0 :: 'a :: pre-kleene) \leq x$ 
  unfolding order-def by simp

instance kleene-by-complete-lattice  $\subseteq$  kleene
proof

  fix  $a\ x :: 'a$ 

  have [simp]:  $1 \leq star\ a$ 
    unfolding star-cont[of 1 a 1, simplified]
    by (subst power-0[symmetric]) (rule le-SUPI [OF UNIV-I])

  show  $1 + a * star\ a \leq star\ a$ 
    apply (rule plus-leI, simp)
    apply (simp add:star-cont[of a a 1, simplified])
    apply (simp add:star-cont[of 1 a 1, simplified])
    apply (subst power-Suc[symmetric])
    by (intro SUP-leI le-SUPI UNIV-I)

```

```

show  $1 + \text{star } a * a \leq \text{star } a$ 
  apply (rule plus-leI, simp)
  apply (simp add:star-cont[of 1 a a, simplified])
  apply (simp add:star-cont[of 1 a 1, simplified])
  by (auto intro: SUP-leI le-SUPI UNIV-I simp add: power-Suc[symmetric]
power-commutes)

```

```

show  $a * x \leq x \implies \text{star } a * x \leq x$ 

```

```

proof -
  assume a:  $a * x \leq x$ 

  {
    fix n
    have  $a ^ (Suc n) * x \leq a ^ n * x$ 
    proof (induct n)
      case 0 thus ?case by (simp add:a power-Suc)
    next
      case (Suc n)
      hence  $a * (a ^ Suc n * x) \leq a * (a ^ n * x)$ 
        by (auto intro: mult-mono)
      thus ?case
        by (simp add:power-Suc mult-assoc)
    qed
  }
  note a = this

```

```

{
  fix n have  $a ^ n * x \leq x$ 
  proof (induct n)
    case 0 show ?case by simp
  next
    case (Suc n) with a[of n]
    show ?case by simp
  qed
}
note b = this

```

```

show  $\text{star } a * x \leq x$ 
  unfolding star-cont[of 1 a x, simplified]
  by (rule SUP-leI) (rule b)
qed

```

```

show  $x * a \leq x \implies x * \text{star } a \leq x$ 

```

```

proof -
  assume a:  $x * a \leq x$ 

```

```

{
  fix n

```

```

    have  $x * a ^ (Suc\ n) \leq x * a ^ n$ 
  proof (induct n)
    case 0 thus ?case by (simp add: a power-Suc)
  next
    case (Suc n)
    hence  $(x * a ^ Suc\ n) * a \leq (x * a ^ n) * a$ 
      by (auto intro: mult-mono)
    thus ?case
      by (simp add: power-Suc power-commutes mult-assoc)
  qed
}
note a = this

{
  fix n have  $x * a ^ n \leq x$ 
  proof (induct n)
    case 0 show ?case by simp
  next
    case (Suc n) with a[of n]
    show ?case by simp
  qed
}
note b = this

show  $x * star\ a \leq x$ 
  unfolding star-cont[of x a 1, simplified]
  by (rule SUP-leI) (rule b)
qed
qed

lemma less-add[simp]:
  fixes a b :: 'a :: order-by-add
  shows  $a \leq a + b$ 
  and  $b \leq a + b$ 
  unfolding order-def
  by (auto simp: add-ac)

lemma add-est1:
  fixes a b c :: 'a :: order-by-add
  assumes a:  $a + b \leq c$ 
  shows  $a \leq c$ 
  using less-add(1) a
  by (rule order-trans)

lemma add-est2:
  fixes a b c :: 'a :: order-by-add
  assumes a:  $a + b \leq c$ 
  shows  $b \leq c$ 
  using less-add(2) a

```


by (rule order-trans)

lemma star3':
 fixes a b x :: 'a :: kleene
 assumes a: $b + a * x \leq x$
 shows $star\ a * b \leq x$
 proof (rule order-trans)
 from a have $b \leq x$ by (rule add-est1)
 show $star\ a * b \leq star\ a * x$
 by (rule mult-mono) (auto simp: $b \leq x$)

 from a have $a * x \leq x$ by (rule add-est2)
 with star3 show $star\ a * x \leq x$.
 qed

lemma star4':
 fixes a b x :: 'a :: kleene
 assumes a: $b + x * a \leq x$
 shows $b * star\ a \leq x$
 proof (rule order-trans)
 from a have $b \leq x$ by (rule add-est1)
 show $b * star\ a \leq x * star\ a$
 by (rule mult-mono) (auto simp: $b \leq x$)

 from a have $x * a \leq x$ by (rule add-est2)
 with star4 show $x * star\ a \leq x$.
 qed

lemma star-idemp:
 fixes x :: 'a :: kleene
 shows $star\ (star\ x) = star\ x$
 oops

lemma star-unfold-left:
 fixes a :: 'a :: kleene
 shows $1 + a * star\ a = star\ a$
 proof (rule order-antisym, rule star1)

 have $1 + a * (1 + a * star\ a) \leq 1 + a * star\ a$
 apply (rule add-mono, rule)
 apply (rule mult-mono, auto)
 apply (rule star1)
 done

with star3' have $star\ a * 1 \leq 1 + a * star\ a$.
 thus $star\ a \leq 1 + a * star\ a$ by simp

qed

lemma *star-unfold-right*:

fixes $a :: 'a :: \text{kleene}$

shows $1 + \text{star } a * a = \text{star } a$

proof (*rule order-antisym, rule star2*)

have $1 + (1 + \text{star } a * a) * a \leq 1 + \text{star } a * a$

apply (*rule add-mono, rule*)

apply (*rule mult-mono, auto*)

apply (*rule star2*)

done

with star_4' **have** $1 * \text{star } a \leq 1 + \text{star } a * a$.

thus $\text{star } a \leq 1 + \text{star } a * a$ **by** *simp*

qed

lemma *star-zero[simp]*:

shows $\text{star } (0 :: 'a :: \text{kleene}) = 1$

by (*rule star-unfold-left[of 0, simplified]*)

lemma *star-commute*:

fixes $a \ b \ x :: 'a :: \text{kleene}$

assumes $a : a * x = x * b$

shows $\text{star } a * x = x * \text{star } b$

proof (*rule order-antisym*)

show $\text{star } a * x \leq x * \text{star } b$

proof (*rule star3', rule order-trans*)

from a **have** $a * x \leq x * b$ **by** *simp*

hence $a * x * \text{star } b \leq x * b * \text{star } b$

by (*rule mult-mono*) *auto*

thus $x + a * (x * \text{star } b) \leq x + x * b * \text{star } b$

using *add-mono* **by** (*auto simp: mult-assoc*)

show $\dots \leq x * \text{star } b$

proof –

have $x * (1 + b * \text{star } b) \leq x * \text{star } b$

by (*rule mult-mono[OF - star1]*) *auto*

thus *?thesis*

by (*simp add:right-distrib mult-assoc*)

qed

qed

show $x * \text{star } b \leq \text{star } a * x$

proof (*rule star4', rule order-trans*)

```

from  $a$  have  $b: x * b \leq a * x$  by simp
have  $star\ a * x * b \leq star\ a * a * x$ 
  unfolding mult-assoc
  by (rule mult-mono[OF - b]) auto
thus  $x + star\ a * x * b \leq x + star\ a * a * x$ 
  using add-mono by auto

show  $\dots \leq star\ a * x$ 
proof -
  have  $(1 + star\ a * a) * x \leq star\ a * x$ 
    by (rule mult-mono[OF star2]) auto
  thus ?thesis
    by (simp add:left-distrib mult-assoc)
qed
qed
qed

lemma star-assoc:
  fixes  $c\ d :: 'a :: kleene$ 
  shows  $star\ (c * d) * c = c * star\ (d * c)$ 
  by (auto simp:mult-assoc star-commute)

lemma star-dist:
  fixes  $a\ b :: 'a :: kleene$ 
  shows  $star\ (a + b) = star\ a * star\ (b * star\ a)$ 
  oops

lemma star-one:
  fixes  $a\ p\ p' :: 'a :: kleene$ 
  assumes  $p * p' = 1$  and  $p' * p = 1$ 
  shows  $p' * star\ a * p = star\ (p' * a * p)$ 
proof -
  from assms
  have  $p' * star\ a * p = p' * star\ (p * p' * a) * p$ 
    by simp
  also have  $\dots = p' * p * star\ (p' * a * p)$ 
    by (simp add: mult-assoc star-assoc)
  also have  $\dots = star\ (p' * a * p)$ 
    by (simp add: assms)
  finally show ?thesis .
qed

lemma star-mono:
  fixes  $x\ y :: 'a :: kleene$ 
  assumes  $x \leq y$ 
  shows  $star\ x \leq star\ y$ 
  oops

```

```

lemma x-less-star[simp]:
  fixes x :: 'a :: kleene
  shows  $x \leq x * \text{star } a$ 
proof -
  have  $x \leq x * (1 + a * \text{star } a)$  by (simp add:right-distrib)
  also have  $\dots = x * \text{star } a$  by (simp only: star-unfold-left)
  finally show ?thesis .
qed

```

2.1 Transitive Closure

definition

$\text{tcl } (x :: 'a :: \text{kleene}) = \text{star } x * x$

lemma *tcl-zero*:

$\text{tcl } (0 :: 'a :: \text{kleene}) = 0$
unfolding *tcl-def* **by** *simp*

lemma *tcl-unfold-right*: $\text{tcl } a = a + \text{tcl } a * a$

proof -
from *star-unfold-right*[*of a*]
have $a * (1 + \text{star } a * a) = a * \text{star } a$ **by** *simp*
from *this*[*simplified right-distrib, simplified*]
show ?thesis
by (*simp add:tcl-def star-commute mult-ac*)
qed

lemma *less-tcl*: $a \leq \text{tcl } a$

proof -
have $a \leq a + \text{tcl } a * a$ **by** *simp*
also have $\dots = \text{tcl } a$ **by** (*rule tcl-unfold-right[symmetric]*)
finally show ?thesis .
qed

2.2 Naive Algorithm to generate the transitive closure

function (*default* $\lambda x. 0$, *tailrec*, *domintros*)

$\text{mk-tcl} :: ('a :: \{\text{plus}, \text{times}, \text{ord}, \text{zero}\}) \Rightarrow 'a \Rightarrow 'a$

where

$\text{mk-tcl } A \ X = (\text{if } X * A \leq X \text{ then } X \text{ else } \text{mk-tcl } A \ (X + X * A))$
by *pat-completeness simp*

declare *mk-tcl.simps*[*simp del*]

lemma *mk-tcl-code*[*code*]:

$\text{mk-tcl } A \ X =$

```

    (let  $XA = X * A$ 
    in if  $XA \leq X$  then  $X$  else  $mk\text{-}tcl\ A\ (X + XA)$ )
    unfolding  $mk\text{-}tcl.simps[of\ A\ X]$  Let-def ..

lemma  $mk\text{-}tcl\text{-}lemma1$ :
  fixes  $X :: 'a :: kleene$ 
  shows  $(X + X * A) * star\ A = X * star\ A$ 
proof -
  have  $A * star\ A \leq 1 + A * star\ A$  by simp
  also have  $\dots = star\ A$  by (simp add:star-unfold-left)
  finally have  $star\ A + A * star\ A = star\ A$  by simp
  hence  $X * (star\ A + A * star\ A) = X * star\ A$  by simp
  thus ?thesis by (simp add:left-distrib right-distrib mult-ac)
qed

lemma  $mk\text{-}tcl\text{-}lemma2$ :
  fixes  $X :: 'a :: kleene$ 
  shows  $X * A \leq X \implies X * star\ A = X$ 
  by (rule order-antisym) (auto simp:star4)

lemma  $mk\text{-}tcl\text{-}correctness$ :
  fixes  $A\ X :: 'a :: \{kleene\}$ 
  assumes  $mk\text{-}tcl\text{-}dom\ (A, X)$ 
  shows  $mk\text{-}tcl\ A\ X = X * star\ A$ 
  using assms
  by induct (auto simp:mk-tcl-lemma1 mk-tcl-lemma2)

lemma  $graph\text{-}implies\text{-}dom$ :  $mk\text{-}tcl\text{-}graph\ x\ y \implies mk\text{-}tcl\text{-}dom\ x$ 
  by (rule mk-tcl-graph.induct) (auto intro:accp.accI elim:mk-tcl-rel.cases)

lemma  $mk\text{-}tcl\text{-}default$ :  $\neg mk\text{-}tcl\text{-}dom\ (a,x) \implies mk\text{-}tcl\ a\ x = 0$ 
  unfolding  $mk\text{-}tcl\text{-}def$ 
  by (rule fundef-default-value[OF mk-tcl-sum-def graph-implies-dom])

We can replace the dom-Condition of the correctness theorem with something executable

lemma  $mk\text{-}tcl\text{-}correctness2$ :
  fixes  $A\ X :: 'a :: \{kleene\}$ 
  assumes  $mk\text{-}tcl\ A\ A \neq 0$ 
  shows  $mk\text{-}tcl\ A\ A = tcl\ A$ 
  using assms mk-tcl-default mk-tcl-correctness
  unfolding  $tcl\text{-}def$ 
  by (auto simp:star-commute)

end

```

3 General Graphs as Sets

```
theory Graphs
imports Main Misc-Tools Kleene-Algebras
begin
```

3.1 Basic types, Size Change Graphs

```
datatype ('a, 'b) graph =
  Graph ('a × 'b × 'a) set

primrec dest-graph :: ('a, 'b) graph ⇒ ('a × 'b × 'a) set
  where dest-graph (Graph G) = G

lemma graph-dest-graph[simp]:
  Graph (dest-graph G) = G
  by (cases G) simp

lemma split-graph-all:
  (⋀gr. PROP P gr) ≡ (⋀set. PROP P (Graph set))
proof
  fix set
  assume ⋀gr. PROP P gr
  then show PROP P (Graph set) .
next
  fix gr
  assume ⋀set. PROP P (Graph set)
  then have PROP P (Graph (dest-graph gr)) .
  then show PROP P gr by simp
qed
```

```
definition
  has-edge :: ('n, 'e) graph ⇒ 'n ⇒ 'e ⇒ 'n ⇒ bool
  (- ⊢ - ∼> -)
where
  has-edge G n e n' = ((n, e, n') ∈ dest-graph G)
```

3.2 Graph composition

```
fun grcomp :: ('n, 'e::times) graph ⇒ ('n, 'e) graph ⇒ ('n, 'e) graph
where
  grcomp (Graph G) (Graph H) =
    Graph {(p, b, q) | p b q.
      (∃ k e e'. (p, e, k) ∈ G ∧ (k, e', q) ∈ H ∧ b = e * e')}
```

```
declare grcomp.simps[code del]
```

```
lemma graph-ext:
```

```

assumes  $\bigwedge n \ e \ n'. \text{has-edge } G \ n \ e \ n' = \text{has-edge } H \ n \ e \ n'$ 
shows  $G = H$ 
using assms
by (cases G, cases H) (auto simp:split-paired-all has-edge-def)

instantiation graph :: (type, type) comm-monoid-add
begin

definition
  graph-zero-def:  $0 = \text{Graph } \{\}$ 

definition
  graph-plus-def [code func del]:  $G + H = \text{Graph } (\text{dest-graph } G \cup \text{dest-graph } H)$ 

instance proof
  fix  $x \ y \ z :: ('a, 'b) \text{ graph}$ 
  show  $x + y + z = x + (y + z)$ 
  and  $x + y = y + x$ 
  and  $0 + x = x$ 
  unfolding graph-plus-def graph-zero-def by auto
qed

end

instantiation graph :: (type, type)  $\{\text{distrib-lattice}, \text{complete-lattice}\}$ 
begin

definition
  graph-leq-def [code func del]:  $G \leq H \longleftrightarrow \text{dest-graph } G \subseteq \text{dest-graph } H$ 

definition
  graph-less-def [code func del]:  $G < H \longleftrightarrow \text{dest-graph } G \subset \text{dest-graph } H$ 

definition
  [code func del]:  $\text{inf } G \ H = \text{Graph } (\text{dest-graph } G \cap \text{dest-graph } H)$ 

definition
  [code func del]:  $\text{sup } (G :: ('a, 'b) \text{ graph}) \ H = G + H$ 

definition
  Inf-graph-def [code func del]:  $\text{Inf} = (\lambda Gs. \text{Graph } (\bigcap (\text{dest-graph } `Gs)))$ 

definition
  Sup-graph-def [code func del]:  $\text{Sup} = (\lambda Gs. \text{Graph } (\bigcup (\text{dest-graph } `Gs)))$ 

instance proof
  fix  $x \ y \ z :: ('a, 'b) \text{ graph}$ 
  fix  $A :: ('a, 'b) \text{ graph set}$ 

```

```

show  $(x < y) = (x \leq y \wedge x \neq y)$ 
  unfolding graph-leq-def graph-less-def
  by (cases x, cases y) auto

show  $x \leq x$  unfolding graph-leq-def ..

