Size-Change Termination

Alexander Krauss

November 22, 2007

1 Miscellaneous Tools for Size-Change Termination

theory *Misc-Tools*
imports *Main*
begin

1.1 Searching in lists

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

lemma *index-of-member*:
$(x \in \mathit{set}\ l) \Longrightarrow (l\ !\ \mathit{index\text{-}of}\ l\ x = x)$
$\langle \mathit{proof} \rangle$

lemma *index-of-length*:
$(x \in \mathit{set}\ l) = (\mathit{index\text{-}of}\ l\ x < \mathit{length}\ l)$
$\langle \mathit{proof} \rangle$

1.2 Some reasoning tools

lemma *three-cases*:
assumes $a1 \Longrightarrow \mathit{thesis}$
assumes $a2 \Longrightarrow \mathit{thesis}$
assumes $a3 \Longrightarrow \mathit{thesis}$
assumes $\bigwedge R.\ [\![a1 \Longrightarrow R;\ a2 \Longrightarrow R;\ a3 \Longrightarrow R]\!] \Longrightarrow R$
shows *thesis*
$\langle \mathit{proof} \rangle$

1.3 Sequences

types
${'a}\ \mathit{sequence} = \mathit{nat} \Rightarrow {'a}$

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 $s\; i < s\; j$
$\langle proof \rangle$

lemma *increasing-weak*:
assumes *increasing* s
assumes $i \leq j$
shows $s\; i \leq s\; j$
$\langle proof \rangle$

lemma *increasing-inc*:
assumes *increasing* s
shows $n \leq s\; n$
$\langle proof \rangle$

lemma *increasing-bij*:
assumes [*simp*]: *increasing* s
shows $(s\; i < s\; j) = (i < j)$
$\langle proof \rangle$

1.3.2 Sections induced by an increasing sequence

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

definition
section-of $s\; n = (LEAST\; i.\; n < s\; (Suc\; i))$

lemma *section-help*:
assumes *increasing* s
shows $\exists\, i.\; n < s\; (Suc\; i)$
$\langle proof \rangle$

lemma *section-of2*:
assumes *increasing* s
shows $n < s\; (Suc\; (section\text{-}of\; s\; n))$
$\langle proof \rangle$

lemma *section-of1*:
assumes [*simp*, *intro*]: *increasing* s
assumes $s\; i \leq n$
shows $s\; (section\text{-}of\; s\; n) \leq n$

⟨*proof*⟩

lemma *section-of-known*:
  assumes [*simp*]: *increasing s*
  assumes *in-sect*: $k \in$ *section s i*
  shows *section-of s k* $= i$ (**is** *?s* $= i$)
⟨*proof*⟩

lemma *in-section-of*:
  assumes *increasing s*
  assumes $s\ i \leq k$
  shows $k \in$ *section s* (*section-of s k*)
  ⟨*proof*⟩

end

2 Kleene Algebras

theory *Kleene-Algebras*
imports *Main*
begin

A type class of kleene algebras

class *star* = *type* +
  fixes *star* :: $'a \Rightarrow 'a$

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

lemma *add-idem2*[*simp*]: (x::$'a$::*idem-add*) + $(x + y) = x + y$
  ⟨*proof*⟩

class *order-by-add* = *idem-add* + *ord* +
  assumes *order-def*: $a \leq b \longleftrightarrow a + b = b$
  assumes *strict-order-def*: $a < b \longleftrightarrow a \leq b \wedge a \neq b$

lemma *ord-simp1*[*simp*]: (x::$'a$::*order-by-add*) $\leq y \Longrightarrow x + y = y$
  ⟨*proof*⟩
lemma *ord-simp2*[*simp*]: (x::$'a$::*order-by-add*) $\leq y \Longrightarrow y + x = y$
  ⟨*proof*⟩
lemma *ord-intro*: (x::$'a$::*order-by-add*) $+ y = y \Longrightarrow x \leq y$
  ⟨*proof*⟩

instance *order-by-add* $\subseteq$ *order*
⟨*proof*⟩

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

instance *pre-kleene* ⊆ *pordered-semiring*
⟨*proof*⟩

class *kleene* = *pre-kleene* + *star* +
  assumes *star1*: *1* + *a* ∗ *star a* ≤ *star a*
  and *star2*: *1* + *star a* ∗ *a* ≤ *star a*
  and *star3*: *a* ∗ *x* ≤ *x* ⟹ *star a* ∗ *x* ≤ *x*
  and *star4*: *x* ∗ *a* ≤ *x* ⟹ *x* ∗ *star a* ≤ *x*

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

lemma *plus-leI*:
  fixes *x* :: ′*a* :: *order-by-add*
  shows *x* ≤ *z* ⟹ *y* ≤ *z* ⟹ *x* + *y* ≤ *z*
  ⟨*proof*⟩

lemma *le-SUPI′*:
  fixes *l* :: ′*a* :: *complete-lattice*
  assumes *l* ≤ *M i*
  shows *l* ≤ (*SUP i*. *M i*)
  ⟨*proof*⟩

lemma *zero-minimum*[*simp*]: (*0*::′*a*::*pre-kleene*) ≤ *x*
  ⟨*proof*⟩

instance *kleene-by-complete-lattice* ⊆ *kleene*
⟨*proof*⟩

lemma *less-add*[*simp*]:
  fixes *a b* :: ′*a* :: *order-by-add*
  shows *a* ≤ *a* + *b*
  and *b* ≤ *a* + *b*
  ⟨*proof*⟩

lemma *add-est1*:
  fixes *a b c* :: ′*a* :: *order-by-add*
  assumes *a*: *a* + *b* ≤ *c*
  shows *a* ≤ *c*
  ⟨*proof*⟩

lemma *add-est2*:
  fixes *a b c* :: ′*a* :: *order-by-add*
  assumes *a*: *a* + *b* ≤ *c*
  shows *b* ≤ *c*
  ⟨*proof*⟩

lemma *star3'*:
  fixes $a\ b\ x :: 'a :: kleene$
  assumes a: $b + a * x \le x$
  shows $star\ a * b \le x$
$\langle proof \rangle$

lemma *star4'*:
  fixes $a\ b\ x :: 'a :: kleene$
  assumes a: $b + x * a \le x$
  shows $b * star\ a \le x$
$\langle proof \rangle$

lemma *star-idemp*:
  fixes $x :: 'a :: kleene$
  shows $star\ (star\ x) = star\ x$
  $\langle proof \rangle$

lemma *star-unfold-left*:
  fixes $a :: 'a :: kleene$
  shows $1 + a * star\ a = star\ a$
$\langle proof \rangle$

lemma *star-unfold-right*:
  fixes $a :: 'a :: kleene$
  shows $1 + star\ a * a = star\ a$
$\langle proof \rangle$

lemma *star-zero*[*simp*]:
  shows $star\ (0::'a::kleene) = 1$
  $\langle proof \rangle$

lemma *star-commute*:
  fixes $a\ b\ x :: 'a :: kleene$
  assumes a: $a * x = x * b$
  shows $star\ a * x = x * star\ b$
$\langle proof \rangle$

lemma *star-assoc*:
  fixes $c\ d :: 'a :: kleene$
  shows $star\ (c * d) * c = c * star\ (d * c)$
  $\langle proof \rangle$