{ assume  $x \leq y \ y \leq z$ 
  with order-trans show  $x \leq z$ 
  unfolding graph-leq-def . }

{ assume  $x \leq y \ y \leq x$  thus  $x = y$ 
  unfolding graph-leq-def
  by (cases x, cases y) simp }

show  $\inf x \ y \leq x \ \inf x \ y \leq y$ 
  unfolding inf-graph-def graph-leq-def
  by auto

{ assume  $x \leq y \ x \leq z$  thus  $x \leq \inf y \ z$ 
  unfolding inf-graph-def graph-leq-def
  by auto }

show  $x \leq \sup x \ y \ y \leq \sup x \ y$ 
  unfolding sup-graph-def graph-leq-def graph-plus-def by auto

{ assume  $y \leq x \ z \leq x$  thus  $\sup y \ z \leq x$ 
  unfolding sup-graph-def graph-leq-def graph-plus-def by auto }

show  $\sup x \ (\inf y \ z) = \inf (\sup x \ y) (\sup x \ z)$ 
  unfolding inf-graph-def sup-graph-def graph-leq-def graph-plus-def by auto

{ assume  $x \in A$  thus  $\inf A \leq x$ 
  unfolding Inf-graph-def graph-leq-def by auto }

{ assume  $\bigwedge x. x \in A \implies z \leq x$  thus  $z \leq \inf A$ 
  unfolding Inf-graph-def graph-leq-def by auto }

{ assume  $x \in A$  thus  $x \leq \sup A$ 
  unfolding Sup-graph-def graph-leq-def by auto }

{ assume  $\bigwedge x. x \in A \implies x \leq z$  thus  $\sup A \leq z$ 
  unfolding Sup-graph-def graph-leq-def by auto }
qed

end

lemma in-grplus:
  has-edge  $(G + H) \ p \ b \ q = (\text{has-edge } G \ p \ b \ q \vee \text{has-edge } H \ p \ b \ q)$ 

```



```

    by (cases G, cases H, auto simp:has-edge-def graph-plus-def)

lemma in-grzero:
  has-edge 0 p b q = False
  by (simp add:graph-zero-def has-edge-def)



### 3.2.1 Multiplicative Structure



instantiation graph :: (type, times) mult-zero
begin

definition
  graph-mult-def [code func del]:  $G * H = grcomp\ G\ H$ 

instance proof
  fix a :: ('a, 'b) graph

  show 0 * a = 0
    unfolding graph-mult-def graph-zero-def
    by (cases a) (simp add:grcomp.simps)
  show a * 0 = 0
    unfolding graph-mult-def graph-zero-def
    by (cases a) (simp add:grcomp.simps)
qed

end

instantiation graph :: (type, one) one
begin

definition
  graph-one-def:  $1 = Graph\ \{ (x, 1, x) \mid x. True \}$ 

instance ..

end

lemma in-grcomp:
  has-edge (G * H) p b q
  = ( $\exists k\ e\ e'. has-edge\ G\ p\ e\ k \wedge has-edge\ H\ k\ e'\ q \wedge b = e * e'$ )
  by (cases G, cases H) (auto simp:graph-mult-def has-edge-def image-def)

lemma in-grunit:
  has-edge 1 p b q = (p = q  $\wedge$  b = 1)
  by (auto simp:graph-one-def has-edge-def)

instance graph :: (type, semigroup-mult) semigroup-mult
proof
  fix G1 G2 G3 :: ('a,'b) graph

```

```

show  $G1 * G2 * G3 = G1 * (G2 * G3)$ 
proof (rule graph-ext, rule trans)
  fix  $p J q$ 
  show  $\text{has-edge } ((G1 * G2) * G3) p J q =$ 
     $(\exists G i H j I.$ 
       $\text{has-edge } G1 p G i$ 
       $\wedge \text{has-edge } G2 i H j$ 
       $\wedge \text{has-edge } G3 j I q$ 
       $\wedge J = (G * H) * I)$ 
    by (simp only: in-grcomp) blast
  show  $\dots = \text{has-edge } (G1 * (G2 * G3)) p J q$ 
    by (simp only: in-grcomp mult-assoc) blast
qed
qed

instantiation graph :: (type, monoid-mult) {semiring-1, idem-add, recpower, star}
begin

primrec power-graph :: ('a::type, 'b::monoid-mult) graph  $\Rightarrow$  nat  $\Rightarrow$  ('a, 'b) graph
where
   $(A :: ('a, 'b) \text{ graph}) ^ 0 = 1$ 
   $| (A :: ('a, 'b) \text{ graph}) ^ \text{Suc } n = A * (A ^ n)$ 

definition
  graph-star-def:  $\text{star } (G :: ('a, 'b) \text{ graph}) = (\text{SUP } n. G ^ n)$ 

instance proof
  fix  $a b c :: ('a, 'b) \text{ graph}$ 

  show  $1 * a = a$ 
    by (rule graph-ext) (auto simp: in-grcomp in-grunit)
  show  $a * 1 = a$ 
    by (rule graph-ext) (auto simp: in-grcomp in-grunit)

  show  $(a + b) * c = a * c + b * c$ 
    by (rule graph-ext, simp add: in-grcomp in-grplus) blast
  show  $a * (b + c) = a * b + a * c$ 
    by (rule graph-ext, simp add: in-grcomp in-grplus) blast

  show  $(0 :: ('a, 'b) \text{ graph}) \neq 1$  unfolding graph-zero-def graph-one-def
    by simp

  show  $a + a = a$  unfolding graph-plus-def by simp

  show  $a ^ 0 = 1 \wedge n. a ^ (\text{Suc } n) = a * a ^ n$ 
    by simp-all
qed

```

end

lemma *graph-leqI*:
assumes $\bigwedge n\ e\ n'. \text{has-edge } G\ n\ e\ n' \implies \text{has-edge } H\ n\ e\ n'$
shows $G \leq H$
using *assms*
unfolding *graph-leq-def has-edge-def*
by *auto*

lemma *in-graph-plusE*:
assumes $\text{has-edge } (G + H)\ n\ e\ n'$
assumes $\text{has-edge } G\ n\ e\ n' \implies P$
assumes $\text{has-edge } H\ n\ e\ n' \implies P$
shows P
using *assms*
by (*auto simp: in-grplus*)

lemma *in-graph-compE*:
assumes $GH: \text{has-edge } (G * H)\ n\ e\ n'$
obtains $e1\ k\ e2$
where $\text{has-edge } G\ n\ e1\ k\ \text{has-edge } H\ k\ e2\ n'\ e = e1 * e2$
using GH
by (*auto simp: in-grcomp*)

lemma
assumes $x \in S\ k$
shows $x \in (\bigcup k. S\ k)$
using *assms* **by** *blast*

lemma *graph-union-least*:
assumes $\bigwedge n. \text{Graph } (G\ n) \leq C$
shows $\text{Graph } (\bigcup n. G\ n) \leq C$
using *assms* **unfolding** *graph-leq-def*
by *auto*

lemma *Sup-graph-eq*:
 $(\text{SUP } n. \text{Graph } (G\ n)) = \text{Graph } (\bigcup n. G\ n)$
proof (*rule order-antisym*)
show $(\text{SUP } n. \text{Graph } (G\ n)) \leq \text{Graph } (\bigcup n. G\ n)$
by (*rule SUP-leI*) (*auto simp add: graph-leq-def*)

show $\text{Graph } (\bigcup n. G\ n) \leq (\text{SUP } n. \text{Graph } (G\ n))$
by (*rule graph-union-least, rule le-SUPI', rule*)
qed

lemma *has-edge-leq*: $\text{has-edge } G\ p\ b\ q = (\text{Graph } \{(p,b,q)\} \leq G)$
unfolding *has-edge-def graph-leq-def*
by (*cases G*) *simp*

```

lemma Sup-graph-eq2:
  (SUP n. G n) = Graph ( $\bigcup$  n. dest-graph (G n))
  using Sup-graph-eq[of  $\lambda n$ . dest-graph (G n), simplified]
  by simp

lemma in-SUP:
  has-edge (SUP x. G s x) p b q = ( $\exists$  x. has-edge (G s x) p b q)
  unfolding Sup-graph-eq2 has-edge-leq graph-leq-def
  by simp

instance graph :: (type, monoid-mult) kleene-by-complete-lattice
proof
  fix a b c :: ('a', 'b') graph

  show  $a \leq b \iff a + b = b$  unfolding graph-leq-def graph-plus-def
    by (cases a, cases b) auto

  from order-less-le show  $a < b \iff a \leq b \wedge a \neq b$  .

  show  $a * \text{star } b * c = (\text{SUP } n. a * b ^ n * c)$ 
    unfolding graph-star-def
    by (rule graph-ext) (force simp:in-SUP in-grcomp)
qed

```

```

lemma in-star:
  has-edge (star G) a x b = ( $\exists$  n. has-edge (G ^ n) a x b)
  by (auto simp:graph-star-def in-SUP)

```

```

lemma tcl-is-SUP:
  tcl (G::('a::type, 'b::monoid-mult) graph) =
    (SUP n. G ^ (Suc n))
  unfolding tcl-def
  using star-cont[of 1 G G]
  by (simp add:power-Suc power-commutes)

```

```

lemma in-tcl:
  has-edge (tcl G) a x b = ( $\exists$   $n > 0$ . has-edge (G ^ n) a x b)
  apply (auto simp: tcl-is-SUP in-SUP simp del: power-graph.simps)
  apply (rule-tac  $x = n - 1$  in exI, auto)
  done

```

3.3 Infinite Paths

```

types ('n', 'e') ipath = ('n × 'e') sequence

```

definition $has-ipath :: ('n, 'e) graph \Rightarrow ('n, 'e) ipath \Rightarrow bool$

where

$has-ipath\ G\ p =$
 $(\forall i. has-edge\ G\ (fst\ (p\ i))\ (snd\ (p\ i))\ (fst\ (p\ (Suc\ i))))$

3.4 Finite Paths

types $('n, 'e) fpath = ('n \times ('e \times 'n) list)$

inductive $has-fpath :: ('n, 'e) graph \Rightarrow ('n, 'e) fpath \Rightarrow bool$

for $G :: ('n, 'e) graph$

where

$has-fpath-empty: has-fpath\ G\ (n, [])$
 $| has-fpath-join: \llbracket G \vdash n \rightsquigarrow^e n'; has-fpath\ G\ (n', es) \rrbracket \Longrightarrow has-fpath\ G\ (n, (e, n')\#es)$

definition

$end-node\ p =$
 $(if\ snd\ p = []\ then\ fst\ p\ else\ snd\ (snd\ p\ !\ (length\ (snd\ p) - 1)))$

definition $path-nth :: ('n, 'e) fpath \Rightarrow nat \Rightarrow ('n \times 'e \times 'n)$

where

$path-nth\ p\ k = (if\ k = 0\ then\ fst\ p\ else\ snd\ (snd\ p\ !\ (k - 1)), snd\ p\ !\ k)$

lemma $endnode-nth:$

assumes $length\ (snd\ p) = Suc\ k$
shows $end-node\ p = snd\ (snd\ (path-nth\ p\ k))$
using $assms$ **unfolding** $end-node-def\ path-nth-def$
by $auto$

lemma $path-nth-graph:$

assumes $k < length\ (snd\ p)$
assumes $has-fpath\ G\ p$
shows $(\lambda(n,e,n'). has-edge\ G\ n\ e\ n')\ (path-nth\ p\ k)$

using $assms$

proof $(induct\ k\ arbitrary: p)$

case 0 **thus** $?case$

unfolding $path-nth-def$ **by** $(auto\ elim:has-fpath.cases)$

next

case $(Suc\ k\ p)$

from $\langle has-fpath\ G\ p \rangle$ **show** $?case$

proof $(rule\ has-fpath.cases)$

case $goal1$ **with** Suc **show** $?case$ **by** $simp$

next

fix $n\ e\ n'\ es$

assume $st: p = (n, (e, n')\# es)$

$G \vdash n \rightsquigarrow^e n'$

$has-fpath\ G\ (n', es)$

```

    with Suc
    have  $(\lambda(n, b, a). G \vdash n \rightsquigarrow^b a)$  (path-nth (n', es) k) by simp
    with st show ?thesis by (cases k, auto simp: path-nth-def)
  qed
qed

lemma path-nth-connected:
  assumes Suc k < length (snd p)
  shows fst (path-nth p (Suc k)) = snd (snd (path-nth p k))
  using assms
  unfolding path-nth-def
  by auto

definition path-loop :: ('n, 'e) fpath  $\Rightarrow$  ('n, 'e) ipath (omega)
where
  omega p  $\equiv$  ( $\lambda i. (\lambda(n,e,n'). (n,e))$  (path-nth p (i mod (length (snd p)))))

lemma fst-p0: fst (path-nth p 0) = fst p
  unfolding path-nth-def by simp

lemma path-loop-connect:
  assumes fst p = end-node p
  and 0 < length (snd p) (is 0 < ?l)
  shows fst (path-nth p (Suc i mod (length (snd p))))
  = snd (snd (path-nth p (i mod length (snd p))))
  (is ... = snd (snd (path-nth p ?k)))
proof -
  from 0 < ?l have i mod ?l < ?l (is ?k < ?l)
  by simp

  show ?thesis
  proof (cases Suc ?k < ?l)
    case True
    hence Suc ?k  $\neq$  ?l by simp
    with path-nth-connected[OF True]
    show ?thesis
    by (simp add: mod-Suc)
  next
    case False
    with ?k < ?l have wrap: Suc ?k = ?l by simp

    hence fst (path-nth p (Suc i mod ?l)) = fst (path-nth p 0)
    by (simp add: mod-Suc)
    also from fst-p0 have ... = fst p .
    also have ... = end-node p by fact
    also have ... = snd (snd (path-nth p ?k))
    by (auto simp: endnode-nth wrap)
    finally show ?thesis .
  qed
qed

```

qed

lemma *path-loop-graph*:

assumes *has-fpath* $G\ p$
and *loop*: $\text{fst } p = \text{end-node } p$
and *nonempty*: $0 < \text{length } (\text{snd } p)$ (**is** $0 < ?l$)
shows *has-ipath* $G\ (\text{omega } p)$

proof –

```
{
  fix i
  from  $0 < ?l$  have  $i \bmod ?l < ?l$  (is  $?k < ?l$ )
  by simp
  from this and  $\langle \text{has-fpath } G\ p \rangle$ 
  have  $pk\text{-}G$ :  $(\lambda(n,e,n'). \text{has-edge } G\ n\ e\ n')\ (\text{path-nth } p\ ?k)$ 
  by (rule path-nth-graph)

  from path-loop-connect[OF loop nonempty]  $pk\text{-}G$ 
  have  $\text{has-edge } G\ (\text{fst } (\text{omega } p\ i))\ (\text{snd } (\text{omega } p\ i))\ (\text{fst } (\text{omega } p\ (\text{Suc } i)))$ 
  unfolding path-loop-def has-edge-def split-def
  by simp
}
```

then show *?thesis* **by** (auto simp:*has-ipath-def*)

qed

definition *prod* :: $(n, 'e::\text{monoid-mult})\ \text{fpath} \Rightarrow 'e$

where

$\text{prod } p = \text{foldr } (op\ *)\ (\text{map } \text{fst } (\text{snd } p))\ 1$

lemma *prod-simps*[*simp*]:

$\text{prod } (n, []) = 1$
 $\text{prod } (n, (e,n')\#es) = e * (\text{prod } (n',es))$

unfolding *prod-def*

by *simp-all*

lemma *power-induces-path*:

assumes *a*: $\text{has-edge } (A\ ^k)\ n\ G\ m$

obtains *p*

where *has-fpath* $A\ p$
and $n = \text{fst } p\ m = \text{end-node } p$
and $G = \text{prod } p$
and $k = \text{length } (\text{snd } p)$

using *a*

proof (*induct k arbitrary:m n G thesis*)

case $(0\ m\ n\ G)$

let $?p = (n, [])$

from 0 **have** $\text{has-fpath } A\ ?p\ m = \text{end-node } ?p\ G = \text{prod } ?p$

by (auto simp:*in-grunit end-node-def* intro:*has-fpath.intros*)

thus *?case* **using** 0 **by** (auto simp:*end-node-def*)

next

```

case (Suc k m n G)
hence has-edge (A * A ^ k) n G m
  by (simp add:power-Suc power-commutes)
then obtain G' H j where
  a-A: has-edge A n G' j
  and H-pow: has-edge (A ^ k) j H m
  and [simp]: G = G' * H
  by (auto simp:in-grcomp)

from H-pow and Suc
obtain p
  where p-path: has-fpath A p
  and [simp]: j = fst p m = end-node p H = prod p
  k = length (snd p)
  by blast

let ?p' = (n, (G', j)#snd p)
from a-A and p-path
have has-fpath A ?p' m = end-node ?p' G = prod ?p'
  by (auto simp:end-node-def nth.simps intro:has-fpath.intros split:nat.split)
thus ?case using Suc by auto
qed

```

3.5 Sub-Paths

definition *sub-path* :: ('n, 'e) ipath \Rightarrow nat \Rightarrow nat \Rightarrow ('n, 'e) fpath
 ((-(-,-)))
where
 $p\langle i,j \rangle =$
 $(fst (p i), map (\lambda k. (snd (p k), fst (p (Suc k)))) [i ..< j])$

lemma *sub-path-is-path*:
assumes ipath: has-ipath G p
assumes l: $i \leq j$
shows has-fpath G (p⟨i,j⟩)
using l
proof (induct i rule:inc-induct)
case base **show** ?case **by** (auto simp:sub-path-def intro:has-fpath.intros)
next
case (step i)
with ipath upt-rec[of i j]
show ?case
by (auto simp:sub-path-def has-ipath-def intro:has-fpath.intros)
qed

lemma *sub-path-start*[simp]:
 $fst (p\langle i,j \rangle) = fst (p i)$


```

by (simp add:sub-path-def)

lemma nth-upto[simp]:  $k < j - i \implies [i ..< j] ! k = i + k$ 
  by (induct k) auto

lemma sub-path-end[simp]:
   $i < j \implies \text{end-node } (p\langle i,j \rangle) = \text{fst } (p\ j)$ 
  by (auto simp:sub-path-def end-node-def)

lemma foldr-map:  $\text{foldr } f \ (\text{map } g\ xs) = \text{foldr } (f\ o\ g)\ xs$ 
  by (induct xs) auto

lemma upto-append[simp]:
  assumes  $i \leq j \leq k$ 
  shows  $[i ..< j] @ [j ..< k] = [i ..< k]$ 
  using assms and upt-add-eq-append[of i j k - j]
  by simp

lemma foldr-monoid:  $\text{foldr } (op\ *)\ xs\ 1 = \text{foldr } (op\ *)\ ys\ 1$ 
   $= \text{foldr } (op\ *)\ (xs\ @\ ys)\ (1::'a::\text{monoid-mult})$ 
  by (induct xs) (auto simp:mult-assoc)

lemma sub-path-prod:
  assumes  $i < j$ 
  assumes  $j < k$ 
  shows  $\text{prod } (p\langle i,k \rangle) = \text{prod } (p\langle i,j \rangle) * \text{prod } (p\langle j,k \rangle)$ 
  using assms
  unfolding prod-def sub-path-def
  by (simp add:map-compose[symmetric] comp-def)
  (simp only:foldr-monoid map-append[symmetric] upto-append)

lemma path-acgpow-aux:
  assumes  $\text{length } es = l$ 
  assumes  $\text{has-fpath } G\ (n, es)$ 
  shows  $\text{has-edge } (G\ ^\ l)\ n\ (\text{prod } (n, es))\ (\text{end-node } (n, es))$ 
  using assms
  proof (induct l arbitrary:n es)
    case 0 thus ?case
      by (simp add:in-grunit end-node-def)
  next
    case (Suc l n es)
    hence  $es \neq []$  by auto
    let ?n' =  $\text{snd } (\text{hd } es)$ 
    let ?es' =  $\text{tl } es$ 
    let ?e =  $\text{fst } (\text{hd } es)$ 

    from Suc have len:  $\text{length } ?es' = l$  by auto

```

```

from Suc
have [simp]: end-node (n, es) = end-node (?n', ?es')
  by (cases es) (auto simp: end-node-def nth.simps split: nat.split)

from ⟨has-fpath G (n, es)⟩
have has-fpath G (?n', ?es')
  by (rule has-fpath.cases) (auto intro: has-fpath.intros)
with Suc len
have has-edge (G ^ l) ?n' (prod (?n', ?es') (end-node (?n', ?es')))
  by auto
moreover
from ⟨es ≠ []⟩
have prod (n, es) = ?e * (prod (?n', ?es'))
  by (cases es) auto
moreover
from ⟨has-fpath G (n, es)⟩ have c: has-edge G n ?e ?n'
  by (rule has-fpath.cases) (insert ⟨es ≠ []⟩, auto)

ultimately
show ?case
  unfolding power-Suc
  by (auto simp: in-grcomp)
qed

```

```

lemma path-acgpow:
  has-fpath G p
   $\implies$  has-edge (G ^ length (snd p)) (fst p) (prod p) (end-node p)
by (cases p)
  (rule path-acgpow-aux[of snd p length (snd p) - fst p, simplified])

```

```

lemma star-paths:
  has-edge (star G) a x b =
    ( $\exists p. \text{has-fpath } G \ p \wedge a = \text{fst } p \wedge b = \text{end-node } p \wedge x = \text{prod } p$ )
proof
  assume has-edge (star G) a x b
  then obtain n where pow: has-edge (G ^ n) a x b
    by (auto simp: in-star)

  then obtain p where
    has-fpath G p a = fst p b = end-node p x = prod p
    by (rule power-induces-path)

```

```

  thus  $\exists p. \text{has-fpath } G \ p \wedge a = \text{fst } p \wedge b = \text{end-node } p \wedge x = \text{prod } p$ 
    by blast
next
  assume  $\exists p. \text{has-fpath } G \ p \wedge a = \text{fst } p \wedge b = \text{end-node } p \wedge x = \text{prod } p$ 
  then obtain p where

```

$\text{has-fpath } G \ p \ a = \text{fst } p \ b = \text{end-node } p \ x = \text{prod } p$
by *blast*

hence $\text{has-edge } (G \hat{\ } \text{length } (\text{snd } p)) \ a \ x \ b$
by *(auto intro:path-acgpow)*

thus $\text{has-edge } (\text{star } G) \ a \ x \ b$
by *(auto simp:in-star)*

qed

lemma *plus-paths*:

$\text{has-edge } (\text{tcl } G) \ a \ x \ b =$
 $(\exists p. \text{has-fpath } G \ p \wedge a = \text{fst } p \wedge b = \text{end-node } p \wedge x = \text{prod } p \wedge 0 < \text{length}$
 $(\text{snd } p))$

proof

assume $\text{has-edge } (\text{tcl } G) \ a \ x \ b$

then obtain n **where** $\text{pow: has-edge } (G \hat{\ } n) \ a \ x \ b$ **and** $0 < n$
by *(auto simp:in-tcl)*

from pow **obtain** p **where**
 $\text{has-fpath } G \ p \ a = \text{fst } p \ b = \text{end-node } p \ x = \text{prod } p$
 $n = \text{length } (\text{snd } p)$
by *(rule power-induces-path)*

with $\langle 0 < n \rangle$
show $\exists p. \text{has-fpath } G \ p \wedge a = \text{fst } p \wedge b = \text{end-node } p \wedge x = \text{prod } p \wedge 0 <$
 $\text{length } (\text{snd } p)$
by *blast*

next

assume $\exists p. \text{has-fpath } G \ p \wedge a = \text{fst } p \wedge b = \text{end-node } p \wedge x = \text{prod } p$
 $\wedge 0 < \text{length } (\text{snd } p)$

then obtain p **where**
 $\text{has-fpath } G \ p \ a = \text{fst } p \ b = \text{end-node } p \ x = \text{prod } p$
 $0 < \text{length } (\text{snd } p)$
by *blast*

hence $\text{has-edge } (G \hat{\ } \text{length } (\text{snd } p)) \ a \ x \ b$
by *(auto intro:path-acgpow)*

with $\langle 0 < \text{length } (\text{snd } p) \rangle$
show $\text{has-edge } (\text{tcl } G) \ a \ x \ b$
by *(auto simp:in-tcl)*

qed

definition

$\text{contract } s \ p =$

$(\lambda i. (fst (p (s i)), prod (p \langle s i, s (Suc i) \rangle)))$

lemma *ipath-contract*:

assumes *[simp]*: *increasing s*

assumes *ipath*: *has-ipath G p*

shows *has-ipath (tcl G) (contract s p)*

unfolding *has-ipath-def*

proof

fix *i*

let *?p* = *p \langle s i, s (Suc i) \rangle*

from *increasing-strict*

have *fst (p (s (Suc i))) = end-node ?p* **by** *simp*

moreover

from *increasing-strict[of s i Suc i]* **have** *snd ?p ≠ []*

by (*simp add:sub-path-def*)

moreover

from *ipath increasing-weak[of s]* **have** *has-fpath G ?p*

by (*rule sub-path-is-path*) *auto*

ultimately

show *has-edge (tcl G)*

(fst (contract s p i)) (snd (contract s p i)) (fst (contract s p (Suc i)))

unfolding *contract-def plus-paths*

by (*intro exI*) *auto*

qed

lemma *prod-unfold*:

i < j ⇒ prod (p \langle i, j \rangle)

*= snd (p i) * prod (p \langle Suc i, j \rangle)*

unfolding *prod-def*

by (*simp add:sub-path-def upt-rec[of i j]*)

lemma *sub-path-loop*:

assumes *0 < k*

assumes *k*: *k = length (snd loop)*

assumes *loop*: *fst loop = end-node loop*

shows *(omega loop) \langle k * i, k * Suc i \rangle = loop* (**is** *?ω = loop*)

proof (*rule prod-eqI*)

show *fst ?ω = fst loop*

by (*auto simp:path-loop-def path-nth-def split-def k*)

show *snd ?ω = snd loop*

proof (*rule nth-equalityI[rule-format]*)

show *leneq: length (snd ?ω) = length (snd loop)*

unfolding *sub-path-def k* **by** *simp*

fix *j* **assume** *j < length (snd (?ω))*

with *leneq* **and** *k* **have** *j < k* **by** *simp*

```

have a:  $\bigwedge i. \text{fst } (\text{path-nth loop } (\text{Suc } i \text{ mod } k))$ 
  = snd (snd (path-nth loop (i mod k)))
  unfolding k
  apply (rule path-loop-connect[OF loop])
  using  $\langle 0 < k \rangle$  and k
  apply auto
  done

from  $\langle j < k \rangle$ 
show snd ? $\omega$  ! j = snd loop ! j
  unfolding sub-path-def
  apply (simp add:path-loop-def split-def add-ac)
  apply (simp add:a k[symmetric])
  apply (simp add:path-nth-def)
  done
qed
qed
end

```

4 The Size-Change Principle (Definition)

```

theory Criterion
imports Graphs Infinite-Set
begin

```

4.1 Size-Change Graphs

```

datatype sedge =
  LESS ( $\downarrow$ )
| LEQ ( $\Downarrow$ )

```

```

instantiation sedge :: comm-monoid-mult
begin

```

```

definition
  one-sedge-def:  $1 = \Downarrow$ 

```

```

definition
  mult-sedge-def:  $a * b = (\text{if } a = \downarrow \text{ then } \downarrow \text{ else } b)$ 

```

```

instance proof
  fix a b c :: sedge
  show a * b * c = a * (b * c) by (simp add:mult-sedge-def)
  show 1 * a = a by (simp add:mult-sedge-def one-sedge-def)
  show a * b = b * a unfolding mult-sedge-def
    by (cases a, simp) (cases b, auto)

```

qed

end

lemma *sedg-UNIV*:

$UNIV = \{ LESS, LEQ \}$

proof (*intro equalityI subsetI*)

fix x **show** $x \in \{ LESS, LEQ \}$

by (*cases x*) *auto*

qed *auto*

instance *sedg :: finite*

proof

show *finite (UNIV::sedg set)*

by (*simp add: sedg-UNIV*)

qed

types $'a\ scg = ('a, sedg)\ graph$

types $'a\ acg = ('a, 'a\ scg)\ graph$

4.2 Size-Change Termination

abbreviation (*input*)

$desc\ P\ Q == ((\exists n. \forall i \geq n. P\ i) \wedge (\exists \infty i. Q\ i))$

abbreviation (*input*)

$dsc\ G\ i\ j \equiv has-edge\ G\ i\ LESS\ j$

abbreviation (*input*)

$eqp\ G\ i\ j \equiv has-edge\ G\ i\ LEQ\ j$

abbreviation

$eql :: 'a\ scg \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$

$(- \vdash - \rightsquigarrow -)$

where

$eql\ G\ i\ j \equiv has-edge\ G\ i\ LESS\ j \vee has-edge\ G\ i\ LEQ\ j$

abbreviation (*input*) $descat :: ('a, 'a\ scg)\ ipath \Rightarrow 'a\ sequence \Rightarrow nat \Rightarrow bool$

where

$descat\ p\ \vartheta\ i \equiv has-edge\ (snd\ (p\ i))\ (\vartheta\ i)\ LESS\ (\vartheta\ (Suc\ i))$

abbreviation (*input*) $eqat :: ('a, 'a\ scg)\ ipath \Rightarrow 'a\ sequence \Rightarrow nat \Rightarrow bool$

where

$eqat\ p\ \vartheta\ i \equiv has-edge\ (snd\ (p\ i))\ (\vartheta\ i)\ LEQ\ (\vartheta\ (Suc\ i))$

abbreviation $(input) \text{ eqlat} :: ('a, 'a \text{ scg}) \text{ ipath} \Rightarrow 'a \text{ sequence} \Rightarrow nat \Rightarrow bool$
where

$\text{eqlat } p \ \vartheta \ i \equiv (\text{has-edge } (snd \ (p \ i)) \ (\vartheta \ i) \text{ LESS } (\vartheta \ (Suc \ i)))$
 $\vee \text{has-edge } (snd \ (p \ i)) \ (\vartheta \ i) \text{ LEQ } (\vartheta \ (Suc \ i)))$

definition $\text{is-desc-thread} :: 'a \text{ sequence} \Rightarrow ('a, 'a \text{ scg}) \text{ ipath} \Rightarrow bool$
where

$\text{is-desc-thread } \vartheta \ p = ((\exists n. \forall i \geq n. \text{eqlat } p \ \vartheta \ i) \wedge (\exists_{\infty} i. \text{descat } p \ \vartheta \ i))$

definition $\text{SCT} :: 'a \text{ acg} \Rightarrow bool$
where

$\text{SCT } \mathcal{A} =$
 $(\forall p. \text{has-ipath } \mathcal{A} \ p \longrightarrow (\exists \vartheta. \text{is-desc-thread } \vartheta \ p))$

definition $\text{no-bad-graphs} :: 'a \text{ acg} \Rightarrow bool$
where

$\text{no-bad-graphs } A =$
 $(\forall n \ G. \text{has-edge } A \ n \ G \ n \wedge G * G = G$
 $\longrightarrow (\exists p. \text{has-edge } G \ p \text{ LESS } p))$

definition $\text{SCT}' :: 'a \text{ acg} \Rightarrow bool$
where

$\text{SCT}' \ A = \text{no-bad-graphs } (tcl \ A)$

end

5 Proof of the Size-Change Principle

theory *Correctness*
imports *Main Ramsey Misc-Tools Criterion*
begin

5.1 Auxiliary definitions

definition $\text{is-thread} :: nat \Rightarrow 'a \text{ sequence} \Rightarrow ('a, 'a \text{ scg}) \text{ ipath} \Rightarrow bool$
where

$\text{is-thread } n \ \vartheta \ p = (\forall i \geq n. \text{eqlat } p \ \vartheta \ i)$

definition $\text{is-fthread} ::$
 $'a \text{ sequence} \Rightarrow ('a, 'a \text{ scg}) \text{ ipath} \Rightarrow nat \Rightarrow nat \Rightarrow bool$
where

$\text{is-fthread } \vartheta \ mp \ i \ j = (\forall k \in \{i..<j\}. \text{eqlat } mp \ \vartheta \ k)$

definition $\text{is-desc-fthread} ::$

$'a \text{ sequence} \Rightarrow ('a, 'a \text{ scg}) \text{ ipath} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$
where
 $\text{is-desc-fthread } \vartheta \text{ mp } i \text{ j} =$
 $(\text{is-fthread } \vartheta \text{ mp } i \text{ j} \wedge$
 $(\exists k \in \{i..<j\}. \text{descat mp } \vartheta \text{ k}))$

definition

$\text{has-fth } p \text{ i j n m} =$
 $(\exists \vartheta. \text{is-fthread } \vartheta \text{ p i j} \wedge \vartheta \text{ i} = n \wedge \vartheta \text{ j} = m)$

definition

$\text{has-desc-fth } p \text{ i j n m} =$
 $(\exists \vartheta. \text{is-desc-fthread } \vartheta \text{ p i j} \wedge \vartheta \text{ i} = n \wedge \vartheta \text{ j} = m)$

5.2 Everything is finite

lemma *finite-range*:

fixes $f :: \text{nat} \Rightarrow 'a$
assumes $\text{fin}: \text{finite } (\text{range } f)$
shows $\exists x. \exists_{\infty} i. f \text{ i} = x$

proof (*rule classical*)

assume $\neg(\exists x. \exists_{\infty} i. f \text{ i} = x)$

hence $\forall x. \exists j. \forall i > j. f \text{ i} \neq x$

unfolding *INF-nat* **by** *blast*

with *choice*

have $\exists j. \forall x. \forall i > (j \text{ x}). f \text{ i} \neq x$.

then obtain j **where**

$\text{neq}: \bigwedge x \text{ i}. j \text{ x} < i \implies f \text{ i} \neq x$ **by** *blast*

from fin **have** $\text{finite } (\text{range } (j \text{ o } f))$

by (*auto simp: comp-def*)

with *finite-nat-bounded*

obtain m **where** $\text{range } (j \text{ o } f) \subseteq \{..<m\}$ **by** *blast*

hence $j \text{ (f m)} < m$ **unfolding** *comp-def* **by** *auto*

with $\text{neq}[of \text{ f m m}]$ **show** *?thesis* **by** *blast*

qed

lemma *finite-range-ignore-prefix*:

fixes $f :: \text{nat} \Rightarrow 'a$

assumes $fA: \text{finite } A$

assumes $\text{inA}: \forall x \geq n. f \text{ x} \in A$

shows $\text{finite } (\text{range } f)$

proof –

have $a: \text{UNIV} = \{0 ..< (n::\text{nat})\} \cup \{x. n \leq x\}$ **by** *auto*

have $b: \text{range } f = f \text{ ` } \{0 ..< n\} \cup f \text{ ` } \{x. n \leq x\}$

(*is ... = ?A* \cup *?B*)

by (*unfold a*) (*simp add: image-Un*)

have *finite* ?*A* **by** (rule *finite-imageI*) *simp*
moreover
from *inA* **have** ?*B* \subseteq *A* **by** *auto*
from *this fA* **have** *finite* ?*B* **by** (rule *finite-subset*)
ultimately show ?*thesis* **using** *b* **by** *simp*
qed

definition

finite-graph *G* = *finite* (*dest-graph* *G*)

definition

all-finite *G* = ($\forall n\ H\ m. \text{has-edge } G\ n\ H\ m \longrightarrow \text{finite-graph } H$)

definition

finite-acg *A* = (*finite-graph* *A* \wedge *all-finite* *A*)

definition

nodes *G* = *fst* ‘ *dest-graph* *G* \cup *snd* ‘ *snd* ‘ *dest-graph* *G*

definition

edges *G* = *fst* ‘ *snd* ‘ *dest-graph* *G*

definition

smallnodes *G* = $\bigcup (\text{nodes } ' \text{ edges } G)$

lemma *thread-image-nodes*:

assumes *th*: *is-thread* *n* ϑ *p*

shows $\forall i \geq n. \vartheta\ i \in \text{nodes } (\text{snd } (p\ i))$

using *prems*

unfolding *is-thread-def* *has-edge-def* *nodes-def*

by *force*

lemma *finite-nodes*: *finite-graph* *G* \implies *finite* (*nodes* *G*)

unfolding *finite-graph-def* *nodes-def*

by *auto*

lemma *nodes-subgraph*: *A* \leq *B* \implies *nodes* *A* \subseteq *nodes* *B*

unfolding *graph-leq-def* *nodes-def*

by *auto*

lemma *finite-edges*: *finite-graph* *G* \implies *finite* (*edges* *G*)

unfolding *finite-graph-def* *edges-def*

by *auto*

lemma *edges-sum*[*simp*]: *edges* (*A* + *B*) = *edges* *A* \cup *edges* *B*

unfolding *edges-def* *graph-plus-def*

by *auto*

lemma *nodes-sum*[*simp*]: *nodes* (*A* + *B*) = *nodes* *A* \cup *nodes* *B*

unfolding *nodes-def* *graph-plus-def*

by *auto*