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

$\langle proof \rangle$

lemma *star-one*:
fixes $a\ p\ p' :: 'a :: kleene$
assumes $p * p' = 1$ **and** $p' * p = 1$
shows $p' * star\ a * p = star\ (p' * a * p)$
$\langle proof \rangle$

lemma *star-mono*:
fixes $x\ y :: 'a :: kleene$
assumes $x \leq y$
shows $star\ x \leq star\ y$
$\langle proof \rangle$

lemma *x-less-star*[*simp*]:
fixes $x :: 'a :: kleene$
shows $x \leq x * star\ a$
$\langle proof \rangle$

2.1 Transitive Closure

definition
$tcl\ (x::'a::kleene) = star\ x * x$

lemma *tcl-zero*:
$tcl\ (0::'a::kleene) = 0$
$\langle proof \rangle$

lemma *tcl-unfold-right*: $tcl\ a = a + tcl\ a * a$
$\langle proof \rangle$

lemma *less-tcl*: $a \leq tcl\ a$
$\langle proof \rangle$

2.2 Naive Algorithm to generate the transitive closure

function (*default* $\lambda x.\ 0$, *tailrec*, *domintros*)
$mk\text{-}tcl :: ('a::\{plus,times,ord,zero\}) \Rightarrow 'a \Rightarrow 'a$
where
$mk\text{-}tcl\ A\ X = (if\ X * A \leq X\ then\ X\ else\ mk\text{-}tcl\ A\ (X + X * A))$
$\langle proof \rangle$

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

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

mk-tcl A X =
(let XA = X ∗ A
in if XA ≤ X then X else mk-tcl A (X + XA))
⟨*proof*⟩

lemma *mk-tcl-lemma1*:
fixes *X* :: *'a* :: *kleene*
shows (*X* + *X* ∗ *A*) ∗ *star A* = *X* ∗ *star A*
⟨*proof*⟩

lemma *mk-tcl-lemma2*:
fixes *X* :: *'a* :: *kleene*
shows *X* ∗ *A* ≤ *X* ⟹ *X* ∗ *star A* = *X*
⟨*proof*⟩

lemma *mk-tcl-correctness*:
fixes *A X* :: *'a* :: {*kleene*}
assumes *mk-tcl-dom* (*A*, *X*)
shows *mk-tcl A X* = *X* ∗ *star A*
⟨*proof*⟩

lemma *graph-implies-dom*: *mk-tcl-graph x y* ⟹ *mk-tcl-dom x*
⟨*proof*⟩

lemma *mk-tcl-default*: ¬ *mk-tcl-dom* (*a*,*x*) ⟹ *mk-tcl a x* = *0*
⟨*proof*⟩

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

lemma *mk-tcl-correctness2*:
fixes *A X* :: *'a* :: {*kleene*}
assumes *mk-tcl A A* ≠ *0*
shows *mk-tcl A A* = *tcl A*
⟨*proof*⟩

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 $('a \times 'b \times 'a)$ *set*

fun *dest-graph* :: $('a, 'b)$ *graph* $\Rightarrow$ $('a \times 'b \times 'a)$ *set*
  where *dest-graph* (*Graph* G) = G

lemma *graph-dest-graph*[*simp*]:
  Graph (*dest-graph* G) = G
  ⟨*proof*⟩

lemma *split-graph-all*:
  $(\bigwedge gr.\ PROP\ P\ gr) \equiv (\bigwedge set.\ PROP\ P\ (Graph\ set))$
⟨*proof*⟩

definition
  has-edge :: $('n,'e)$ *graph* $\Rightarrow$ $'n \Rightarrow 'e \Rightarrow 'n \Rightarrow$ *bool*
$(- \vdash - \leadsto^{-} -)$
where
  has-edge $G\ n\ e\ n'$ = $((n,\ e,\ n') \in$ *dest-graph* $G)$

3.2 Graph composition

fun *grcomp* :: $('n, 'e{::}times)$ *graph* $\Rightarrow$ $('n, 'e)$ *graph* $\Rightarrow$ $('n, 'e)$ *graph*
where
  grcomp (*Graph* G) (*Graph* H) =
  Graph $\{(p,b,q) \mid p\ b\ q.$
  $(\exists\, k\ e\ e'.\ (p,e,k){\in}G \wedge (k,e',q){\in}H \wedge b = e * e')\}$

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

lemma *graph-ext*:
  assumes $\bigwedge n\ e\ n'.$ *has-edge* $G\ n\ e\ n'$ = *has-edge* $H\ n\ e\ n'$
  shows $G = H$
  ⟨*proof*⟩

instance *graph* :: (*type*, *type*) {*comm-monoid-add*}
  graph-zero-def: 0 == *Graph* {}
  graph-plus-def: $G + H$ == *Graph* (*dest-graph* G $\cup$ *dest-graph* H)
⟨*proof*⟩

lemmas [*code func del*] = *graph-plus-def*

instance *graph* :: (*type*, *type*) {*distrib-lattice*, *complete-lattice*}
  graph-leq-def: $G \leq H \equiv$ *dest-graph* $G \subseteq$ *dest-graph* H
  graph-less-def: $G < H \equiv$ *dest-graph* $G \subset$ *dest-graph* H

inf G H ≡ *Graph* (*dest-graph G* ∩ *dest-graph H*)
sup G H ≡ *G* + *H*
Inf-graph-def: *Inf* ≡ λ*Gs*. *Graph* (⋂(*dest-graph* ‘ *Gs*))
Sup-graph-def: *Sup* ≡ λ*Gs*. *Graph* (⋃(*dest-graph* ‘ *Gs*))
⟨*proof*⟩

lemmas [*code func del*] = *graph-leq-def graph-less-def*
inf-graph-def sup-graph-def Inf-graph-def Sup-graph-def

lemma *in-grplus*:
has-edge (*G* + *H*) *p b q* = (*has-edge G p b q* ∨ *has-edge H p b q*)
⟨*proof*⟩

lemma *in-grzero*:
has-edge 0 p b q = *False*
⟨*proof*⟩

3.2.1 Multiplicative Structure

instance *graph* :: (*type*, *times*) *mult-zero*
graph-mult-def: *G* ∗ *H* == *grcomp G H*
⟨*proof*⟩

lemmas [*code func del*] = *graph-mult-def*

instance *graph* :: (*type*, *one*) *one*
graph-one-def: *1* == *Graph* { (*x*, *1*, *x*) |*x*. *True*} ⟨*proof*⟩

lemma *in-grcomp*:
has-edge (*G* ∗ *H*) *p b q*
= (∃ *k e e′*. *has-edge G p e k* ∧ *has-edge H k e′ q* ∧ *b* = *e* ∗ *e′*)
⟨*proof*⟩

lemma *in-grunit*:
has-edge 1 p b q = (*p* = *q* ∧ *b* = *1*)
⟨*proof*⟩

instance *graph* :: (*type*, *semigroup-mult*) *semigroup-mult*
⟨*proof*⟩

fun *grpow* :: *nat* ⇒ (′*a*::*type*, ′*b*::*monoid-mult*) *graph* ⇒ (′*a*, ′*b*) *graph*
where
grpow 0 A = *1*
| *grpow* (*Suc n*) *A* = *A* ∗ (*grpow n A*)

instance *graph* :: (*type*, *monoid-mult*)
{*semiring-1*,*idem-add*,*recpower*,*star*}
graph-pow-def: *A* ˆ *n* == *grpow n A*
graph-star-def: *star G* == (*SUP n*. *G* ˆ *n*)

⟨*proof*⟩

lemma *graph-leqI*:
  assumes $\bigwedge n\ e\ n'.\ \mathit{has\text{-}edge}\ G\ n\ e\ n' \Longrightarrow \mathit{has\text{-}edge}\ H\ n\ e\ n'$
  shows $G \le H$
  ⟨*proof*⟩

lemma *in-graph-plusE*:
  assumes $\mathit{has\text{-}edge}\ (G + H)\ n\ e\ n'$
  assumes $\mathit{has\text{-}edge}\ G\ n\ e\ n' \Longrightarrow P$
  assumes $\mathit{has\text{-}edge}\ H\ n\ e\ n' \Longrightarrow P$
  shows P
  ⟨*proof*⟩

lemma *in-graph-compE*:
  assumes *GH*: $\mathit{has\text{-}edge}\ (G * H)\ n\ e\ n'$
  obtains *e1 k e2*
  where $\mathit{has\text{-}edge}\ G\ n\ e1\ k\ \mathit{has\text{-}edge}\ H\ k\ e2\ n'\ e = e1 * e2$
  ⟨*proof*⟩

lemma
  assumes $x \in S\ k$
  shows $x \in (\bigcup k.\ S\ k)$
  ⟨*proof*⟩

lemma *graph-union-least*:
  assumes $\bigwedge n.\ \mathit{Graph}\ (G\ n) \le C$
  shows $\mathit{Graph}\ (\bigcup n.\ G\ n) \le C$
  ⟨*proof*⟩

lemma *Sup-graph-eq*:
  $(\mathit{SUP}\ n.\ \mathit{Graph}\ (G\ n)) = \mathit{Graph}\ (\bigcup n.\ G\ n)$
⟨*proof*⟩

lemma *has-edge-leq*: $\mathit{has\text{-}edge}\ G\ p\ b\ q = (\mathit{Graph}\ \{(p,b,q)\} \le G)$
  ⟨*proof*⟩

lemma *Sup-graph-eq2*:
  $(\mathit{SUP}\ n.\ G\ n) = \mathit{Graph}\ (\bigcup n.\ \mathit{dest\text{-}graph}\ (G\ n))$
  ⟨*proof*⟩

lemma *in-SUP*:
  $\mathit{has\text{-}edge}\ (\mathit{SUP}\ x.\ \mathit{Gs}\ x)\ p\ b\ q = (\exists x.\ \mathit{has\text{-}edge}\ (\mathit{Gs}\ x)\ p\ b\ q)$
  ⟨*proof*⟩

instance *graph* :: (*type*, *monoid-mult*) *kleene-by-complete-lattice*
⟨*proof*⟩

lemma *in-star*:
  $\textit{has-edge}\ (\textit{star}\ G)\ a\ x\ b = (\exists\, n.\ \textit{has-edge}\ (G\ \hat{}\ n)\ a\ x\ b)$
  ⟨*proof*⟩

lemma *tcl-is-SUP*:
  $\textit{tcl}\ (G{::}('a{::}\textit{type},\ 'b{::}\textit{monoid-mult})\ \textit{graph}) =$
  $(\textit{SUP}\ n.\ G\ \hat{}\ (\textit{Suc}\ n))$
  ⟨*proof*⟩

lemma *in-tcl*:
  $\textit{has-edge}\ (\textit{tcl}\ G)\ a\ x\ b = (\exists\, n{>}0.\ \textit{has-edge}\ (G\ \hat{}\ n)\ a\ x\ b)$
  ⟨*proof*⟩

3.3 Infinite Paths

types $('n,\ 'e)\ \textit{ipath} = ('n \times 'e)\ \textit{sequence}$

definition $\textit{has-ipath} :: ('n,\ 'e)\ \textit{graph} \Rightarrow ('n,\ 'e)\ \textit{ipath} \Rightarrow \textit{bool}$
where
  $\textit{has-ipath}\ G\ p =$
  $(\forall\, i.\ \textit{has-edge}\ G\ (\textit{fst}\ (p\ i))\ (\textit{snd}\ (p\ i))\ (\textit{fst}\ (p\ (\textit{Suc}\ i))))$

3.4 Finite Paths

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

inductive $\textit{has-fpath} :: ('n,\ 'e)\ \textit{graph} \Rightarrow ('n,\ 'e)\ \textit{fpath} \Rightarrow \textit{bool}$
  for $G :: ('n,\ 'e)\ \textit{graph}$
where
  $\textit{has-fpath-empty}$: $\textit{has-fpath}\ G\ (n,\ [])$
| $\textit{has-fpath-join}$: $[\![G \vdash n \leadsto^{e} n';\ \textit{has-fpath}\ G\ (n',\ es)]\!] \Longrightarrow \textit{has-fpath}\ G\ (n,\ (e,\ n')\#es)$

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

definition $\textit{path-nth} :: ('n,\ 'e)\ \textit{fpath} \Rightarrow \textit{nat} \Rightarrow ('n \times 'e \times 'n)$
where
  $\textit{path-nth}\ p\ k = (\textit{if}\ k = 0\ \textit{then}\ \textit{fst}\ p\ \textit{else}\ \textit{snd}\ (\textit{snd}\ p\ !\ (k - 1)),\ \textit{snd}\ p\ !\ k)$

lemma *endnode-nth*:
  assumes $\textit{length}\ (\textit{snd}\ p) = \textit{Suc}\ k$
  shows $\textit{end-node}\ p = \textit{snd}\ (\textit{snd}\ (\textit{path-nth}\ p\ k))$
  ⟨*proof*⟩

lemma *path-nth-graph*:
  assumes $k < \textit{length}\ (\textit{snd}\ p)$

assumes *has-fpath G p*
shows $(\lambda(n,e,n').\ \mathit{has\text{-}edge}\ G\ n\ e\ n')\ (\mathit{path\text{-}nth}\ p\ k)$
⟨*proof*⟩

lemma *path-nth-connected*:
assumes $\mathit{Suc}\ k < \mathit{length}\ (\mathit{snd}\ p)$
shows $\mathit{fst}\ (\mathit{path\text{-}nth}\ p\ (\mathit{Suc}\ k)) = \mathit{snd}\ (\mathit{snd}\ (\mathit{path\text{-}nth}\ p\ k))$
⟨*proof*⟩