```

lemma finite-acg-subset:
   $A \leq B \implies \text{finite-acg } B \implies \text{finite-acg } A$ 
  unfolding finite-acg-def finite-graph-def all-finite-def
  has-edge-def graph-leq-def
  by (auto elim:finite-subset)

lemma scg-finite:
  fixes  $G :: 'a \text{ scg}$ 
  assumes  $\text{fin: finite (nodes } G)$ 
  shows finite-graph G
  unfolding finite-graph-def
proof (rule finite-subset)
  show  $\text{dest-graph } G \subseteq \text{nodes } G \times \text{UNIV} \times \text{nodes } G$  (is -  $\subseteq$  ?P)
    unfolding nodes-def
    by force
  show finite ?P
    by (intro finite-cartesian-product fin finite)
qed

lemma smallnodes-sum[simp]:
   $\text{smallnodes } (A + B) = \text{smallnodes } A \cup \text{smallnodes } B$ 
  unfolding smallnodes-def
  by auto

lemma in-smallnodes:
  fixes  $A :: 'a \text{ acg}$ 
  assumes  $e: \text{has-edge } A \ x \ G \ y$ 
  shows  $\text{nodes } G \subseteq \text{smallnodes } A$ 
proof -
  have  $\text{fst } (\text{snd } (x, G, y)) \in \text{fst } ' \text{snd } ' \text{dest-graph } A$ 
    unfolding has-edge-def
    by (rule imageI) + (rule e[unfolded has-edge-def])
  then have  $G \in \text{edges } A$ 
    unfolding edges-def by simp
  thus ?thesis
    unfolding smallnodes-def
    by blast
qed

lemma finite-smallnodes:
  assumes  $fA: \text{finite-acg } A$ 
  shows finite (smallnodes A)
  unfolding smallnodes-def edges-def
proof
  from  $fA$ 
  show finite (nodes 'fst 'snd 'dest-graph A)
    unfolding finite-acg-def finite-graph-def
    by simp

```

```

fix  $M$  assume  $M \in \text{nodes } 'fst' 'snd' 'dest\text{-}graph\ A$ 
then obtain  $n\ G\ m$ 
  where  $M: M = \text{nodes } G$  and  $nGm: (n, G, m) \in \text{dest}\text{-}graph\ A$ 
  by auto

from  $fA$ 
have all-finite  $A$  unfolding finite-acg-def by simp
with  $nGm$  have finite-graph  $G$ 
  unfolding all-finite-def has-edge-def by auto
with finite-nodes
show finite  $M$ 
  unfolding finite-graph-def  $M$  .
qed

lemma nodes-tcl:
   $\text{nodes } (tcl\ A) = \text{nodes } A$ 
proof
  show  $\text{nodes } A \subseteq \text{nodes } (tcl\ A)$ 
    apply (rule nodes-subgraph)
    by (subst tcl-unfold-right) simp

  show  $\text{nodes } (tcl\ A) \subseteq \text{nodes } A$ 
  proof
    fix  $x$  assume  $x \in \text{nodes } (tcl\ A)$ 
    then obtain  $z\ G\ y$ 
      where  $z: z \in \text{dest}\text{-}graph\ (tcl\ A)$ 
      and  $dis: z = (x, G, y) \vee z = (y, G, x)$ 
      unfolding nodes-def
      by auto force+

    from  $dis$ 
    show  $x \in \text{nodes } A$ 
    proof
      assume  $z = (x, G, y)$ 
      with  $z$  have has-edge  $(tcl\ A)\ x\ G\ y$  unfolding has-edge-def by simp
      then obtain  $n$  where  $n > 0$  and  $An: \text{has-edge } (A \hat{\ } n)\ x\ G\ y$ 
        unfolding in-tcl by auto
      then obtain  $n'$  where  $n = \text{Suc } n'$  by (cases n, auto)
      hence  $A \hat{\ } n = A * A \hat{\ } n'$  by (simp add:power-Suc)
      with  $An$  obtain  $e\ k$ 
        where has-edge  $A\ x\ e\ k$  by (auto simp:in-grcomp)
      thus  $x \in \text{nodes } A$  unfolding has-edge-def nodes-def
        by force

    next
      assume  $z = (y, G, x)$ 
      with  $z$  have has-edge  $(tcl\ A)\ y\ G\ x$  unfolding has-edge-def by simp
      then obtain  $n$  where  $n > 0$  and  $An: \text{has-edge } (A \hat{\ } n)\ y\ G\ x$ 
        unfolding in-tcl by auto

```

```

    then obtain n' where n = Suc n' by (cases n, auto)
    hence A ^ n = A ^ n' * A by (simp add:power-Suc power-commutes)
    with An obtain e k
      where has-edge A k e x by (auto simp:in-grcomp)
    thus x ∈ nodes A unfolding has-edge-def nodes-def
      by force
  qed
qed
qed

```

```

lemma smallnodes-tcl:
  fixes A :: 'a acg
  shows smallnodes (tcl A) = smallnodes A
proof (intro equalityI subsetI)
  fix n assume n ∈ smallnodes (tcl A)
  then obtain x G y where edge: has-edge (tcl A) x G y
    and n ∈ nodes G
    unfolding smallnodes-def edges-def has-edge-def
    by auto

  from ⟨n ∈ nodes G⟩
  have n ∈ fst ' dest-graph G ∨ n ∈ snd ' snd ' dest-graph G
    (is ?A ∨ ?B)
    unfolding nodes-def by blast
  thus n ∈ smallnodes A
proof
  assume ?A
  then obtain m e where A: has-edge G n e m
    unfolding has-edge-def by auto

  have tcl A = A * star A
    unfolding tcl-def
    by (simp add: star-commute[of A A A, simplified])

  with edge
  have has-edge (A * star A) x G y by simp
  then obtain H H' z
    where AH: has-edge A x H z and G: G = H * H'
    by (auto simp:in-grcomp)
  from A
  obtain m' e' where has-edge H n e' m'
    by (auto simp:G in-grcomp)
  hence n ∈ nodes H unfolding nodes-def has-edge-def
    by force
  with in-smallnodes[OF AH] show n ∈ smallnodes A ..
next
  assume ?B
  then obtain m e where B: has-edge G m e n
    unfolding has-edge-def by auto

```

```

with edge
have has-edge (star A * A) x G y by (simp add:tcl-def)
then obtain H H' z
  where AH': has-edge A z H' y and G: G = H * H'
  by (auto simp:in-grcomp)
from B
obtain m' e' where has-edge H' m' e' n
  by (auto simp:G in-grcomp)
hence n ∈ nodes H' unfolding nodes-def has-edge-def
  by force
with in-smallnodes[OF AH'] show n ∈ smallnodes A ..
qed
next
fix x assume x ∈ smallnodes A
then show x ∈ smallnodes (tcl A)
  by (subst tcl-unfold-right) simp
qed

lemma finite-nodegraphs:
  assumes F: finite F
  shows finite { G::'a scg. nodes G ⊆ F } (is finite ?P)
proof (rule finite-subset)
show ?P ⊆ Graph ' (Pow (F × UNIV × F)) (is ?P ⊆ ?Q)
proof
fix x assume xP: x ∈ ?P
obtain S where x[simp]: x = Graph S
  by (cases x) auto
from xP
show x ∈ ?Q
  apply (simp add:nodes-def)
  apply (rule imageI)
  apply (rule PowI)
  apply force
done
qed
show finite ?Q
  by (auto intro:finite-imageI finite-cartesian-product F finite)
qed

lemma finite-graphI:
  fixes A :: 'a acg
  assumes fin: finite (nodes A) finite (smallnodes A)
  shows finite-graph A
proof -
obtain S where A[simp]: A = Graph S
  by (cases A) auto

have finite S

```

```

proof (rule finite-subset)
  show  $S \subseteq \text{nodes } A \times \{ G :: 'a \text{ scg. nodes } G \subseteq \text{smallnodes } A \} \times \text{nodes } A$ 
    (is  $S \subseteq ?T$ )
proof
  fix  $x$  assume  $xS$ :  $x \in S$ 
  obtain  $a \ b \ c$  where  $x[\text{simp}]$ :  $x = (a, b, c)$ 
    by (cases  $x$ ) auto

  then have  $\text{edg: has-edge } A \ a \ b \ c$ 
    unfolding  $\text{has-edge-def}$  using  $xS$ 
    by  $\text{simp}$ 

  hence  $a \in \text{nodes } A \ c \in \text{nodes } A$ 
    unfolding  $\text{nodes-def has-edge-def}$  by  $\text{force+}$ 
  moreover
  from  $\text{edg}$  have  $\text{nodes } b \subseteq \text{smallnodes } A$  by (rule in-smallnodes)
  hence  $b \in \{ G :: 'a \text{ scg. nodes } G \subseteq \text{smallnodes } A \}$  by  $\text{simp}$ 
  ultimately show  $x \in ?T$  by  $\text{simp}$ 
qed

show  $\text{finite } ?T$ 
  by (intro finite-cartesian-product fin finite-nodegraphs)
qed
thus  $?thesis$ 
  unfolding  $\text{finite-graph-def}$  by  $\text{simp}$ 
qed

lemma  $\text{smallnodes-allfinite}$ :
  fixes  $A :: 'a \text{ acg}$ 
  assumes  $\text{fin: finite (smallnodes } A)$ 
  shows  $\text{all-finite } A$ 
  unfolding  $\text{all-finite-def}$ 
proof (intro allI impI)
  fix  $n \ H \ m$  assume  $\text{has-edge } A \ n \ H \ m$ 
  then have  $\text{nodes } H \subseteq \text{smallnodes } A$ 
    by (rule in-smallnodes)
  then have  $\text{finite (nodes } H)$ 
    by (rule finite-subset) (rule fin)
  thus  $\text{finite-graph } H$  by (rule scg-finite)
qed

lemma  $\text{finite-tcl}$ :
  fixes  $A :: 'a \text{ acg}$ 
  shows  $\text{finite-acg (tcl } A) \longleftrightarrow \text{finite-acg } A$ 
proof
  assume  $f$ :  $\text{finite-acg } A$ 
  from  $f$  have  $g$ :  $\text{finite-graph } A$  and  $\text{all-finite } A$ 
    unfolding  $\text{finite-acg-def}$  by auto

```

```

from  $g$  have  $\text{finite } (\text{nodes } A)$  by (rule  $\text{finite-nodes}$ )
then have  $\text{finite } (\text{nodes } (\text{tcl } A))$  unfolding  $\text{nodes-tcl}$  .
moreover
from  $f$  have  $\text{finite } (\text{smallnodes } A)$  by (rule  $\text{finite-smallnodes}$ )
then have  $\text{fs: finite } (\text{smallnodes } (\text{tcl } A))$  unfolding  $\text{smallnodes-tcl}$  .
ultimately
have  $\text{finite-graph } (\text{tcl } A)$  by (rule  $\text{finite-graphI}$ )

moreover from  $\text{fs}$  have  $\text{all-finite } (\text{tcl } A)$ 
  by (rule  $\text{smallnodes-allfinite}$ )
ultimately show  $\text{finite-acg } (\text{tcl } A)$  unfolding  $\text{finite-acg-def}$  ..
next
  assume  $a: \text{finite-acg } (\text{tcl } A)$ 
  have  $A \leq \text{tcl } A$  by (rule  $\text{less-tcl}$ )
  thus  $\text{finite-acg } A$  using  $a$ 
    by (rule  $\text{finite-acg-subset}$ )
qed

lemma  $\text{finite-acg-empty: finite-acg } (\text{Graph } \{\})$ 
  unfolding  $\text{finite-acg-def finite-graph-def all-finite-def}$ 
   $\text{has-edge-def}$ 
  by  $\text{simp}$ 

lemma  $\text{finite-acg-ins:}$ 
  assumes  $fA: \text{finite-acg } (\text{Graph } A)$ 
  assumes  $fG: \text{finite } G$ 
  shows  $\text{finite-acg } (\text{Graph } (\text{insert } (a, \text{Graph } G, b) A))$ 
  using  $fA fG$ 
  unfolding  $\text{finite-acg-def finite-graph-def all-finite-def}$ 
   $\text{has-edge-def}$ 
  by  $\text{auto}$ 

lemmas  $\text{finite-acg-simps} = \text{finite-acg-empty finite-acg-ins finite-graph-def}$ 

```

5.3 Contraction and more

abbreviation

$\text{pdesc } P == (\text{fst } P, \text{prod } P, \text{end-node } P)$

lemma pdesc-acgplus:

```

assumes  $\text{has-ipath } \mathcal{A} p$ 
and  $i < j$ 
shows  $\text{has-edge } (\text{tcl } \mathcal{A}) (\text{fst } (p\langle i,j \rangle)) (\text{prod } (p\langle i,j \rangle)) (\text{end-node } (p\langle i,j \rangle))$ 
unfolding  $\text{plus-paths}$ 
apply (rule  $\text{exI}$ )
apply (insert  $\text{prems}$ )
by ( $\text{auto intro: sub-path-is-path[of } \mathcal{A} p i j] \text{ simp: sub-path-def}$ )

```

```

lemma combine-fthreads:
  assumes range:  $i < j \leq k$ 
  shows
     $\text{has-fth } p \ i \ k \ m \ r =$ 
     $(\exists n. \text{has-fth } p \ i \ j \ m \ n \wedge \text{has-fth } p \ j \ k \ n \ r) \text{ (is } ?L = ?R)$ 
proof (intro iffI)
  assume  $?L$ 
  then obtain  $\vartheta$ 
    where  $\text{is-fthread } \vartheta \ p \ i \ k$ 
    and  $[simp]: \vartheta \ i = m \ \vartheta \ k = r$ 
    by (auto simp:has-fth-def)

  with range
  have  $\text{is-fthread } \vartheta \ p \ i \ j$  and  $\text{is-fthread } \vartheta \ p \ j \ k$ 
    by (auto simp:is-fthread-def)
  hence  $\text{has-fth } p \ i \ j \ m \ (\vartheta \ j)$  and  $\text{has-fth } p \ j \ k \ (\vartheta \ j) \ r$ 
    by (auto simp:has-fth-def)
  thus  $?R$  by auto
next
  assume  $?R$ 
  then obtain  $n \ \vartheta 1 \ \vartheta 2$ 
    where  $\text{ths: is-fthread } \vartheta 1 \ p \ i \ j \ \text{is-fthread } \vartheta 2 \ p \ j \ k$ 
    and  $[simp]: \vartheta 1 \ i = m \ \vartheta 1 \ j = n \ \vartheta 2 \ j = n \ \vartheta 2 \ k = r$ 
    by (auto simp:has-fth-def)

  let  $? \vartheta = (\lambda i. \text{if } i < j \text{ then } \vartheta 1 \ i \text{ else } \vartheta 2 \ i)$ 
  have  $\text{is-fthread } ? \vartheta \ p \ i \ k$ 
    unfolding is-fthread-def
  proof
    fix  $l$  assume range:  $l \in \{i..<k\}$ 

    show  $\text{eqLat } p \ ? \vartheta \ l$ 
    proof (cases rule:three-cases)
      assume  $\text{Suc } l < j$ 
      with ths range show  $?thesis$ 
        unfolding is-fthread-def Ball-def
        by simp
      next
        assume  $\text{Suc } l = j$ 
        hence  $l < j \ \vartheta 2 \ (\text{Suc } l) = \vartheta 1 \ (\text{Suc } l)$  by auto
        with ths range show  $?thesis$ 
          unfolding is-fthread-def Ball-def
          by simp
      next
        assume  $j \leq l$ 
        with ths range show  $?thesis$ 
          unfolding is-fthread-def Ball-def
          by simp
    qed
  qed

```



```

    qed arith
  qed
  moreover
  have "? $\vartheta$  i = m ? $\vartheta$  k = r" using range by auto
  ultimately show has-fth p i k m r
    by (auto simp: has-fth-def)
  qed

```

```

lemma desc-is-fthread:
  is-desc-fthread  $\vartheta$  p i k  $\implies$  is-fthread  $\vartheta$  p i k
  unfolding is-desc-fthread-def
  by simp

```

```

lemma combine-dfthreads:
  assumes range:  $i < j \leq k$ 
  shows
    has-desc-fth p i k m r =
    ( $\exists n. (has-desc-fth p i j m n \wedge has-fth p j k n r)$ 
     $\vee (has-fth p i j m n \wedge has-desc-fth p j k n r)$ ) (is ?L = ?R)

```

```

proof
  assume ?L
  then obtain  $\vartheta$ 
    where desc: is-desc-fthread  $\vartheta$  p i k
    and [simp]:  $\vartheta$  i = m  $\vartheta$  k = r
    by (auto simp: has-desc-fth-def)

  hence is-fthread  $\vartheta$  p i k
    by (simp add: desc-is-fthread)
  with range have fths: is-fthread  $\vartheta$  p i j is-fthread  $\vartheta$  p j k
    unfolding is-fthread-def
    by auto
  hence hfths: has-fth p i j m ( $\vartheta$  j) has-fth p j k ( $\vartheta$  j) r
    by (auto simp: has-fth-def)

```

```

from desc obtain l
  where  $i \leq l < k$ 
  and descat p  $\vartheta$  l
  by (auto simp: is-desc-fthread-def)

```

```

with fths
have is-desc-fthread  $\vartheta$  p i j  $\vee$  is-desc-fthread  $\vartheta$  p j k
  unfolding is-desc-fthread-def
  by (cases l < j) auto
hence has-desc-fth p i j m ( $\vartheta$  j)  $\vee$  has-desc-fth p j k ( $\vartheta$  j) r
  by (auto simp: has-desc-fth-def)
with hfths show ?R
  by auto

```

```

next
  assume ?R
  then obtain n  $\vartheta 1$   $\vartheta 2$ 
    where (is-desc-fthread  $\vartheta 1$   $p$   $i$   $j$   $\wedge$  is-fthread  $\vartheta 2$   $p$   $j$   $k$ )
       $\vee$  (is-fthread  $\vartheta 1$   $p$   $i$   $j$   $\wedge$  is-desc-fthread  $\vartheta 2$   $p$   $j$   $k$ )
    and [simp]:  $\vartheta 1$   $i = m$   $\vartheta 1$   $j = n$   $\vartheta 2$   $j = n$   $\vartheta 2$   $k = r$ 
    by (auto simp:has-fth-def has-desc-fth-def)

  hence ths2: is-fthread  $\vartheta 1$   $p$   $i$   $j$  is-fthread  $\vartheta 2$   $p$   $j$   $k$ 
    and dths: is-desc-fthread  $\vartheta 1$   $p$   $i$   $j$   $\vee$  is-desc-fthread  $\vartheta 2$   $p$   $j$   $k$ 
    by (auto simp:desc-is-fthread)

  let ? $\vartheta$  = ( $\lambda i.$  if  $i < j$  then  $\vartheta 1$   $i$  else  $\vartheta 2$   $i$ )
  have is-fthread ? $\vartheta$   $p$   $i$   $k$ 
    unfolding is-fthread-def
  proof
    fix  $l$  assume range:  $l \in \{i..<k\}$ 

    show eqlat  $p$  ? $\vartheta$   $l$ 
    proof (cases rule:three-cases)
      assume Suc  $l < j$ 
      with ths2 range show ?thesis
        unfolding is-fthread-def Ball-def
        by simp
    next
      assume Suc  $l = j$ 
      hence  $l < j$   $\vartheta 2$  (Suc  $l$ ) =  $\vartheta 1$  (Suc  $l$ ) by auto
      with ths2 range show ?thesis
        unfolding is-fthread-def Ball-def
        by simp
    next
      assume  $j \leq l$ 
      with ths2 range show ?thesis
        unfolding is-fthread-def Ball-def
        by simp
    qed arith
  qed
  moreover
  from dths
  have  $\exists l. i \leq l \wedge l < k \wedge$  descat  $p$  ? $\vartheta$   $l$ 
  proof
    assume is-desc-fthread  $\vartheta 1$   $p$   $i$   $j$ 

    then obtain  $l$  where range:  $i \leq l$   $l < j$  and descat  $p$   $\vartheta 1$   $l$ 
      unfolding is-desc-fthread-def Bex-def by auto
    hence descat  $p$  ? $\vartheta$   $l$ 
      by (cases Suc  $l = j$ , auto)
    with  $\langle j \leq k \rangle$  and range show ?thesis
      by (rule-tac  $x=l$  in exI, auto)
  end

```

next
 assume *is-desc-fthread* $\vartheta 2\ p\ j\ k$
 then obtain l where range: $j \leq l\ l < k$ and *descat* $p\ \vartheta 2\ l$
 unfolding *is-desc-fthread-def Bex-def* by *auto*
 with $\langle i < j \rangle$ have *descat* $p\ \vartheta\ l\ i \leq l$
 by *auto*
 with range show *?thesis*
 by (*rule-tac* $x=l$ in *exI*, *auto*)
qed
 ultimately have *is-desc-fthread* $\vartheta\ p\ i\ k$
 by (*simp* add: *is-desc-fthread-def Bex-def*)

moreover
 have $\vartheta\ i = m\ \vartheta\ k = r$ using range by *auto*

 ultimately show *has-desc-fth* $p\ i\ k\ m\ r$
 by (*auto simp:has-desc-fth-def*)
qed

lemma *fth-single*:
has-fth $p\ i\ (Suc\ i)\ m\ n = eql\ (snd\ (p\ i))\ m\ n$ (is $?L = ?R$)
proof
 assume $?L$ thus $?R$
 unfolding *is-fthread-def Ball-def has-fth-def*
 by *auto*
next
 let $\vartheta = \lambda k. \text{if } k = i \text{ then } m \text{ else } n$
 assume edge: $?R$
 hence *is-fthread* $\vartheta\ p\ i\ (Suc\ i) \wedge \vartheta\ i = m \wedge \vartheta\ (Suc\ i) = n$
 unfolding *is-fthread-def Ball-def*
 by *auto*

 thus $?L$
 unfolding *has-fth-def*
 by *auto*
qed

lemma *desc-fth-single*:
has-desc-fth $p\ i\ (Suc\ i)\ m\ n =$
dsc $(snd\ (p\ i))\ m\ n$ (is $?L = ?R$)
proof
 assume $?L$ thus $?R$
 unfolding *is-desc-fthread-def has-desc-fth-def is-fthread-def Bex-def*
 by (*elim* *exE conjE*) (*case-tac* $k = i$, *auto*)
next
 let $\vartheta = \lambda k. \text{if } k = i \text{ then } m \text{ else } n$

assume *edge*: ?*R*
hence *is-desc-fthread* ? ϑ *p i* (*Suc i*) \wedge ? ϑ *i* = *m* \wedge ? ϑ (*Suc i*) = *n*
unfolding *is-desc-fthread-def is-fthread-def Ball-def Bex-def*
by *auto*
thus ?*L*
unfolding *has-desc-fth-def*
by *auto*
qed

lemma *mk-eql*: ($G \vdash m \rightsquigarrow^e n$) \implies *eql* *G m n*
by (*cases e, auto*)

lemma *eql-scgcomp*:
eql (*G * H*) *m r* =
 $(\exists n. \text{eql } G \text{ } m \text{ } n \wedge \text{eql } H \text{ } n \text{ } r)$ (**is** ?*L* = ?*R*)
proof
show ?*L* \implies ?*R*
by (*auto simp:in-grcomp intro!:mk-eql*)

assume ?*R*
then obtain *n* **where** *l*: *eql G m n* **and** *r*:*eql H n r* **by** *auto*
thus ?*L*
by (*cases dsc G m n*) (*auto simp:in-grcomp mult-sedge-def*)
qed

lemma *desc-scgcomp*:
dsc (*G * H*) *m r* =
 $(\exists n. (dsc \text{ } G \text{ } m \text{ } n \wedge \text{eql } H \text{ } n \text{ } r) \vee (eqp \text{ } G \text{ } m \text{ } n \wedge dsc \text{ } H \text{ } n \text{ } r))$ (**is** ?*L* = ?*R*)
proof
show ?*R* \implies ?*L* **by** (*auto simp:in-grcomp mult-sedge-def*)

assume ?*L*
thus ?*R*
by (*auto simp:in-grcomp mult-sedge-def*)
(*case-tac e, auto, case-tac e', auto*)
qed

lemma *has-fth-unfold*:
assumes $i < j$
shows *has-fth* *p i j m n* =
 $(\exists r. \text{has-fth } p \text{ } i \text{ } (\text{Suc } i) \text{ } m \text{ } r \wedge \text{has-fth } p \text{ } (\text{Suc } i) \text{ } j \text{ } r \text{ } n)$
by (*rule combine-fthreads*) (*insert* $\langle i < j \rangle$, *auto*)

lemma *has-dfth-unfold*:
assumes *range*: $i < j$
shows
has-desc-fth *p i j m r* =
 $(\exists n. (\text{has-desc-fth } p \text{ } i \text{ } (\text{Suc } i) \text{ } m \text{ } n \wedge \text{has-fth } p \text{ } (\text{Suc } i) \text{ } j \text{ } n \text{ } r))$

$\vee (has_fth\ p\ i\ (Suc\ i)\ m\ n \wedge has_desc_fth\ p\ (Suc\ i)\ j\ n\ r))$
by (rule combine-dfthreads) (insert $\langle i < j \rangle$, auto)

lemma Lemma7a:

$i \leq j \implies has_fth\ p\ i\ j\ m\ n = eql\ (prod\ (p\langle i, j \rangle))\ m\ n$
proof (induct i arbitrary: m rule:inc-induct)
case base **show** ?case
unfolding has-fth-def is-fthread-def sub-path-def
by (auto simp:in-grunit one-sedge-def)
next
case (step i)
note IH = $\langle \bigwedge m. has_fth\ p\ (Suc\ i)\ j\ m\ n =$
 $eql\ (prod\ (p\langle Suc\ i, j \rangle))\ m\ n \rangle$
have has-fth p i j m n
 $= (\exists r. has_fth\ p\ i\ (Suc\ i)\ m\ r \wedge has_fth\ p\ (Suc\ i)\ j\ r\ n)$
by (rule has-fth-unfold[OF $\langle i < j \rangle$])
also have ... = $(\exists r. has_fth\ p\ i\ (Suc\ i)\ m\ r$
 $\wedge eql\ (prod\ (p\langle Suc\ i, j \rangle))\ r\ n)$
by (simp only:IH)
also have ... = $(\exists r. eql\ (snd\ (p\ i))\ m\ r$
 $\wedge eql\ (prod\ (p\langle Suc\ i, j \rangle))\ r\ n)$
by (simp only:fth-single)
also have ... = $eql\ (snd\ (p\ i) * prod\ (p\langle Suc\ i, j \rangle))\ m\ n$
by (simp only:eql-scgcomp)
also have ... = $eql\ (prod\ (p\langle i, j \rangle))\ m\ n$
by (simp only:prod-unfold[OF $\langle i < j \rangle$])
finally show ?case .
qed

lemma Lemma7b:

assumes $i \leq j$
shows
 $has_desc_fth\ p\ i\ j\ m\ n =$
 $dsc\ (prod\ (p\langle i, j \rangle))\ m\ n$
using prems
proof (induct i arbitrary: m rule:inc-induct)
case base **show** ?case
unfolding has-desc-fth-def is-desc-fthread-def sub-path-def
by (auto simp:in-grunit one-sedge-def)
next
case (step i)
thus ?case
by (simp only:prod-unfold desc-scgcomp desc-fth-single
has-dfth-unfold fth-single Lemma7a) auto
qed

lemma *descat-contract*:
assumes $[simp]$: *increasing s*
shows
 $descat\ (contract\ s\ p)\ \vartheta\ i =$
 $has-desc-fth\ p\ (s\ i)\ (s\ (Suc\ i))\ (\vartheta\ i)\ (\vartheta\ (Suc\ i))$
by (*simp add:Lemma7b increasing-weak contract-def*)

lemma *eqlat-contract*:
assumes $[simp]$: *increasing s*
shows
 $eqlat\ (contract\ s\ p)\ \vartheta\ i =$
 $has-fth\ p\ (s\ i)\ (s\ (Suc\ i))\ (\vartheta\ i)\ (\vartheta\ (Suc\ i))$
by (*auto simp:Lemma7a increasing-weak contract-def*)

5.3.1 Connecting threads

definition
 $connect\ s\ \vartheta s = (\lambda k. \vartheta s\ (section-of\ s\ k)\ k)$

lemma *next-in-range*:
assumes $[simp]$: *increasing s*
assumes a : $k \in section\ s\ i$
shows $(Suc\ k \in section\ s\ i) \vee (Suc\ k \in section\ s\ (Suc\ i))$
proof –
from a **have** $k < s\ (Suc\ i)$ **by** *simp*

hence $Suc\ k < s\ (Suc\ i) \vee Suc\ k = s\ (Suc\ i)$ **by** *arith*
thus *?thesis*
proof
assume $Suc\ k < s\ (Suc\ i)$
with a **have** $Suc\ k \in section\ s\ i$ **by** *simp*
thus *?thesis* ..
next
assume eq : $Suc\ k = s\ (Suc\ i)$
with *increasing-strict* **have** $Suc\ k < s\ (Suc\ (Suc\ i))$ **by** *simp*
with eq **have** $Suc\ k \in section\ s\ (Suc\ i)$ **by** *simp*
thus *?thesis* ..
qed
qed

lemma *connect-threads*:
assumes $[simp]$: *increasing s*
assumes *connected*: $\vartheta s\ i\ (s\ (Suc\ i)) = \vartheta s\ (Suc\ i)\ (s\ (Suc\ i))$
assumes *fth*: *is-fthread* $(\vartheta s\ i)\ p\ (s\ i)\ (s\ (Suc\ i))$

shows

```

is-fthread (connect s ∅s) p (s i) (s (Suc i))
unfolding is-fthread-def
proof
  fix k assume krng: k ∈ section s i

  with fth have eqlat: eqlat p (∅s i) k
  unfolding is-fthread-def by simp

  from krng and next-in-range
  have (Suc k ∈ section s i) ∨ (Suc k ∈ section s (Suc i))
  by simp
  thus eqlat p (connect s ∅s) k
  proof
    assume Suc k ∈ section s i
    with krng eqlat show ?thesis
    unfolding connect-def
    by (simp only: section-of-known ⟨increasing s⟩)
  next
    assume skrng: Suc k ∈ section s (Suc i)
    with krng have Suc k = s (Suc i) by auto

    with krng skrng eqlat show ?thesis
    unfolding connect-def
    by (simp only: section-of-known connected[symmetric] ⟨increasing s⟩)
  qed
qed

```

```

lemma connect-dthreads:
  assumes inc[simp]: increasing s
  assumes connected: ∅s i (s (Suc i)) = ∅s (Suc i) (s (Suc i))
  assumes fth: is-desc-fthread (∅s i) p (s i) (s (Suc i))

  shows
    is-desc-fthread (connect s ∅s) p (s i) (s (Suc i))
  unfolding is-desc-fthread-def
  proof
    show is-fthread (connect s ∅s) p (s i) (s (Suc i))
    apply (rule connect-threads)
    apply (insert fth)
    by (auto simp: connected is-desc-fthread-def)

  from fth
  obtain k where dsc: dscat p (∅s i) k and krng: k ∈ section s i
  unfolding is-desc-fthread-def by blast

  from krng and next-in-range
  have (Suc k ∈ section s i) ∨ (Suc k ∈ section s (Suc i))
  by simp

```

```

hence descat p (connect s ∅ s) k
proof
  assume Suc k ∈ section s i
  with krng dsc show ?thesis unfolding connect-def
    by (simp only: section-of-known inc)
next
  assume skrng: Suc k ∈ section s (Suc i)
  with krng have Suc k = s (Suc i) by auto

  with krng skrng dsc show ?thesis unfolding connect-def
    by (simp only: section-of-known connected[symmetric] inc)
qed
with krng show ∃ k ∈ section s i. descat p (connect s ∅ s) k ..
qed

```

```

lemma mk-inf-thread:
  assumes [simp]: increasing s
  assumes fths:  $\bigwedge i. i > n \implies \text{is-fthread } \varnothing p (s\ i) (s\ (\text{Suc } i))$ 
  shows is-thread (s (Suc n)) ∅ p
  unfolding is-thread-def
proof (intro allI impI)
  fix j assume st: s (Suc n) ≤ j

  let ?k = section-of s j
  from in-section-of st
  have rs: j ∈ section s ?k by simp

  with st have s (Suc n) < s (Suc ?k) by simp
  with increasing-bij have n < ?k by simp
  with rs and fths[of ?k]
  show eqlat p ∅ j by (simp add: is-fthread-def)
qed

```

```

lemma mk-inf-desc-thread:
  assumes [simp]: increasing s
  assumes fths:  $\bigwedge i. i > n \implies \text{is-fthread } \varnothing p (s\ i) (s\ (\text{Suc } i))$ 
  assumes fdths:  $\exists_{\infty} i. \text{is-desc-fthread } \varnothing p (s\ i) (s\ (\text{Suc } i))$ 
  shows is-desc-thread ∅ p
  unfolding is-desc-thread-def
proof (intro exI conjI)

  from mk-inf-thread[of s n ∅ p] fths
  show  $\forall i \geq s\ (\text{Suc } n). \text{eqlat } p\ \varnothing\ i$ 
    by (fold is-thread-def) simp

  show  $\exists_{\infty} l. \text{desc}_{at}\ p\ \varnothing\ l$ 
    unfolding INF-nat
  proof

```



```

fix i

let ?k = section-of s i
from fdths obtain j
  where ?k < j is-desc-fthread  $\vartheta$  p (s j) (s (Suc j))
  unfolding INF-nat by auto
then obtain l where s j  $\leq$  l and desc: descat p  $\vartheta$  l
  unfolding is-desc-fthread-def
  by auto

have i < s (Suc ?k) by (rule section-of2) simp
also have ...  $\leq$  s j
  by (rule increasing-weak [OF ‹increasing s›]) (insert ‹?k < j›, arith)
also note ‹...  $\leq$  l›
finally have i < l .
with desc
show  $\exists l. i < l \wedge \text{descat } p \ \vartheta \ l$  by blast
qed
qed

```

```

lemma desc-ex-choice:
  assumes A:  $((\exists n. \forall i \geq n. \exists x. P \ x \ i) \wedge (\exists_{\infty} i. \exists x. Q \ x \ i))$ 
  and imp:  $\bigwedge x \ i. Q \ x \ i \implies P \ x \ i$ 
  shows  $\exists xs. ((\exists n. \forall i \geq n. P \ (xs \ i) \ i) \wedge (\exists_{\infty} i. Q \ (xs \ i) \ i))$ 
  (is  $\exists xs. ?Ps \ xs \wedge ?Qs \ xs$ )
proof
  let ?w =  $\lambda i. (if (\exists x. Q \ x \ i) \text{ then } (SOME \ x. Q \ x \ i)$ 
     $\text{ else } (SOME \ x. P \ x \ i))$ 

```

```

from A
obtain n where P:  $\bigwedge i. n \leq i \implies \exists x. P \ x \ i$ 
  by auto
{
  fix i::'a assume n  $\leq$  i

  have P (?w i) i
  proof (cases  $\exists x. Q \ x \ i$ )
    case True
      hence Q (?w i) i by (auto intro:someI)
      with imp show P (?w i) i .
    next
      case False
        with P[OF ‹n  $\leq$  i›] show P (?w i) i
          by (auto intro:someI)
  qed
}

```

```

hence ?Ps ?w by (rule-tac x=n in exI) auto

```

```

moreover
from  $A$  have  $\exists_{\infty} i. (\exists x. Q\ x\ i) \dots$ 
hence  $?Qs\ ?w$  by (rule INF-mono) (auto intro:someI)
ultimately
show  $?Ps\ ?w \wedge ?Qs\ ?w \dots$ 
qed

```

```

lemma dthreads-join:
  assumes [simp]: increasing s
  assumes dthread: is-desc-thread  $\vartheta$  (contract s p)
  shows  $\exists \vartheta s. \text{desc } (\lambda i. \text{is-fthread } (\vartheta s\ i)\ p\ (s\ i)\ (s\ (\text{Suc } i)))$ 
     $\wedge \vartheta s\ i\ (s\ i) = \vartheta\ i$ 
     $\wedge \vartheta s\ i\ (s\ (\text{Suc } i)) = \vartheta\ (\text{Suc } i)$ 
     $(\lambda i. \text{is-desc-fthread } (\vartheta s\ i)\ p\ (s\ i)\ (s\ (\text{Suc } i)))$ 
     $\wedge \vartheta s\ i\ (s\ i) = \vartheta\ i$ 
     $\wedge \vartheta s\ i\ (s\ (\text{Suc } i)) = \vartheta\ (\text{Suc } i)$ 
  apply (rule desc-ex-choice)
  apply (insert dthread)
  apply (simp only:is-desc-thread-def)
  apply (simp add:eqlat-contract)
  apply (simp add:descat-contract)
  apply (simp only:has-fth-def has-desc-fth-def)
  by (auto simp:is-desc-fthread-def)

```