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

lemma *fst-p0*: $\mathit{fst}\ (\mathit{path\text{-}nth}\ p\ 0) = \mathit{fst}\ p$
⟨*proof*⟩

lemma *path-loop-connect*:
assumes $\mathit{fst}\ p = \mathit{end\text{-}node}\ p$
and $0 < \mathit{length}\ (\mathit{snd}\ p)$ (**is** $0 < ?l$)
shows $\mathit{fst}\ (\mathit{path\text{-}nth}\ p\ (\mathit{Suc}\ i\ \mathit{mod}\ (\mathit{length}\ (\mathit{snd}\ p))))$
$= \mathit{snd}\ (\mathit{snd}\ (\mathit{path\text{-}nth}\ p\ (i\ \mathit{mod}\ \mathit{length}\ (\mathit{snd}\ p))))$
(**is** $\ldots = \mathit{snd}\ (\mathit{snd}\ (\mathit{path\text{-}nth}\ p\ ?k))$)
⟨*proof*⟩

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

definition $\mathit{prod} :: ('n,\ 'e{::}\mathit{monoid\text{-}mult})\ \mathit{fpath} \Rightarrow 'e$
where
$\mathit{prod}\ p = \mathit{foldr}\ (\mathit{op}\ *)\ (\mathit{map}\ \mathit{fst}\ (\mathit{snd}\ p))\ 1$

lemma *prod-simps*[*simp*]:
$\mathit{prod}\ (n,\ []) = 1$
$\mathit{prod}\ (n,\ (e,n')\#es) = e * (\mathit{prod}\ (n',es))$
⟨*proof*⟩

lemma *power-induces-path*:
assumes *a*: $\mathit{has\text{-}edge}\ (A\ \hat{}\ k)\ n\ G\ m$
obtains *p*
where *has-fpath A p*
and $n = \mathit{fst}\ p\ m = \mathit{end\text{-}node}\ p$
and $G = \mathit{prod}\ p$
and $k = \mathit{length}\ (\mathit{snd}\ p)$
⟨*proof*⟩

3.5 Sub-Paths

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

lemma *sub-path-is-path*:
assumes *ipath*: *has-ipath G p*
assumes *l*: $i \leq j$
shows *has-fpath G* $(p\langle i,j\rangle)$
$\langle \mathit{proof} \rangle$

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

lemma *nth-upto*[*simp*]: $k < j - i \Longrightarrow [i\ ..<\ j]\ !\ k = i + k$
$\langle \mathit{proof} \rangle$

lemma *sub-path-end*[*simp*]:
$i < j \Longrightarrow \mathit{end\text{-}node}\ (p\langle i,j\rangle) = \mathit{fst}\ (p\ j)$
$\langle \mathit{proof} \rangle$

lemma *foldr-map*: $\mathit{foldr}\ f\ (\mathit{map}\ g\ \mathit{xs}) = \mathit{foldr}\ (f\ o\ g)\ \mathit{xs}$
$\langle \mathit{proof} \rangle$

lemma *upto-append*[*simp*]:
assumes $i \leq j\ j \leq k$
shows $[\ i\ ..<\ j\ ]\ @\ [j\ ..<\ k] = [i\ ..<\ k]$
$\langle \mathit{proof} \rangle$

lemma *foldr-monoid*: $\mathit{foldr}\ (\mathit{op}\ *)\ \mathit{xs}\ 1 * \mathit{foldr}\ (\mathit{op}\ *)\ \mathit{ys}\ 1$
$= \mathit{foldr}\ (\mathit{op}\ *)\ (\mathit{xs}\ @\ \mathit{ys})\ (1::'a::\mathit{monoid\text{-}mult})$
$\langle \mathit{proof} \rangle$

lemma *sub-path-prod*:
assumes $i < j$
assumes $j < k$
shows $\mathit{prod}\ (p\langle i,k\rangle) = \mathit{prod}\ (p\langle i,j\rangle) * \mathit{prod}\ (p\langle j,k\rangle)$
$\langle \mathit{proof} \rangle$

lemma *path-acgpow-aux*:
assumes *length es* $= l$
assumes *has-fpath G* (n,es)
shows *has-edge* $(G\ \hat{}\ l)\ n\ (\mathit{prod}\ (n,\mathit{es}))\ (\mathit{end\text{-}node}\ (n,\mathit{es}))$

⟨*proof*⟩

lemma *path-acgpow*:
  has-fpath G p
  $\Longrightarrow$ *has-edge* (*G* ^ *length* (*snd p*)) (*fst p*) (*prod p*) (*end-node p*)
⟨*proof*⟩

lemma *star-paths*:
  has-edge (*star G*) *a x b* =
  ($\exists$ *p*. *has-fpath G p* $\wedge$ *a* = *fst p* $\wedge$ *b* = *end-node p* $\wedge$ *x* = *prod p*)
⟨*proof*⟩

lemma *plus-paths*:
  has-edge (*tcl G*) *a x b* =
  ($\exists$ *p*. *has-fpath G p* $\wedge$ *a* = *fst p* $\wedge$ *b* = *end-node p* $\wedge$ *x* = *prod p* $\wedge$ *0* < *length* (*snd p*))
⟨*proof*⟩

definition
  contract s p =
  (λ*i*. (*fst* (*p* (*s i*)), *prod* (*p*⟨*s i*,*s* (*Suc i*)⟩)))

lemma *ipath-contract*:
  assumes [*simp*]: *increasing s*
  assumes *ipath*: *has-ipath G p*
  shows *has-ipath* (*tcl G*) (*contract s p*)
  ⟨*proof*⟩

lemma *prod-unfold*:
  i < *j* $\Longrightarrow$ *prod* (*p*⟨*i*,*j*⟩)
  = *snd* (*p i*) * *prod* (*p*⟨*Suc i*, *j*⟩)
  ⟨*proof*⟩

lemma *sub-path-loop*:
  assumes *0* < *k*
  assumes *k*: *k* = *length* (*snd loop*)
  assumes *loop*: *fst loop* = *end-node loop*
  shows (*omega loop*)⟨*k* * *i*,*k* * *Suc i*⟩ = *loop* (**is** *?*ω = *loop*)
⟨*proof*⟩

end

4 The Size-Change Principle (Definition)

theory *Criterion*
imports *Graphs Infinite-Set*
begin

4.1 Size-Change Graphs

datatype *sedge* =
  LESS (↓)
| *LEQ* (⇓)

instance *sedge* :: *one*
  one-sedge-def: $1 \equiv \Downarrow$ ⟨*proof*⟩

instance *sedge* :: *times*
  mult-sedge-def: $a * b \equiv$ *if* $a = \downarrow$ *then* $\downarrow$ *else* b ⟨*proof*⟩

instance *sedge* :: *comm-monoid-mult*
⟨*proof*⟩

lemma *sedge-UNIV*:
  UNIV = { *LESS*, *LEQ* }
⟨*proof*⟩

instance *sedge* :: *finite*
⟨*proof*⟩

lemmas [*code func*] = *sedge-UNIV*

types *'a scg* = (*'a*, *sedge*) *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 ≡ *has-edge G i LESS j*