```

lemma INF-drop-prefix:
   $(\exists_{\infty} i::\text{nat}. i > n \wedge P\ i) = (\exists_{\infty} i. P\ i)$ 
  apply (auto simp:INF-nat)
  apply (drule-tac x = max m n in spec)
  apply (elim exE conjE)
  apply (rule-tac x = na in exI)
  by auto

```

```

lemma contract-keeps-threads:
  assumes inc[simp]: increasing s
  shows  $(\exists \vartheta. \text{is-desc-thread } \vartheta\ p)$ 
     $\longleftrightarrow (\exists \vartheta. \text{is-desc-thread } \vartheta\ (\text{contract } s\ p))$ 
    (is ?A  $\longleftrightarrow$  ?B)
proof
  assume ?A
  then obtain  $\vartheta\ n$ 
    where fr:  $\forall i \geq n. \text{eqlat } p\ \vartheta\ i$ 
    and ds:  $\exists_{\infty} i. \text{descat } p\ \vartheta\ i$ 

```

unfolding *is-desc-thread-def*
by *auto*

let $?c\vartheta = \lambda i. \vartheta (s\ i)$

have *is-desc-thread* $?c\vartheta$ (*contract* $s\ p$)
unfolding *is-desc-thread-def*
proof (*intro exI conjI*)

show $\forall i \geq n. \text{eqlat } (\text{contract } s\ p) \ ?c\vartheta\ i$
proof (*intro allI impI*)
fix i **assume** $n \leq i$
also have $i \leq s\ i$
using *increasing-inc* **by** *auto*
finally have $n \leq s\ i$.

with *fr* **have** *is-fthread* $\vartheta\ p\ (s\ i)\ (s\ (\text{Suc } i))$
unfolding *is-fthread-def* **by** *auto*
hence *has-fth* $p\ (s\ i)\ (s\ (\text{Suc } i))\ (\vartheta\ (s\ i))\ (\vartheta\ (s\ (\text{Suc } i)))$
unfolding *has-fth-def* **by** *auto*
with *less-imp-le*[*OF increasing-strict*]
have *eql* (*prod* ($p\ (s\ i, s\ (\text{Suc } i)))$) ($\vartheta\ (s\ i)$) ($\vartheta\ (s\ (\text{Suc } i))$)
by (*simp add:Lemma7a*)
thus *eqlat* (*contract* $s\ p$) $?c\vartheta\ i$ **unfolding** *contract-def*
by *auto*
qed

show $\exists_{\infty} i. \text{descat } (\text{contract } s\ p) \ ?c\vartheta\ i$
unfolding *INF-nat*
proof
fix i

let $?K = \text{section-of } s\ (\max\ (s\ (\text{Suc } i))\ n)$
from $\langle \exists_{\infty} i. \text{descat } p\ \vartheta\ i \rangle$ **obtain** j
where $s\ (\text{Suc } ?K) < j \text{ desc } p\ \vartheta\ j$
unfolding *INF-nat* **by** *blast*

let $?L = \text{section-of } s\ j$
{
fix x **assume** $r: x \in \text{section } s\ ?L$

have $e1: \max\ (s\ (\text{Suc } i))\ n < s\ (\text{Suc } ?K)$ **by** (*rule section-of2*) *simp*
note $s\ (\text{Suc } ?K) < j$
also have $j < s\ (\text{Suc } ?L)$
by (*rule section-of2*) *simp*
finally have $\text{Suc } ?K \leq ?L$
by (*simp add:increasing-bij*)
with *increasing-weak* **have** $s\ (\text{Suc } ?K) \leq s\ ?L$ **by** *simp*
with $e1\ r$ **have** $\max\ (s\ (\text{Suc } i))\ n < x$ **by** *simp*

```

    hence (s (Suc i)) < x n < x by auto
  }
note range-est = this

have is-desc-fthread  $\vartheta$  p (s ?L) (s (Suc ?L))
  unfolding is-desc-fthread-def is-fthread-def
proof
  show  $\forall m \in \text{section } s \text{ ?L. } \text{eqlat } p \ \vartheta \ m$ 
  proof
    fix m assume  $m \in \text{section } s \text{ ?L}$ 
    with range-est(2) have  $n < m$  .
    with fr show  $\text{eqlat } p \ \vartheta \ m$  by simp
  qed

from in-section-of inc less-imp-le[OF  $\langle s \text{ (Suc ?K) } < j \rangle$ ]
have  $j \in \text{section } s \text{ ?L}$  .

with  $\langle \text{descat } p \ \vartheta \ j \rangle$ 
show  $\exists m \in \text{section } s \text{ ?L. } \text{descat } p \ \vartheta \ m \ ..$ 
qed

with less-imp-le[OF increasing-strict]
have a: descat (contract s p) ?c $\vartheta$  ?L
  unfolding contract-def Lemma7b[symmetric]
  by (auto simp:Lemma7b[symmetric] has-desc-fth-def)

have  $i < ?L$ 
proof (rule classical)
  assume  $\neg i < ?L$ 
  hence  $s \text{ ?L} < s \text{ (Suc } i)$ 
    by (simp add:increasing-bij)
  also have  $\dots < s \text{ ?L}$ 
    by (rule range-est(1)) (simp add:increasing-strict)
  finally show ?thesis .
qed
with a show  $\exists l. i < l \wedge \text{descat } (contract \ s \ p) \ ?c\vartheta \ l$ 
  by blast
qed
qed
with exI show ?B .
next
assume ?B
then obtain  $\vartheta$ 
  where dthread: is-desc-thread  $\vartheta$  (contract s p) ..

with dthreads-join inc
obtain  $\vartheta s$  where ths-spec:
  desc ( $\lambda i. \text{is-fthread } (\vartheta s \ i) \ p \ (s \ i) \ (s \text{ (Suc } i))$ )

```

```

       $\wedge \vartheta s\ i\ (s\ i) = \vartheta\ i$ 
       $\wedge \vartheta s\ i\ (s\ (Suc\ i)) = \vartheta\ (Suc\ i)$ 
       $(\lambda i. is\_desc\_fthread\ (\vartheta s\ i)\ p\ (s\ i)\ (s\ (Suc\ i)))$ 
       $\wedge \vartheta s\ i\ (s\ i) = \vartheta\ i$ 
       $\wedge \vartheta s\ i\ (s\ (Suc\ i)) = \vartheta\ (Suc\ i)$ 
      (is desc ?alw ?inf)
    by blast

  then obtain n where fr:  $\forall i \geq n. ?alw\ i$  by blast
  hence connected:  $\bigwedge i. n < i \implies \vartheta s\ i\ (s\ (Suc\ i)) = \vartheta s\ (Suc\ i)\ (s\ (Suc\ i))$ 
    by auto

  let ?jv = connect s v s

  from fr ths-spec have ths-spec2:
     $\bigwedge i. i > n \implies is\_fthread\ (\vartheta s\ i)\ p\ (s\ i)\ (s\ (Suc\ i))$ 
     $\exists_{\infty} i. is\_desc\_fthread\ (\vartheta s\ i)\ p\ (s\ i)\ (s\ (Suc\ i))$ 
    by (auto intro: INF-mono)

  have p1:  $\bigwedge i. i > n \implies is\_fthread\ ?jv\ p\ (s\ i)\ (s\ (Suc\ i))$ 
    by (rule connect-threads) (auto simp: connected ths-spec2)

  from ths-spec2(2)
  have  $\exists_{\infty} i. n < i \wedge is\_desc\_fthread\ (\vartheta s\ i)\ p\ (s\ i)\ (s\ (Suc\ i))$ 
    unfolding INF-drop-prefix .

  hence p2:  $\exists_{\infty} i. is\_desc\_fthread\ ?jv\ p\ (s\ i)\ (s\ (Suc\ i))$ 
    apply (rule INF-mono)
    apply (rule connect-dthreads)
    by (auto simp: connected)

  with <increasing s> p1
  have is-desc-thread ?jv p
    by (rule mk-inf-desc-thread)
  with exI show ?A .
qed

```

```

lemma repeated-edge:
  assumes  $\bigwedge i. i > n \implies dsc\ (snd\ (p\ i))\ k\ k$ 
  shows is-desc-thread  $(\lambda i. k)\ p$ 
proof-
  have th:  $\forall m. \exists na > m. n < na$  by arith
  show ?thesis using prems
  unfolding is-desc-thread-def
  apply (auto)
  apply (rule-tac x=Suc n in exI, auto)
  apply (rule INF-mono[where P= $\lambda i. n < i$ ])
  apply (simp only: INF-nat)

```

by (auto simp add: th)
qed

lemma *fin-from-inf*:
 assumes *is-thread* $n \ \vartheta \ p$
 assumes $n < i$
 assumes $i < j$
 shows *is-fthread* $\vartheta \ p \ i \ j$
 using *prems*
 unfolding *is-thread-def is-fthread-def*
 by *auto*

5.4 Ramsey's Theorem

definition

set2pair $S = (THE \ (x,y). \ x < y \wedge S = \{x,y\})$

lemma *set2pair-conv*:
 fixes $x \ y :: nat$
 assumes $x < y$
 shows *set2pair* $\{x, y\} = (x, y)$
 unfolding *set2pair-def*
proof (rule *the-equality*, *simp-all only:split-conv split-paired-all*)
 from $\langle x < y \rangle$ show $x < y \wedge \{x,y\}=\{x,y\}$ **by** *simp*
next
 fix $a \ b$
 assume $a: a < b \wedge \{x, y\} = \{a, b\}$
 hence $\{a, b\} = \{x, y\}$ **by** *simp-all*
 hence $(a, b) = (x, y) \vee (a, b) = (y, x)$
 by (*cases* $x = y$) *auto*
 thus $(a, b) = (x, y)$
proof
 assume $(a, b) = (y, x)$
 with a and $\langle x < y \rangle$
 show *?thesis* **by** *auto*
qed
qed

definition

set2list = *inv set*

lemma *finite-set2list*:
 assumes *finite* S
 shows *set* (*set2list* S) = S
 unfolding *set2list-def*
proof (rule *f-inv-f*)
 from $\langle \text{finite } S \rangle$ **have** $\exists l. \text{set } l = S$
 by (rule *finite-list*)
 thus $S \in \text{range set}$

unfolding *image-def*
 by *auto*
 qed

corollary *RamseyNatpairs*:

fixes $S :: 'a \text{ set}$
 and $f :: \text{nat} \times \text{nat} \Rightarrow 'a$

assumes *finite S*

and $\text{in}S: \bigwedge x y. x < y \implies f(x, y) \in S$

obtains $T :: \text{nat set}$ and $s :: 'a$

where *infinite T*

and $s \in S$

and $\bigwedge x y. \llbracket x \in T; y \in T; x < y \rrbracket \implies f(x, y) = s$

proof –

from $\langle \text{finite } S \rangle$

have $\text{set}(\text{set2list } S) = S$ **by** (*rule finite-set2list*)

then

obtain l where $S: S = \text{set } l$ **by** *auto*

also from *set-conv-nth* have $\dots = \{l ! i \mid i. i < \text{length } l\}$.

finally have $S = \{l ! i \mid i. i < \text{length } l\}$.

let $?s = \text{length } l$

from $\text{in}S$

have *index-less*: $\bigwedge x y. x \neq y \implies \text{index-of } l (f(\text{set2pair } \{x, y\})) < ?s$

proof –

fix $x y :: \text{nat}$

assume *neg*: $x \neq y$

have $f(\text{set2pair } \{x, y\}) \in S$

proof (*cases* $x < y$)

case *True* **hence** $\text{set2pair } \{x, y\} = (x, y)$

by (*rule set2pair-conv*)

with *True inS*

show *?thesis* **by** *simp*

next

case *False*

with *neg* have *y-less*: $y < x$ **by** *simp*

have $x:\{x,y\} = \{y,x\}$ **by** *auto*

with *y-less* have $\text{set2pair } \{x, y\} = (y, x)$

by (*simp add:set2pair-conv*)

with *y-less inS*

show *?thesis* **by** *simp*

qed

thus $\text{index-of } l (f(\text{set2pair } \{x, y\})) < \text{length } l$

by (*simp add: S index-of-length*)

```

qed

have  $\exists Y. \text{infinite } Y \wedge$ 
  ( $\exists t. t < ?s \wedge$ 
    ( $\forall x \in Y. \forall y \in Y. x \neq y \longrightarrow$ 
       $\text{index-of } l \ (f \ (\text{set2pair } \{x, y\})) = t$ ))
  by (rule Ramsey2[of UNIV::nat set, simplified])
    (auto simp:index-less)
then obtain  $T \ i$ 
  where  $\text{inf: infinite } T$ 
  and  $i < \text{length } l$ 
  and  $d: \bigwedge x \ y. \llbracket x \in T; y \in T; x \neq y \rrbracket$ 
     $\implies \text{index-of } l \ (f \ (\text{set2pair } \{x, y\})) = i$ 
  by auto

have  $l \ ! \ i \in S$  unfolding  $S$  using  $i$ 
  by (rule nth-mem)
moreover
have  $\bigwedge x \ y. x \in T \implies y \in T \implies x < y$ 
   $\implies f \ (x, y) = l \ ! \ i$ 
proof -
  fix  $x \ y$  assume  $x \in T \ y \in T \ x < y$ 
  with  $d$  have
     $\text{index-of } l \ (f \ (\text{set2pair } \{x, y\})) = i$  by auto
  with  $\langle x < y \rangle$ 
  have  $i = \text{index-of } l \ (f \ (x, y))$ 
    by (simp add:set2pair-conv)
  with  $\langle i < \text{length } l \rangle$ 
  show  $f \ (x, y) = l \ ! \ i$ 
    by (auto intro:index-of-member[symmetric] iff:index-of-length)
qed
moreover note  $\text{inf}$ 
ultimately
show  $?thesis$  using  $\text{prems}$ 
  by blast
qed

```

5.5 Main Result

```

theorem LJA-Theorem4:
  assumes finite-acg A
  shows  $SCT \ A \longleftrightarrow SCT' \ A$ 
proof
  assume  $SCT \ A$ 

  show  $SCT' \ A$ 
  proof (rule classical)
    assume  $\neg SCT' \ A$ 

```


then obtain $n \ G$
where *in-closure*: $(\text{tcl } A) \vdash n \rightsquigarrow^G n$
and *idemp*: $G * G = G$
and *no-strict-arc*: $\forall p. \neg(G \vdash p \rightsquigarrow^\downarrow p)$
unfolding *SCT'-def no-bad-graphs-def* **by** *auto*

from *in-closure* **obtain** k
where *k-pow*: $A \wedge^k \vdash n \rightsquigarrow^G n$
and $0 < k$
unfolding *in-tcl* **by** *auto*

from *power-induces-path k-pow*
obtain *loop* **where** *loop-props*:
has-fpath $A \ \text{loop}$
 $n = \text{fst } \text{loop} \ n = \text{end-node } \text{loop}$
 $G = \text{prod } \text{loop} \ k = \text{length } (\text{snd } \text{loop})$.

with $\langle 0 < k \rangle$ **and** *path-loop-graph*
have *has-ipath* $A \ (\text{omega } \text{loop})$ **by** *blast*
with $\langle \text{SCT } A \rangle$
have *thread*: $\exists \vartheta. \text{is-desc-thread } \vartheta \ (\text{omega } \text{loop})$ **by** $(\text{auto simp:SCT-def})$

let $?s = \lambda i. k * i$
let $?cp = \lambda i::\text{nat}. (n, \ G)$

from *loop-props* **have** $\text{fst } \text{loop} = \text{end-node } \text{loop}$ **by** *auto*
with $\langle 0 < k \rangle \ \langle k = \text{length } (\text{snd } \text{loop}) \rangle$
have $\bigwedge i. (\text{omega } \text{loop}) \langle ?s \ i, ?s \ (\text{Suc } i) \rangle = \text{loop}$
by $(\text{rule sub-path-loop})$

with $\langle n = \text{fst } \text{loop} \rangle \ \langle G = \text{prod } \text{loop} \rangle \ \langle k = \text{length } (\text{snd } \text{loop}) \rangle$
have $a: \text{contract } ?s \ (\text{omega } \text{loop}) = ?cp$
unfolding *contract-def*
by $(\text{simp add:path-loop-def split-def fst-p0})$

from $\langle 0 < k \rangle$ **have** *increasing* $?s$
by $(\text{auto simp:increasing-def})$
with *thread* **have** $\exists \vartheta. \text{is-desc-thread } \vartheta \ ?cp$
unfolding $a[\text{symmetric}]$
by $(\text{unfold contract-keeps-threads[symmetric]})$

then obtain ϑ **where** *desc*: $\text{is-desc-thread } \vartheta \ ?cp$ **by** *auto*

then obtain n **where** *thr*: $\text{is-thread } n \ \vartheta \ ?cp$
unfolding *is-desc-thread-def is-thread-def*
by *auto*

have *finite* $(\text{range } \vartheta)$
proof $(\text{rule finite-range-ignore-prefix})$

```

from  $\langle \text{finite-acg } A \rangle$ 
have  $\text{finite-acg } (\text{tcl } A)$  by ( $\text{simp add:finite-tcl}$ )
with  $\text{in-closure}$  have  $\text{finite-graph } G$ 
  unfolding  $\text{finite-acg-def all-finite-def}$  by  $\text{blast}$ 
thus  $\text{finite } (\text{nodes } G)$  by ( $\text{rule finite-nodes}$ )

from  $\text{thread-image-nodes}[OF \text{ thr}]$ 
show  $\forall i \geq n. \exists i \in \text{nodes } G$  by  $\text{simp}$ 
qed
with  $\text{finite-range}$ 
obtain  $p$  where  $\text{inf-visit}: \exists_{\infty} i. \exists i = p$  by  $\text{auto}$ 

then obtain  $i$  where  $n < i \wedge i = p$ 
  by ( $\text{auto simp:INF-nat}$ )

from  $\text{desc}$ 
have  $\exists_{\infty} i. \text{descat } ?cp \ i$ 
  unfolding  $\text{is-desc-thread-def}$  by  $\text{auto}$ 
then obtain  $j$ 
  where  $i < j$  and  $\text{descat } ?cp \ j$ 
  unfolding  $\text{INF-nat}$  by  $\text{auto}$ 
from  $\text{inf-visit}$  obtain  $k$  where  $j < k \wedge k = p$ 
  by ( $\text{auto simp:INF-nat}$ )

from  $\langle i < j \rangle \langle j < k \rangle \langle n < i \rangle \text{ thr}$ 
   $\text{fin-from-inf}[of \ n \ \exists ?cp]$ 
   $\langle \text{descat } ?cp \ j \rangle$ 
have  $\text{is-desc-fthread } \exists \ ?cp \ i \ k$ 
  unfolding  $\text{is-desc-fthread-def}$ 
  by  $\text{auto}$ 

with  $\langle \exists \ k = p \rangle \langle \exists \ i = p \rangle$ 
have  $\text{dfth}: \text{has-desc-fth } ?cp \ i \ k \ p \ p$ 
  unfolding  $\text{has-desc-fth-def}$ 
  by  $\text{auto}$ 

from  $\langle i < j \rangle \langle j < k \rangle$  have  $i < k$  by  $\text{auto}$ 
hence  $\text{prod } (?cp(i, k)) = G$ 
proof ( $\text{induct } i \text{ rule:strict-inc-induct}$ )
  case  $\text{base}$  thus  $?case$  by ( $\text{simp add:sub-path-def}$ )
next
  case ( $\text{step } i$ ) thus  $?case$ 
    by ( $\text{simp add:sub-path-def upt-rec}[of \ i \ k] \text{ idemp}$ )
qed

with  $\langle i < j \rangle \langle j < k \rangle \text{ dfth Lemma7b}[of \ i \ k \ ?cp \ p \ p]$ 
have  $\text{dsc } G \ p \ p$  by  $\text{auto}$ 
with  $\text{no-strict-arc}$  have  $\text{False}$  by  $\text{auto}$ 

```

```

    thus ?thesis ..
  qed
next
  assume  $SCT' A$ 

  show  $SCT A$ 
  proof (rule classical)
    assume  $\neg SCT A$ 

    with  $SCT\text{-}def$ 
    obtain  $p$ 
      where  $ipath: has\text{-}ipath A p$ 
      and  $no\text{-}desc\text{-}th: \neg (\exists \vartheta. is\text{-}desc\text{-}thread \vartheta p)$ 
      by blast

    from  $\langle finite\text{-}acg A \rangle$ 
    have  $finite\text{-}acg (tcl A)$  by (simp add: finite-tcl)
    hence  $finite (dest\text{-}graph (tcl A))$  (is  $finite ?AG$ )
      by (simp add: finite-acg-def finite-graph-def)

    from  $pdesc\text{-}acgplus[OF ipath]$ 
    have  $a: \bigwedge x y. x < y \implies pdesc p \langle x, y \rangle \in dest\text{-}graph (tcl A)$ 
      unfolding has-edge-def .

    obtain  $S G$ 
      where  $infinite S G \in dest\text{-}graph (tcl A)$ 
      and  $all\text{-}G: \bigwedge x y. \llbracket x \in S; y \in S; x < y \rrbracket \implies$ 
         $pdesc (p \langle x, y \rangle) = G$ 
      apply (rule RamseyNatpairs[of ?AG  $\lambda(x,y). pdesc p \langle x, y \rangle$ ])
      apply (rule  $\langle finite ?AG \rangle$ )
      by (simp only: split-conv, rule a, auto)

    obtain  $n H m$  where
       $G\text{-}struct: G = (n, H, m)$  by (cases G)

    let ?s = enumerate S
    let ?q = contract ?s p

    note  $all\text{-}in\text{-}S[simp] = enumerate\text{-}in\text{-}set[OF \langle infinite S \rangle]$ 
      from  $\langle infinite S \rangle$ 
    have  $inc[simp]: increasing ?s$ 
      unfolding increasing-def by (simp add: enumerate-mono)
    note  $increasing\text{-}bij[OF this, simp]$ 

    from  $ipath\text{-}contract inc ipath$ 
    have  $has\text{-}ipath (tcl A) ?q$  .

    from  $all\text{-}G G\text{-}struct$ 
    have  $all\text{-}H: \bigwedge i. (snd (?q i)) = H$ 

```

```

    unfolding contract-def
  by simp

have loop: (tcl A) ⊢ n ∼H n
and idemp: H * H = H
proof -
  let ?i = ?s 0 and ?j = ?s (Suc 0) and ?k = ?s (Suc (Suc 0))

  have pdesc (p⟨?i, ?j⟩) = G
    and pdesc (p⟨?j, ?k⟩) = G
    and pdesc (p⟨?i, ?k⟩) = G
    using all-G
    by auto

  with G-struct
  have m = end-node (p⟨?i, ?j⟩)
    n = fst (p⟨?j, ?k⟩)
    and Hs: prod (p⟨?i, ?j⟩) = H
    prod (p⟨?j, ?k⟩) = H
    prod (p⟨?i, ?k⟩) = H
    by auto

  hence m = n by simp
  thus tcl A ⊢ n ∼H n
    using G-struct ⟨G ∈ dest-graph (tcl A)⟩
    by (simp add: has-edge-def)

  from sub-path-prod[of ?i ?j ?k p]
  show H * H = H
    unfolding Hs by simp
qed
moreover have ⋀k. ¬dsc H k k
proof
  fix k :: 'a assume dsc H k k

  with all-H repeated-edge
  have ∃∅. is-desc-thread ∅ ?q by fast
    with inc have ∃∅. is-desc-thread ∅ p
      by (subst contract-keeps-threads)
  with no-desc-th
  show False ..
qed
ultimately
have False
  using ⟨SCT' A⟩[unfolded SCT'-def no-bad-graphs-def]
  by blast
thus ?thesis ..
qed
qed

```

end

6 Applying SCT to function definitions

theory *Interpretation*

imports *Main Misc-Tools Criterion*

begin

definition

$idseq\ R\ s\ x = (s\ 0 = x \wedge (\forall i. R\ (s\ (Suc\ i))\ (s\ i)))$

lemma *not-acc-smaller*:

assumes *notacc*: $\neg accp\ R\ x$

shows $\exists y. R\ y\ x \wedge \neg accp\ R\ y$

proof (*rule classical*)

assume $\neg ?thesis$

hence $\bigwedge y. R\ y\ x \implies accp\ R\ y$ **by** *blast*

with *accp.accI* **have** $accp\ R\ x$.

with *notacc* **show** *?thesis* **by** *contradiction*

qed

lemma *non-acc-has-idseq*:

assumes $\neg accp\ R\ x$

shows $\exists s. idseq\ R\ s\ x$

proof –

have $\exists f. \forall x. \neg accp\ R\ x \longrightarrow R\ (f\ x)\ x \wedge \neg accp\ R\ (f\ x)$

by (*rule choice, auto simp: not-acc-smaller*)

then obtain *f* **where**

in-R: $\bigwedge x. \neg accp\ R\ x \implies R\ (f\ x)\ x$

and *nia*: $\bigwedge x. \neg accp\ R\ x \implies \neg accp\ R\ (f\ x)$

by *blast*

let $?s = \lambda i. (f\ ^i\ x)$

{

fix *i*

have $\neg accp\ R\ (?s\ i)$

by (*induct i*) (*auto simp: nia* $\langle \neg accp\ R\ x \rangle$)

hence $R\ (f\ (?s\ i))\ (?s\ i)$

by (*rule in-R*)

}

hence $idseq\ R\ ?s\ x$

unfolding *idseq-def*

by *auto*

thus *?thesis* **by** *auto*
qed

types $('a, 'q) \text{ cdesc} =$
 $('q \Rightarrow \text{bool}) \times ('q \Rightarrow 'a) \times ('q \Rightarrow 'a)$

fun *in-cdesc* :: $('a, 'q) \text{ cdesc} \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$
where
in-cdesc $(\Gamma, r, l) \ x \ y = (\exists q. x = r \ q \wedge y = l \ q \wedge \Gamma \ q)$

primrec *mk-rel* :: $('a, 'q) \text{ cdesc list} \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$
where
mk-rel $[] \ x \ y = \text{False}$
 $| \text{mk-rel } (c \# cs) \ x \ y =$
 $(\text{in-cdesc } c \ x \ y \vee \text{mk-rel } cs \ x \ y)$

lemma *some-rd*:
assumes *mk-rel rds x y*
shows $\exists rd \in \text{set } rds. \text{in-cdesc } rd \ x \ y$
using *assms*
by (*induct rds*) (*auto simp:in-cdesc-def*)

lemma *ex-cs*:
assumes *idseq idseq (mk-rel rds) s x*
shows $\exists cs. \forall i. cs \ i \in \text{set } rds \wedge \text{in-cdesc } (cs \ i) \ (s \ (\text{Suc } i)) \ (s \ i)$
proof –
from *idseq*
have $a: \forall i. \exists rd \in \text{set } rds. \text{in-cdesc } rd \ (s \ (\text{Suc } i)) \ (s \ i)$
by (*auto simp:idseq-def intro:some-rd*)

show *?thesis*
by (*rule choice*) (*insert a, blast*)
qed

types $'a \text{ measures} = \text{nat} \Rightarrow 'a \Rightarrow \text{nat}$

fun *stepP* :: $('a, 'q) \text{ cdesc} \Rightarrow ('a, 'q) \text{ cdesc} \Rightarrow$
 $('a \Rightarrow \text{nat}) \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}) \Rightarrow \text{bool}$
where

$$\begin{aligned}
& \text{stepP } (\Gamma 1, r1, l1) (\Gamma 2, r2, l2) m1 m2 R \\
& = (\forall q_1 q_2. \Gamma 1 q_1 \wedge \Gamma 2 q_2 \wedge r1 q_1 = l2 q_2 \\
& \longrightarrow R (m2 (l2 q_2)) ((m1 (l1 q_1))))
\end{aligned}$$

definition

$$\begin{aligned}
& \text{decr} :: ('a, 'q) \text{cdesc} \Rightarrow ('a, 'q) \text{cdesc} \Rightarrow \\
& ('a \Rightarrow \text{nat}) \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow \text{bool}
\end{aligned}$$

where

$$\text{decr } c1 c2 m1 m2 = \text{stepP } c1 c2 m1 m2 (op <)$$

definition

$$\begin{aligned}
& \text{decreq} :: ('a, 'q) \text{cdesc} \Rightarrow ('a, 'q) \text{cdesc} \Rightarrow \\
& ('a \Rightarrow \text{nat}) \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow \text{bool}
\end{aligned}$$

where

$$\text{decreq } c1 c2 m1 m2 = \text{stepP } c1 c2 m1 m2 (op \leq)$$

definition

$$\text{no-step} :: ('a, 'q) \text{cdesc} \Rightarrow ('a, 'q) \text{cdesc} \Rightarrow \text{bool}$$

where

$$\text{no-step } c1 c2 = \text{stepP } c1 c2 (\lambda x. 0) (\lambda x. 0) (\lambda x y. \text{False})$$

lemma *decr-in-cdesc*:

$$\begin{aligned}
& \text{assumes } \text{in-cdesc } RD1 y x \\
& \text{assumes } \text{in-cdesc } RD2 z y \\
& \text{assumes } \text{decr } RD1 RD2 m1 m2 \\
& \text{shows } m2 y < m1 x \\
& \text{using } \text{assms} \\
& \text{by } (\text{cases } RD1, \text{cases } RD2, \text{auto simp:decr-def})
\end{aligned}$$

lemma *decreq-in-cdesc*:

$$\begin{aligned}
& \text{assumes } \text{in-cdesc } RD1 y x \\
& \text{assumes } \text{in-cdesc } RD2 z y \\
& \text{assumes } \text{decreq } RD1 RD2 m1 m2 \\
& \text{shows } m2 y \leq m1 x \\
& \text{using } \text{assms} \\
& \text{by } (\text{cases } RD1, \text{cases } RD2, \text{auto simp:decreq-def})
\end{aligned}$$

lemma *no-inf-desc-nat-sequence*:

$$\begin{aligned}
& \text{fixes } s :: \text{nat} \Rightarrow \text{nat} \\
& \text{assumes } \text{leq: } \bigwedge i. n \leq i \implies s (Suc i) \leq s i \\
& \text{assumes } \text{less: } \exists_{\infty} i. s (Suc i) < s i \\
& \text{shows } \text{False}
\end{aligned}$$

proof –

$$\begin{aligned}
& \{ \\
& \quad \text{fix } i j :: \text{nat}
\end{aligned}$$

```

    assume  $n \leq i$ 
    assume  $i \leq j$ 
    {
      fix  $k$ 
      have  $s (i + k) \leq s i$ 
      proof (induct  $k$ )
        case 0 thus ?case by simp
      next
        case (Suc  $k$ )
        with  $leq[of i + k] \langle n \leq i \rangle$ 
        show ?case by simp
      qed
    }
    from  $this[of j - i] \langle n \leq i \rangle \langle i \leq j \rangle$ 
    have  $s j \leq s i$  by auto
  }
note  $decr = this$ 

let  $?min = LEAST x. x \in range (\lambda i. s (n + i))$ 
have  $?min \in range (\lambda i. s (n + i))$ 
  by (rule LeastI) auto
then obtain  $k$  where  $min: ?min = s (n + k)$  by auto

from  $less$ 
obtain  $k'$  where  $n + k < k'$ 
  and  $s (Suc k') < s k'$ 
  unfolding  $INF-nat$  by auto

with  $decr[of n + k k'] min$ 
have  $s (Suc k') < ?min$  by auto
moreover from  $\langle n + k < k' \rangle$ 
have  $s (Suc k') = s (n + (Suc k' - n))$  by simp
ultimately
show  $False$  using  $not-less-Least$  by blast
qed

```

definition

```

approx :: nat scg  $\Rightarrow$  ('a, 'q) cdesc  $\Rightarrow$  ('a, 'q) cdesc
 $\Rightarrow$  'a measures  $\Rightarrow$  'a measures  $\Rightarrow$  bool
where
approx  $G C C' M M'$ 
= ( $\forall i j. (dsc G i j \longrightarrow decr C C' (M i) (M' j))$ )
 $\wedge (eqp G i j \longrightarrow decreq C C' (M i) (M' j))$ 

```


lemma *approx-empty*:
approx (*Graph* {}) *c1 c2 ms1 ms2*
unfolding *approx-def has-edge-def dest-graph.simps* **by** *simp*

lemma *approx-less*:
assumes *stepP c1 c2 (ms1 i) (ms2 j) (op <)*
assumes *approx (Graph Es) c1 c2 ms1 ms2*
shows *approx (Graph (insert (i, ↓, j) Es)) c1 c2 ms1 ms2*
using *assms*
unfolding *approx-def has-edge-def dest-graph.simps decr-def*
by *auto*

lemma *approx-leq*:
assumes *stepP c1 c2 (ms1 i) (ms2 j) (op ≤)*
assumes *approx (Graph Es) c1 c2 ms1 ms2*
shows *approx (Graph (insert (i, ↓↓, j) Es)) c1 c2 ms1 ms2*
using *assms*
unfolding *approx-def has-edge-def dest-graph.simps decreq-def*
by *auto*

lemma *approx (Graph {(1, ↓, 2), (2, ↓↓, 3)}) c1 c2 ms1 ms2*
apply (*intro approx-less approx-leq approx-empty*)
oops

lemma *no-stepI*:
stepP c1 c2 m1 m2 (λx y. False)
 \Rightarrow *no-step c1 c2*
by (*cases c1, cases c2*) (*auto simp: no-step-def*)

definition
sound-int :: *nat acg* \Rightarrow (*'a, 'q*) *cdesc list*
 \Rightarrow *'a measures list* \Rightarrow *bool*
where
sound-int *A RDs M* =
 $(\forall n < \text{length } RDs. \forall m < \text{length } RDs.$
 $\text{no-step } (RDs ! n) (RDs ! m) \vee$
 $(\exists G. (\mathcal{A} \vdash n \rightsquigarrow^G m) \wedge \text{approx } G (RDs ! n) (RDs ! m) (M ! n) (M ! m)))$

lemma *length-simps*: *length [] = 0 length (x#xs) = Suc (length xs)*
by *auto*

lemma *all-less-zero*: $\forall n < (0 :: \text{nat}). P\ n$
by *simp*

lemma *all-less-Suc*:
assumes *Pk*: $P\ k$
assumes *Pn*: $\forall n < k. P\ n$
shows $\forall n < \text{Suc}\ k. P\ n$
proof (*intro allI impI*)
fix *n* **assume** $n < \text{Suc}\ k$
show $P\ n$
proof (*cases n < k*)
case *True* **with** *Pn* **show** *?thesis* **by** *simp*
next
case *False* **with** $\langle n < \text{Suc}\ k \rangle$ **have** $n = k$ **by** *simp*
with *Pk* **show** *?thesis* **by** *simp*
qed
qed

lemma *step-witness*:
assumes *in-cdesc RD1* $y\ x$
assumes *in-cdesc RD2* $z\ y$
shows $\neg \text{no-step}\ RD1\ RD2$
using *assms*
by (*cases RD1, cases RD2*) (*auto simp: no-step-def*)

theorem *SCT-on-relations*:
assumes *R*: $R = \text{mk-rel}\ RDs$
assumes *sound*: *sound-int* $\mathcal{A}\ RDs\ M$
assumes *SCT* \mathcal{A}
shows $\forall x. \text{accp}\ R\ x$
proof (*rule, rule classical*)
fix *x*
assume $\neg \text{accp}\ R\ x$
with *non-acc-has-idseq*
have $\exists s. \text{idseq}\ R\ s\ x$.
then obtain *s* **where** $\text{idseq}\ R\ s\ x$..
hence $\exists cs. \forall i. cs\ i \in \text{set}\ RDs \wedge$
 $\text{in-cdesc}\ (cs\ i)\ (s\ (\text{Suc}\ i))\ (s\ i)$
unfolding *R* **by** (*rule ex-cs*)
then obtain *cs* **where**
 $[simp]: \bigwedge i. cs\ i \in \text{set}\ RDs$
and $\text{ird}[simp]: \bigwedge i. \text{in-cdesc}\ (cs\ i)\ (s\ (\text{Suc}\ i))\ (s\ i)$
by *blast*

let *?cis* = $\lambda i. \text{index-of}\ RDs\ (cs\ i)$
have $\forall i. \exists G. (\mathcal{A} \vdash ?cis\ i \rightsquigarrow^G (?cis\ (\text{Suc}\ i)))$
 $\wedge \text{approx}\ G\ (RDs\ !\ ?cis\ i)\ (RDs\ !\ ?cis\ (\text{Suc}\ i))$