abbreviation (*input*)
  eq G i j ≡ *has-edge G i LEQ j*

abbreviation
  eql :: *'a scg* ⇒ *'a* ⇒ *'a* ⇒ *bool*
(- ⊢ - ⇝ -)
where
  eql G i j ≡ *has-edge G i LESS j* ∨ *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* (ϑ (*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* (ϑ (*Suc* i))

abbreviation (*input*) *eqlat* :: ($'a$, $'a$ *scg*) *ipath* $\Rightarrow$ $'a$ *sequence* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where
eqlat $p\ \vartheta\ i \equiv$ (*has-edge* (*snd* ($p\ i$)) ($\vartheta\ i$) *LESS* (ϑ (*Suc* i))
$\vee$ *has-edge* (*snd* ($p\ i$)) ($\vartheta\ i$) *LEQ* (ϑ (*Suc* i)))

definition *is-desc-thread* :: $'a$ *sequence* $\Rightarrow$ ($'a$, $'a$ *scg*) *ipath* $\Rightarrow$ *bool*
where
is-desc-thread $\vartheta\ p$ = (($\exists\, n.\forall\, i\geq n.$ *eqlat* $p\ \vartheta\ i$) $\wedge$ ($\exists_\infty i.$ *descat* $p\ \vartheta\ i$))

definition *SCT* :: $'a$ *acg* $\Rightarrow$ *bool*
where
SCT $\mathcal{A}$ =
($\forall\, p.$ *has-ipath* $\mathcal{A}\ p \longrightarrow$ ($\exists\,\vartheta.$ *is-desc-thread* $\vartheta\ p$))

definition *no-bad-graphs* :: $'a$ *acg* $\Rightarrow$ *bool*
where
no-bad-graphs A =
($\forall\, n\ G.$ *has-edge* $A\ n\ G\ n \wedge G * G = G$
$\longrightarrow$ ($\exists\, p.$ *has-edge* $G\ p$ *LESS* p))

definition *SCT′* :: $'a$ *acg* $\Rightarrow$ *bool*
where
SCT′ A = *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 *is-thread* :: $nat \Rightarrow {'a}\ sequence \Rightarrow ({'a}, {'a}\ scg)\ ipath \Rightarrow bool$
where
  $is\text{-}thread\ n\ \vartheta\ p = (\forall\, i{\geq}n.\ eqlat\ p\ \vartheta\ i)$

definition *is-fthread* ::
  ${'a}\ sequence \Rightarrow ({'a}, {'a}\ scg)\ ipath \Rightarrow nat \Rightarrow nat \Rightarrow bool$
where
  $is\text{-}fthread\ \vartheta\ mp\ i\ j = (\forall\, k{\in}\{i..{<}j\}.\ eqlat\ mp\ \vartheta\ k)$

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

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

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

5.2 Everything is finite

lemma *finite-range*:
  fixes f :: $nat \Rightarrow {'a}$
  assumes *fin*: $finite\ (range\ f)$
  shows $\exists\, x.\ \exists_\infty i.\ f\ i = x$
⟨*proof*⟩

lemma *finite-range-ignore-prefix*:
  fixes f :: $nat \Rightarrow {'a}$
  assumes *fA*: $finite\ A$
  assumes *inA*: $\forall\, x{\geq}n.\ f\ x \in A$
  shows $finite\ (range\ f)$
⟨*proof*⟩

definition
  $finite\text{-}graph\ G = finite\ (dest\text{-}graph\ G)$
definition
  $all\text{-}finite\ G = (\forall\, n\ H\ m.\ has\text{-}edge\ G\ n\ H\ m \longrightarrow finite\text{-}graph\ H)$
definition
  $finite\text{-}acg\ A = (finite\text{-}graph\ A \wedge all\text{-}finite\ A)$

definition
nodes G = *fst* ‘ *dest-graph* $G \cup$ *snd* ‘ *snd* ‘ *dest-graph* G
definition
edges G = *fst* ‘ *snd* ‘ *dest-graph* G
definition
smallnodes $G = \bigcup$(*nodes* ‘ *edges* G)

lemma *thread-image-nodes*:
assumes *th*: *is-thread* $n\ \vartheta\ p$
shows $\forall\, i \geq n.\ \vartheta\ i \in$ *nodes* (*snd* (p i))
⟨*proof*⟩

lemma *finite-nodes*: *finite-graph* $G \Longrightarrow$ *finite* (*nodes* G)
⟨*proof*⟩

lemma *nodes-subgraph*: $A \leq B \Longrightarrow$ *nodes* $A \subseteq$ *nodes* B
⟨*proof*⟩

lemma *finite-edges*: *finite-graph* $G \Longrightarrow$ *finite* (*edges* G)
⟨*proof*⟩

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

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

lemma *finite-acg-subset*:
$A \leq B \Longrightarrow$ *finite-acg* $B \Longrightarrow$ *finite-acg* A
⟨*proof*⟩

lemma *scg-finite*:
fixes G :: $'a$ *scg*
assumes *fin*: *finite* (*nodes* G)
shows *finite-graph* G
⟨*proof*⟩

lemma *smallnodes-sum*[*simp*]:
smallnodes $(A + B)$ = *smallnodes* $A \cup$ *smallnodes* B
⟨*proof*⟩

lemma *in-smallnodes*:
fixes A :: $'a$ *acg*
assumes *e*: *has-edge* $A\ x\ G\ y$
shows *nodes* $G \subseteq$ *smallnodes* A
⟨*proof*⟩

lemma *finite-smallnodes*:
assumes *fA*: *finite-acg* A

shows *finite* (*smallnodes* A)
⟨*proof*⟩

lemma *nodes-tcl*:
nodes (*tcl* A) = *nodes* A
⟨*proof*⟩

lemma *smallnodes-tcl*:
fixes A :: $'a$ *acg*
shows *smallnodes* (*tcl* A) = *smallnodes* A
⟨*proof*⟩

lemma *finite-nodegraphs*:
assumes F: *finite* F
shows *finite* { G::$'a$ *scg*. *nodes* $G \subseteq F$ } (**is** *finite* *?P*)
⟨*proof*⟩

lemma *finite-graphI*:
fixes A :: $'a$ *acg*
assumes *fin*: *finite* (*nodes* A) *finite* (*smallnodes* A)
shows *finite-graph* A
⟨*proof*⟩

lemma *smallnodes-allfinite*:
fixes A :: $'a$ *acg*
assumes *fin*: *finite* (*smallnodes* A)
shows *all-finite* A
⟨*proof*⟩

lemma *finite-tcl*:
fixes A :: $'a$ *acg*
shows *finite-acg* (*tcl* A) $\longleftrightarrow$ *finite-acg* A
⟨*proof*⟩

lemma *finite-acg-empty*: *finite-acg* (*Graph* {})
⟨*proof*⟩

lemma *finite-acg-ins*:
assumes *fA*: *finite-acg* (*Graph* A)
assumes *fG*: *finite* G
shows *finite-acg* (*Graph* (*insert* (a, *Graph* G, b) A))
⟨*proof*⟩

lemmas *finite-acg-simps* = *finite-acg-empty* *finite-acg-ins* *finite-graph-def*

5.3 Contraction and more

abbreviation

pdesc P == (*fst* P, *prod* P, *end-node* P)

lemma *pdesc-acgplus*:
assumes *has-ipath* $\mathcal{A}$ p
and $i < j$
shows *has-edge* (*tcl* $\mathcal{A}$) (*fst* ($p\langle i,j\rangle$)) (*prod* ($p\langle i,j\rangle$)) (*end-node* ($p\langle i,j\rangle$))
$\langle proof \rangle$

lemma *combine-fthreads*:
assumes *range*: $i < j$ $j \leq k$
shows
has-fth p i k m r =
($\exists n$. *has-fth* p i j m n $\wedge$ *has-fth* p j k n r) (**is** ?L = ?R)
$\langle proof \rangle$

lemma *desc-is-fthread*:
is-desc-fthread ϑ p i k $\Longrightarrow$ *is-fthread* ϑ p i k
$\langle proof \rangle$

lemma *combine-dfthreads*:
assumes *range*: $i < j$ $j \leq k$
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)
$\langle proof \rangle$

lemma *fth-single*:
has-fth p i (*Suc* i) m n = *eql* (*snd* (p i)) m n (**is** ?L = ?R)
$\langle proof \rangle$

lemma *desc-fth-single*:
has-desc-fth p i (*Suc* i) m n =
dsc (*snd* (p i)) m n (**is** ?L = ?R)
$\langle proof \rangle$

lemma *mk-eql*: ($G \vdash m \leadsto^{e} n$) $\Longrightarrow$ *eql* G m n
$\langle proof \rangle$

lemma *eql-scgcomp*:
eql ($G * H$) m r =
($\exists n$. *eql* G m n $\wedge$ *eql* H n r) (**is** ?L = ?R)
$\langle proof \rangle$

lemma *desc-scgcomp*:
$dsc\ (G * H)\ m\ r =$
$(\exists n.\ (dsc\ G\ m\ n \wedge eql\ H\ n\ r) \vee (eq\ G\ m\ n \wedge dsc\ H\ n\ r))$ (**is** $?L = ?R$)
$\langle proof \rangle$

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

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

lemma *Lemma7a*:
$i \le j \Longrightarrow has\text{-}fth\ p\ i\ j\ m\ n = eql\ (prod\ (p\langle i,j\rangle))\ m\ n$
$\langle proof \rangle$

lemma *Lemma7b*:
assumes $i \le j$
shows
$has\text{-}desc\text{-}fth\ p\ i\ j\ m\ n =$
$dsc\ (prod\ (p\langle i,j\rangle))\ m\ n$
$\langle proof \rangle$

lemma *descat-contract*:
assumes [*simp*]: *increasing s*
shows
$descat\ (contract\ s\ p)\ \vartheta\ i =$
$has\text{-}desc\text{-}fth\ p\ (s\ i)\ (s\ (Suc\ i))\ (\vartheta\ i)\ (\vartheta\ (Suc\ i))$
$\langle proof \rangle$

lemma *eqlat-contract*:
assumes [*simp*]: *increasing s*
shows
$eqlat\ (contract\ s\ p)\ \vartheta\ i =$
$has\text{-}fth\ p\ (s\ i)\ (s\ (Suc\ i))\ (\vartheta\ i)\ (\vartheta\ (Suc\ i))$
$\langle proof \rangle$

5.3.1 Connecting threads

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

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*⟩

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

  shows
  $is\text{-}fthread\ (connect\ s\ \vartheta s)\ p\ (s\ i)\ (s\ (Suc\ i))$
  ⟨*proof*⟩

lemma *connect-dthreads*:
  assumes *inc*[*simp*]: *increasing s*
  assumes *connected*: $\vartheta s\ i\ (s\ (Suc\ i)) = \vartheta s\ (Suc\ i)\ (s\ (Suc\ i))$
  assumes *fth*: $is\text{-}desc\text{-}fthread\ (\vartheta s\ i)\ p\ (s\ i)\ (s\ (Suc\ i))$

  shows
  $is\text{-}desc\text{-}fthread\ (connect\ s\ \vartheta s)\ p\ (s\ i)\ (s\ (Suc\ i))$
  ⟨*proof*⟩

lemma *mk-inf-thread*:
  assumes [*simp*]: *increasing s*
  assumes *fths*: $\bigwedge i.\ i > n \Longrightarrow is\text{-}fthread\ \vartheta\ p\ (s\ i)\ (s\ (Suc\ i))$
  shows $is\text{-}thread\ (s\ (Suc\ n))\ \vartheta\ p$
  ⟨*proof*⟩

lemma *mk-inf-desc-thread*:
  assumes [*simp*]: *increasing s*
  assumes *fths*: $\bigwedge i.\ i > n \Longrightarrow is\text{-}fthread\ \vartheta\ p\ (s\ i)\ (s\ (Suc\ i))$
  assumes *fdths*: $\exists_\infty i.\ is\text{-}desc\text{-}fthread\ \vartheta\ p\ (s\ i)\ (s\ (Suc\ i))$
  shows $is\text{-}desc\text{-}thread\ \vartheta\ p$
  ⟨*proof*⟩

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*: $\bigwedge x\ i.\ Q\ x\ i \Longrightarrow P\ x\ i$

shows $\exists xs.\ ((\exists n.\forall i{\geq}n.\ P\ (xs\ i)\ i) \wedge (\exists_\infty i.\ Q\ (xs\ i)\ i))$
(**is** $\exists xs.\ ?Ps\ xs \wedge ?Qs\ xs$)
$\langle proof \rangle$

lemma *dthreads-join*:
assumes [*simp*]: *increasing s*
assumes *dthread*: *is-desc-thread* ϑ (*contract s p*)
shows $\exists \vartheta s.\ desc\ (\lambda i.\ \textit{is-fthread}\ (\vartheta s\ i)\ p\ (s\ i)\ (s\ (Suc\ i))$
$\wedge\ \vartheta s\ i\ (s\ i) = \vartheta\ i$
$\wedge\ \vartheta s\ i\ (s\ (Suc\ i)) = \vartheta\ (Suc\ i))$
$(\lambda i.\ \textit{is-desc-fthread}\ (\vartheta s\ i)\ p\ (s\ i)\ (s\ (Suc\ i))$
$\wedge\ \vartheta s\ i\ (s\ i) = \vartheta\ i$
$\wedge\ \vartheta s\ i\ (s\ (Suc\ i)) = \vartheta\ (Suc\ i))$
$\langle proof \rangle$

lemma *INF-drop-prefix*:
$(\exists_\infty i{::}nat.\ i > n \wedge P\ i) = (\exists_\infty i.\ P\ i)$
$\langle proof \rangle$

lemma *contract-keeps-threads*:
assumes *inc*[*simp*]: *increasing s*
shows $(\exists \vartheta.\ \textit{is-desc-thread}\ \vartheta\ p)$
$\longleftrightarrow (\exists \vartheta.\ \textit{is-desc-thread}\ \vartheta\ (contract\ s\ p))$
(**is** $?A \longleftrightarrow ?B$)
$\langle proof \rangle$

lemma *repeated-edge*:
assumes $\bigwedge i.\ i > n \Longrightarrow dsc\ (snd\ (p\ i))\ k\ k$
shows *is-desc-thread* $(\lambda i.\ k)\ p$
$\langle proof \rangle$

lemma *fin-from-inf*:
assumes *is-thread* $n\ \vartheta\ p$
assumes $n < i$
assumes $i < j$
shows *is-fthread* $\vartheta\ p\ i\ j$
$\langle proof \rangle$