```

    (M ! ?cis i) (M ! ?cis (Suc i)) (is  $\forall i. \exists G. ?P i G$ )
  proof
    fix i
    let ?n = ?cis i and ?n' = ?cis (Suc i)

    have in-cdesc (RDs ! ?n) (s (Suc i)) (s i)
      in-cdesc (RDs ! ?n') (s (Suc (Suc i))) (s (Suc i))
      by (simp-all add:index-of-member)
    with step-witness
    have  $\neg$  no-step (RDs ! ?n) (RDs ! ?n') .
    moreover have
      ?n < length RDs
      ?n' < length RDs
      by (simp-all add:index-of-length[symmetric])
    ultimately
    obtain G
      where  $\mathcal{A} \vdash ?n \rightsquigarrow^G ?n'$ 
      and approx G (RDs ! ?n) (RDs ! ?n') (M ! ?n) (M ! ?n')
      using sound
      unfolding sound-int-def by auto

    thus  $\exists G. ?P i G$  by blast
  qed
  with choice
  have  $\exists Gs. \forall i. ?P i (Gs i)$  .
  then obtain Gs where
    A:  $\bigwedge i. \mathcal{A} \vdash ?cis i \rightsquigarrow^{(Gs i)} (?cis (Suc i))$ 
    and B:  $\bigwedge i. approx (Gs i) (RDs ! ?cis i) (RDs ! ?cis (Suc i))$ 
    (M ! ?cis i) (M ! ?cis (Suc i))
    by blast

  let ?p =  $\lambda i. (?cis i, Gs i)$ 

  from A have has-ipath  $\mathcal{A} ?p$ 
    unfolding has-ipath-def
    by auto

  with  $\langle SCT \ \mathcal{A} \rangle$  SCT-def
  obtain th where is-desc-thread th ?p
    by auto

  then obtain n
    where fr:  $\forall i \geq n. eqlat ?p th i$ 
    and inf:  $\exists_{\infty} i. descatt ?p th i$ 
    unfolding is-desc-thread-def by auto

  from B
  have approx:
     $\bigwedge i. approx (Gs i) (cs i) (cs (Suc i))$ 

```

```

      (M ! ?cis i) (M ! ?cis (Suc i))
    by (simp add:index-of-member)

let ?seq =  $\lambda i.$  (M ! ?cis i) (th i) (s i)

have  $\bigwedge i. n < i \implies ?seq (Suc i) \leq ?seq i$ 
proof -
  fix i
  let ?q1 = th i and ?q2 = th (Suc i)
  assume n < i

  with fr have eqlat ?p th i by simp
  hence dsc (Gs i) ?q1 ?q2  $\vee$  eqp (Gs i) ?q1 ?q2
by simp
  thus ?seq (Suc i)  $\leq$  ?seq i
proof
  assume dsc (Gs i) ?q1 ?q2

  with approx
  have a:decr (cs i) (cs (Suc i))
    ((M ! ?cis i) ?q1) ((M ! ?cis (Suc i)) ?q2)
    unfolding approx-def by auto

  show ?thesis
    apply (rule less-imp-le)
    apply (rule decr-in-cdesc[of - s (Suc i) s i])
    by (rule ird a)+
next
  assume eqp (Gs i) ?q1 ?q2

  with approx
  have a:decreq (cs i) (cs (Suc i))
    ((M ! ?cis i) ?q1) ((M ! ?cis (Suc i)) ?q2)
    unfolding approx-def by auto

  show ?thesis
    apply (rule decreq-in-cdesc[of - s (Suc i) s i])
    by (rule ird a)+
qed
qed
moreover have  $\exists_{\infty} i. ?seq (Suc i) < ?seq i$  unfolding INF-nat
proof
  fix i
  from inf obtain j where i < j and d: descst ?p th j
    unfolding INF-nat by auto
  let ?q1 = th j and ?q2 = th (Suc j)
  from d have dsc (Gs j) ?q1 ?q2 by auto

  with approx

```

```

have a:decr (cs j) (cs (Suc j))
  ((M ! ?cis j) ?q1) ((M ! ?cis (Suc j)) ?q2)
  unfolding approx-def by auto

have ?seq (Suc j) < ?seq j
  apply (rule decr-in-cdesc[of - s (Suc j) s j])
  by (rule ird a)+
with ⟨i < j⟩
show ∃ j. i < j ∧ ?seq (Suc j) < ?seq j by auto
qed
ultimately have False
  by (rule no-inf-desc-nat-sequence[of Suc n]) simp
thus accp R x ..
qed

end

```

7 Implementation of the SCT criterion

```

theory Implementation
imports Correctness
begin

```

```

fun edges-match :: ('n × 'e × 'n) × ('n × 'e × 'n) ⇒ bool
where
  edges-match ((n, e, m), (n', e', m')) = (m = n')

```

```

fun connect-edges ::
  ('n × ('e::times) × 'n) × ('n × 'e × 'n)
  ⇒ ('n × 'e × 'n)
where
  connect-edges ((n,e,m), (n', e', m')) = (n, e * e', m')

```

```

lemma grcomp-code [code]:
  grcomp (Graph G) (Graph H) = Graph (connect-edges ' { x ∈ G×H. edges-match
x })
  by (rule graph-ext) (auto simp:graph-mult-def has-edge-def image-def)

```

```

lemma mk-tcl-finite-terminates:
  fixes A :: 'a acg
  assumes fA: finite-acg A
  shows mk-tcl-dom (A, A)
proof -
  from fA have fin-tcl: finite-acg (tcl A)
    by (simp add:finite-tcl)

  hence finite (dest-graph (tcl A))

```

unfolding *finite-acg-def finite-graph-def ..*

let $?count = \lambda G. \text{card } (dest\text{-}graph \ G)$

let $?N = ?count \ (tcl \ A)$

let $?m = \lambda X. ?N - (?count \ X)$

let $?P = \lambda X. mk\text{-}tcl\text{-}dom \ (A, \ X)$

```
{
  fix X
  assume  $X \leq tcl \ A$ 
  then
  have  $mk\text{-}tcl\text{-}dom \ (A, \ X)$ 
  proof (induct X rule:measure-induct-rule[of ?m])
    case (less X)
    show ?case
    proof (cases  $X * A \leq X$ )
      case True
      with  $mk\text{-}tcl.domintros$  show ?thesis by auto
    next
      case False
      then have  $l: X < X + X * A$ 
        unfolding graph-less-def graph-leq-def graph-plus-def
        by auto

      from  $\langle X \leq tcl \ A \rangle$ 
      have  $X * A \leq tcl \ A * A$  by (simp add:mult-mono)
      also have  $\dots \leq A + tcl \ A * A$  by simp
      also have  $\dots = tcl \ A$  by (simp add:tcl-unfold-right[symmetric])
      finally have  $X * A \leq tcl \ A$  .
      with  $\langle X \leq tcl \ A \rangle$ 
      have  $X + X * A \leq tcl \ A + tcl \ A$ 
        by (rule add-mono)
      hence less-tcl:  $X + X * A \leq tcl \ A$  by simp
      hence  $X < tcl \ A$ 
        using  $l \ \langle X \leq tcl \ A \rangle$  by auto

      from less-tcl fin-tcl
      have finite-acg  $(X + X * A)$  by (rule finite-acg-subset)
      hence finite  $(dest\text{-}graph \ (X + X * A))$ 
        unfolding finite-acg-def finite-graph-def ..

      hence  $X: ?count \ X < ?count \ (X + X * A)$ 
        using  $l[simplified \ graph\text{-}less\text{-}def \ graph\text{-}leq\text{-}def]$ 
        by (rule psubset-card-mono)

      have  $?count \ X < ?N$ 
        apply (rule psubset-card-mono)
        by fact (rule  $\langle X < tcl \ A \rangle[simplified \ graph\text{-}less\text{-}def]$ )
    }
}
```

```

    with  $X$  have  $?m (X + X * A) < ?m X$  by arith

    from less.hyps this less-tcl
    have mk-tcl-dom  $(A, X + X * A)$  .
    with mk-tcl.domintros show ?thesis .
  qed
qed
}
from this less-tcl show ?thesis .
qed

```

```

lemma mk-tcl-finite-tcl:
  fixes  $A :: 'a\ acg$ 
  assumes  $fA$ : finite-acg  $A$ 
  shows mk-tcl  $A\ A = tcl\ A$ 
  using mk-tcl-finite-terminates[OF  $fA$ ]
  by (simp only: tcl-def mk-tcl-correctness star-commute)

```

```

definition test-SCT ::  $nat\ acg \Rightarrow bool$ 
where
  test-SCT  $\mathcal{A} =$ 
    (let  $\mathcal{T} = mk-tcl\ \mathcal{A}\ \mathcal{A}$ 
     in  $(\forall (n,G,m) \in dest-graph\ \mathcal{T}.$ 
        $n \neq m \vee G * G \neq G \vee$ 
        $(\exists (p::nat,e,q) \in dest-graph\ G. p = q \wedge e = LESS)))$ )

```

```

lemma SCT'-exec:
  assumes fin: finite-acg  $A$ 
  shows SCT'  $A = test-SCT\ A$ 
  using mk-tcl-finite-tcl[OF fin]
  unfolding test-SCT-def Let-def
  unfolding SCT'-def no-bad-graphs-def has-edge-def
  by force

```

```

code-modulename SML
  Implementation Graphs

```

```

lemma [code func]:
   $(G::('a::eq, 'b::eq)\ graph) \leq H \iff dest-graph\ G \subseteq dest-graph\ H$ 
   $(G::('a::eq, 'b::eq)\ graph) < H \iff dest-graph\ G \subset dest-graph\ H$ 
  unfolding graph-leq-def graph-less-def by rule+

```

```

lemma [code func]:
   $(G::('a::eq, 'b::eq)\ graph) + H = Graph\ (dest-graph\ G \cup dest-graph\ H)$ 
  unfolding graph-plus-def ..

```

```

lemma [code func]:
  (G::('a::eq, 'b::{eq, times}) graph) * H = grcomp G H
unfolding graph-mult-def ..

```

```

lemma SCT'-empty: SCT' (Graph {})
unfolding SCT'-def no-bad-graphs-def graph-zero-def[symmetric]
  tcl-zero
by (simp add:in-grzero)

```

7.1 Witness checking

definition test-SCT-witness :: nat acg \Rightarrow nat acg \Rightarrow bool
where

```

  test-SCT-witness A T =
    (A  $\leq$  T  $\wedge$  A * T  $\leq$  T  $\wedge$ 
     ( $\forall (n, G, m) \in \text{dest-graph } T.$ 
      n  $\neq$  m  $\vee$  G * G  $\neq$  G  $\vee$ 
      ( $\exists (p::\text{nat}, e, q) \in \text{dest-graph } G. p = q \wedge e = \text{LESS}))$ ))

```

```

lemma no-bad-graphs-ucl:
  assumes A  $\leq$  B
  assumes no-bad-graphs B
  shows no-bad-graphs A
  using assms
  unfolding no-bad-graphs-def has-edge-def graph-leq-def
  by blast

```

```

lemma SCT'-witness:
  assumes a: test-SCT-witness A T
  shows SCT' A
proof -
  from a have A  $\leq$  T A * T  $\leq$  T by (auto simp:test-SCT-witness-def)
  hence A + A * T  $\leq$  T
    by (subst add-idem[of T, symmetric], rule add-mono)
  with star3' have tcl A  $\leq$  T unfolding tcl-def .
  moreover
  from a have no-bad-graphs T
    unfolding no-bad-graphs-def test-SCT-witness-def has-edge-def
    by auto
  ultimately
  show ?thesis
    unfolding SCT'-def
    by (rule no-bad-graphs-ucl)
qed

```



```

code-modulename SML
  Graphs SCT
  Kleene-Algebras SCT
  Implementation SCT

```

```

end

```

8 Size-Change Termination

```

theory Size-Change-Termination
imports Correctness Interpretation Implementation
uses sct.ML
begin

```

8.1 Simplifier setup

This is needed to run the SCT algorithm in the simplifier:

```

lemma setbcomp-simps:
   $\{x \in \{\}. P\ x\} = \{\}$ 
   $\{x \in \text{insert } y\ ys. P\ x\} = (\text{if } P\ y \text{ then insert } y\ \{x \in ys. P\ x\} \text{ else } \{x \in ys. P\ x\})$ 
by auto

```

```

lemma setbcomp-cong:
   $A = B \implies (\bigwedge x. P\ x = Q\ x) \implies \{x \in A. P\ x\} = \{x \in B. Q\ x\}$ 
by auto

```

```

lemma cartprod-simps:
   $\{\} \times A = \{\}$ 
   $\text{insert } a\ A \times B = \text{Pair } a\ 'B \cup (A \times B)$ 
by (auto simp:image-def)

```

```

lemma image-simps:
   $fu\ ' \{\} = \{\}$ 
   $fu\ ' \text{insert } a\ A = \text{insert } (fu\ a)\ (fu\ ' A)$ 
by (auto simp:image-def)

```

```

lemmas union-simps =
  Un-empty-left Un-empty-right Un-insert-left

```

```

lemma subset-simps:
   $\{\} \subseteq B$ 
   $\text{insert } a\ A \subseteq B \equiv a \in B \wedge A \subseteq B$ 
by auto

```

lemma *element-simps*:

$x \in \{\} \equiv \text{False}$

$x \in \text{insert } a \ A \equiv x = a \vee x \in A$

by *auto*

lemma *set-eq-simp*:

$A = B \longleftrightarrow A \subseteq B \wedge B \subseteq A$ **by** *auto*

lemma *ball-simps*:

$\forall x \in \{\}. P \ x \equiv \text{True}$

$(\forall x \in \text{insert } a \ A. P \ x) \equiv P \ a \wedge (\forall x \in A. P \ x)$

by *auto*

lemma *bex-simps*:

$\exists x \in \{\}. P \ x \equiv \text{False}$

$(\exists x \in \text{insert } a \ A. P \ x) \equiv P \ a \vee (\exists x \in A. P \ x)$

by *auto*

lemmas *set-simps* =

setbcomp-simps

cartprod-simps image-simps union-simps subset-simps

element-simps set-eq-simp

ball-simps bex-simps

lemma *sedge-simps*:

$\downarrow * x = \downarrow$

$\Downarrow * x = x$

by (*auto simp:mult-sedge-def*)

lemmas *sctTest-simps* =

simp-thms

if-True

if-False

nat.inject

nat.distinct

Pair-eq

grcomp-code

edges-match.simps

connect-edges.simps

sedge-simps

sedge.distinct

set-simps

graph-mult-def

graph-leq-def

dest-graph.simps

```

graph-plus-def
graph.inject
graph-zero-def

test-SCT-def
mk-tcl-code

Let-def
split-conv

lemmas sctTest-congs =
  if-weak-cong let-weak-cong setbcomp-cong

lemma SCT-Main:
  finite-acg A  $\implies$  test-SCT A  $\implies$  SCT A
  using LJA-Theorem4 SCT'-exec
  by auto

end

```

9 Examples for Size-Change Termination

```

theory Examples
imports Size-Change-Termination
begin

function f :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat
where
  f n 0 = n
  | f 0 (Suc m) = f (Suc m) m
  | f (Suc n) (Suc m) = f m n
by pat-completeness auto

termination
  unfolding f-rel-def lfp-const
  apply (rule SCT-on-relations)
  apply (tactic Sct.abs-rel-tac)
  apply (rule ext, rule ext, simp)
  apply (tactic Sct.mk-call-graph @{context})
  apply (rule SCT-Main)
  apply (simp add:finite-acg-simps)
  oops

function p :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat
where
  p m n r = (if r>0 then p m (r - 1) n else

```

```

      if  $n > 0$  then  $p\ r\ (n - 1)\ m$ 
      else  $m$ )
by pat-completeness auto

termination
  unfolding p-rel-def lfp-const
  apply (rule SCT-on-relations)
  apply (tactic Sct.abs-rel-tac)
  apply (rule ext, rule ext, simp)
  apply (tactic Sct.mk-call-graph @{context})
  apply (rule SCT-Main)
  apply (simp add:finite-acg-ins finite-acg-empty finite-graph-def)
oops

function foo :: bool  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat
where
  foo True (Suc n) m = foo True n (Suc m)
| foo True 0 m = foo False 0 m
| foo False n (Suc m) = foo False (Suc n) m
| foo False n 0 = n
by pat-completeness auto

termination
  unfolding foo-rel-def lfp-const
  apply (rule SCT-on-relations)
  apply (tactic Sct.abs-rel-tac)
  apply (rule ext, rule ext, simp)
  apply (tactic Sct.mk-call-graph @{context})
  apply (rule SCT-Main)
  apply (simp add:finite-acg-ins finite-acg-empty finite-graph-def)
oops

function (sequential)
  bar :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat
where
  bar 0 (Suc n) m = bar m m m
| bar k n m = 0
by pat-completeness auto

termination
  unfolding bar-rel-def lfp-const
  apply (rule SCT-on-relations)
  apply (tactic Sct.abs-rel-tac)
  apply (rule ext, rule ext, simp)
  apply (tactic Sct.mk-call-graph @{context})
  apply (rule SCT-Main)
  apply (simp add:finite-acg-ins finite-acg-empty finite-graph-def)
  by (simp only:sctTest-simps cong: sctTest-congs)

```

end