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 $set2pair\ \{x,\ y\} = (x,\ y)$
  $\langle proof \rangle$

definition
  $set2list = inv\ set$

lemma *finite-set2list*:
  assumes *finite S*
  shows $set\ (set2list\ S) = S$
  $\langle proof \rangle$

corollary *RamseyNatpairs*:
  fixes $S :: {}'a\ set$
    and $f :: nat \times nat \Rightarrow {}'a$

  assumes *finite S*
  and *inS*: $\bigwedge x\ y.\ x < y \Longrightarrow f\ (x,\ y) \in S$

  obtains $T :: nat\ set$ **and** $s :: {}'a$
  where *infinite T*
    and $s \in S$
    and $\bigwedge x\ y.\ [\![x \in T;\ y \in T;\ x < y]\!] \Longrightarrow f\ (x,\ y) = s$
$\langle proof \rangle$

5.5 Main Result

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

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*⟩

lemma *non-acc-has-idseq*:
assumes $\neg$ *accp R x*
shows $\exists$ *s*. *idseq R s x*
⟨*proof*⟩

types $('a,\ 'q)$ *cdesc* =
$('q \Rightarrow bool) \times ('q \Rightarrow 'a) \times ('q \Rightarrow 'a)$

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

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

lemma *some-rd*:
assumes *mk-rel rds x y*
shows $\exists$ *rd*$\in$*set rds*. *in-cdesc rd x y*
⟨*proof*⟩

lemma *ex-cs*:
assumes *idseq*: *idseq* (*mk-rel rds*) *s x*
shows $\exists$ *cs*. $\forall$ *i*. *cs i* $\in$ *set rds* $\wedge$ *in-cdesc* (*cs i*) (*s* (*Suc i*)) (*s i*)
⟨*proof*⟩

types *'a measures* = *nat* $\Rightarrow$ *'a* $\Rightarrow$ *nat*

fun *stepP* :: $('a,\ 'q)$ *cdesc* $\Rightarrow$ $('a,\ 'q)$ *cdesc* $\Rightarrow$
$('a \Rightarrow nat) \Rightarrow ('a \Rightarrow nat) \Rightarrow (nat \Rightarrow nat \Rightarrow bool) \Rightarrow bool$
where
stepP $(\Gamma 1, r1, l1)$ $(\Gamma 2, r2, l2)$ *m1 m2 R*
$= (\forall\, q_1\ q_2.\ \Gamma 1\ q_1 \wedge \Gamma 2\ q_2 \wedge r1\ q_1 = l2\ q_2$
$\longrightarrow R\ (m2\ (l2\ q_2))\ ((m1\ (l1\ q_1))))$

definition
decr :: $('a,\ 'q)$ *cdesc* $\Rightarrow$ $('a,\ 'q)$ *cdesc* $\Rightarrow$
$('a \Rightarrow nat) \Rightarrow ('a \Rightarrow nat) \Rightarrow bool$
where
$decr\ c1\ c2\ m1\ m2 = stepP\ c1\ c2\ m1\ m2\ (op <)$

definition
decreq :: $('a,\ 'q)$ *cdesc* $\Rightarrow$ $('a,\ 'q)$ *cdesc* $\Rightarrow$
$('a \Rightarrow nat) \Rightarrow ('a \Rightarrow nat) \Rightarrow bool$
where
$decreq\ c1\ c2\ m1\ m2 = stepP\ c1\ c2\ m1\ m2\ (op \leq)$

definition
no-step :: $('a,\ 'q)$ *cdesc* $\Rightarrow$ $('a,\ 'q)$ *cdesc* $\Rightarrow bool$
where
$no\text{-}step\ c1\ c2 = stepP\ c1\ c2\ (\lambda x.\ 0)\ (\lambda x.\ 0)\ (\lambda x\ y.\ False)$

lemma *decr-in-cdesc*:
assumes *in-cdesc RD1 y x*
assumes *in-cdesc RD2 z y*
assumes *decr RD1 RD2 m1 m2*
shows $m2\ y < m1\ x$
⟨*proof*⟩

lemma *decreq-in-cdesc*:
assumes *in-cdesc RD1 y x*
assumes *in-cdesc RD2 z y*
assumes *decreq RD1 RD2 m1 m2*
shows $m2\ y \leq m1\ x$
⟨*proof*⟩

lemma *no-inf-desc-nat-sequence*:
fixes $s :: nat \Rightarrow nat$
assumes *leq*: $\bigwedge i.\ n \leq i \Longrightarrow s\ (Suc\ i) \leq s\ i$
assumes *less*: $\exists_\infty i.\ s\ (Suc\ i) < s\ i$
shows *False*
⟨*proof*⟩

definition
approx :: *nat scg* $\Rightarrow$ $('a,\ 'q)$ *cdesc* $\Rightarrow$ $('a,\ 'q)$ *cdesc*
$\Rightarrow$ *'a measures* $\Rightarrow$ *'a measures* $\Rightarrow$ *bool*
where

$approx\ G\ C\ C'\ M\ M'$
$= (\forall\, i\, j.\ (dsc\ G\ i\ j \longrightarrow decr\ C\ C'\ (M\ i)\ (M'\ j))$
$\wedge(eq\ G\ i\ j \longrightarrow decreq\ C\ C'\ (M\ i)\ (M'\ j)))$

lemma *approx-empty*:
$approx\ (Graph\ \{\})\ c1\ c2\ ms1\ ms2$
$\langle proof\rangle$

lemma *approx-less*:
assumes $stepP\ c1\ c2\ (ms1\ i)\ (ms2\ j)\ (op <)$
assumes $approx\ (Graph\ Es)\ c1\ c2\ ms1\ ms2$
shows $approx\ (Graph\ (insert\ (i, \downarrow, j)\ Es))\ c1\ c2\ ms1\ ms2$
$\langle proof\rangle$

lemma *approx-leq*:
assumes $stepP\ c1\ c2\ (ms1\ i)\ (ms2\ j)\ (op \leq)$
assumes $approx\ (Graph\ Es)\ c1\ c2\ ms1\ ms2$
shows $approx\ (Graph\ (insert\ (i, \Downarrow, j)\ Es))\ c1\ c2\ ms1\ ms2$
$\langle proof\rangle$

lemma $approx\ (Graph\ \{(1, \downarrow, 2),(2, \Downarrow, 3)\})\ c1\ c2\ ms1\ ms2$
$\langle proof\rangle$

lemma *no-stepI*:
$stepP\ c1\ c2\ m1\ m2\ (\lambda x\ y.\ False)$
$\Longrightarrow no\text{-}step\ c1\ c2$
$\langle proof\rangle$

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

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

⟨*proof*⟩

lemma *all-less-zero*: $\forall\, n<(0::nat).\ P\ n$
⟨*proof*⟩

lemma *all-less-Suc*:
assumes *Pk*: $P\ k$
assumes *Pn*: $\forall\, n<k.\ P\ n$
shows $\forall\, n<Suc\ k.\ P\ n$
⟨*proof*⟩

lemma *step-witness*:
assumes *in-cdesc RD1 y x*
assumes *in-cdesc RD2 z y*
shows ¬ *no-step RD1 RD2*
⟨*proof*⟩

theorem *SCT-on-relations*:
assumes *R*: $R = mk\text{-}rel\ RDs$
assumes *sound*: *sound-int* $\mathcal{A}$ *RDs M*
assumes *SCT* $\mathcal{A}$
shows $\forall\, x.\ accp\ R\ x$
⟨*proof*⟩

end

7 Implemtation of the SCT criterion

theory *Implementation*
imports *Correctness*
begin

fun *edges-match* :: $('n \times 'e \times 'n) \times ('n \times 'e \times 'n) \Rightarrow bool$
where
edges-match $((n,\ e,\ m),\ (n',e',m')) = (m = n')$

fun *connect-edges* ::
$('n \times ('e::times) \times 'n) \times ('n \times 'e \times 'n)$
$\Rightarrow ('n \times 'e \times 'n)$
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* ‘ $\{\ x \in G{\times}H.$ *edges-match* $x\ \}$)
⟨*proof*⟩

lemma *mk-tcl-finite-terminates*:
  fixes $A :: 'a\ acg$
  assumes *fA*: *finite-acg A*
  shows *mk-tcl-dom* (A, A)
$\langle proof \rangle$

lemma *mk-tcl-finite-tcl*:
  fixes $A :: 'a\ acg$
  assumes *fA*: *finite-acg A*
  shows *mk-tcl A A* $=$ *tcl A*
  $\langle proof \rangle$

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

lemma *SCT′-exec*:
  assumes *fin*: *finite-acg A*
  shows *SCT′ A* $=$ *test-SCT A*
  $\langle proof \rangle$

code-modulename *SML*
  Implementation Graphs

lemma [*code func*]:
  $(G::('a::eq,\ 'b::eq)\ graph) \leq H \longleftrightarrow dest\text{-}graph\ G \subseteq dest\text{-}graph\ H$
  $(G::('a::eq,\ 'b::eq)\ graph) < H \longleftrightarrow dest\text{-}graph\ G \subset dest\text{-}graph\ H$
  $\langle proof \rangle$

lemma [*code func*]:
  $(G::('a::eq,\ 'b::eq)\ graph) + H = Graph\ (dest\text{-}graph\ G \cup dest\text{-}graph\ H)$
  $\langle proof \rangle$

lemma [*code func*]:
  $(G::('a::eq,\ 'b::\{eq,\ times\})\ graph) * H = grcomp\ G\ H$
  $\langle proof \rangle$

lemma *SCT′-empty*: *SCT′* (*Graph* $\{\}$)
  $\langle proof \rangle$

7.1 Witness checking

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

lemma *no-bad-graphs-ucl*:
  assumes $A \leq B$
  assumes *no-bad-graphs* B
  shows *no-bad-graphs* A
  ⟨*proof*⟩

lemma *SCT′-witness*:
  assumes *a*: *test-SCT-witness* A T
  shows *SCT′* A
⟨*proof*⟩

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

export-code *test-SCT* **in** *SML*

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*:
  $\{x \in \{\}.\ P\ x\} = \{\}$
  $\{x \in$ *insert* y *ys*$.\ P\ x\} = ($*if* $P\ y$ *then insert* y $\{x \in ys.\ P\ x\}$ *else* $\{x \in ys.\ P\ x\})$
  ⟨*proof*⟩

lemma *setbcomp-cong*:
$A = B \Longrightarrow (\bigwedge x.\ P\ x = Q\ x) \Longrightarrow \{x{\in}A.\ P\ x\} = \{x{\in}B.\ Q\ x\}$
$\langle proof \rangle$

lemma *cartprod-simps*:
$\{\} \times A = \{\}$
$insert\ a\ A \times B = Pair\ a\ `\ B \cup (A \times B)$
$\langle proof \rangle$

lemma *image-simps*:
$fu\ `\ \{\} = \{\}$
$fu\ `\ insert\ a\ A = insert\ (fu\ a)\ (fu\ `\ A)$
$\langle proof \rangle$

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

lemma *subset-simps*:
$\{\} \subseteq B$
$insert\ a\ A \subseteq B \equiv a \in B \wedge A \subseteq B$
$\langle proof \rangle$

lemma *element-simps*:
$x \in \{\} \equiv False$
$x \in insert\ a\ A \equiv x = a \vee x \in A$
$\langle proof \rangle$

lemma *set-eq-simp*:
$A = B \longleftrightarrow A \subseteq B \wedge B \subseteq A$ $\langle proof \rangle$

lemma *ball-simps*:
$\forall x{\in}\{\}.\ P\ x \equiv True$
$(\forall x{\in}insert\ a\ A.\ P\ x) \equiv P\ a \wedge (\forall x{\in}A.\ P\ x)$
$\langle proof \rangle$

lemma *bex-simps*:
$\exists x{\in}\{\}.\ P\ x \equiv False$
$(\exists x{\in}insert\ a\ A.\ P\ x) \equiv P\ a \vee (\exists x{\in}A.\ P\ x)$
$\langle proof \rangle$

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*:
$\downarrow * x = \downarrow$

$\Downarrow * x = x$
⟨*proof*⟩

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 $A \Longrightarrow$ *test-SCT* $A \Longrightarrow$ *SCT* A
⟨*proof*⟩

end

9 Examples for Size-Change Termination

theory *Examples*
imports *Size-Change-Termination*
begin

function f :: $nat \Rightarrow nat \Rightarrow nat$
where
$f\ n\ 0 = n$
$|\ f\ 0\ (Suc\ m) = f\ (Suc\ m)\ m$
$|\ f\ (Suc\ n)\ (Suc\ m) = f\ m\ n$
$\langle proof \rangle$

termination
$\langle proof \rangle$

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

termination
$\langle proof \rangle$

function foo :: $bool \Rightarrow nat \Rightarrow nat \Rightarrow nat$
where
$foo\ True\ (Suc\ n)\ m = foo\ True\ n\ (Suc\ m)$
$|\ foo\ True\ 0\ m = foo\ False\ 0\ m$
$|\ foo\ False\ n\ (Suc\ m) = foo\ False\ (Suc\ n)\ m$
$|\ foo\ False\ n\ 0 = n$
$\langle proof \rangle$

termination
$\langle proof \rangle$

function ($sequential$)
bar :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow nat$
where
$bar\ 0\ (Suc\ n)\ m = bar\ m\ m\ m$
$|\ bar\ k\ n\ m = 0$
$\langle proof \rangle$

termination
$\langle proof \rangle$

end