rustabelle

By sebastian

November 4, 2015

Contents

theory *Rustabelle*
imports *Main* ~~/*src*/*HOL*/*Library*/*While-Combinator*
begin

definition *loop* :: ('*state* ⇒ '*state* × *bool*) ⇒ '*state* ⇒ '*state* **where**
loop l s = (*let* (*s*,*s*',*c*) = *while* (λ(*s*,*s*',*c*). *c*) (λ(*s*,*s*',*c*). (*s*',(*l s*'))) (*s*,(*l s*)) *in s*')

lemma *loop-rule*:
assumes *P s*
assumes ⋀*s s*'. *P s* ⟹ *l s* = (*s*',*True*) ⟹ *P s*' ⋀*s s*'. *P s* ⟹ *l s* = (*s*',*False*) ⟹ *Q s*'
assumes *wf r* ⋀*s s*'. *P s* ⟹ *l s* = (*s*',*True*) ⟹ (*s*', *s*) ∈ *r*
shows *Q* (*loop l s*)
proof−
let *?r*' = {((*s*$_1$, *l s*$_1$), (*s*$_2$, *l s*$_2$)) | *s*$_1$ *s*$_2$. (*s*$_1$,*s*$_2$) ∈ *r*}
have *wf ?r*'
by (*rule compat-wf*[**where** *f*=*fst*, *OF* - *assms*(*4*)]) (*auto simp*: *compat-def*)
show *?thesis*
unfolding *loop-def*
by (*rule while-rule*[**where** *P*=λ(*s*,*s*'). *P s* ∧ *s*' = *l s* **and** *r*=*?r*', *OF* - - - ‹*wf ?r*'›]) (*auto simp*: *assms*)
qed

type-synonym *u32* = *nat*

abbreviation *core-iter-I-IntoIterator-into-iter* ≡ *id*

datatype *core-ops-Range* = *core-ops-Range u32 u32*
abbreviation *core-iter-ops-Range-A--Iterator-next r* ≡ *case r of core-ops-Range l r* ⇒ (*if l* < *r then Some l else None*, *core-ops-Range* (*l*+*1*) *r*)

type-synonym *core-option-Option* = *u32 option*
abbreviation *core-option-Option-None* ≡ *None*
abbreviation *core-option-Option-Some* ≡ *Some*

end
theory *lib*
imports *../../Rustabelle*
begin

definition *examples-fac-16* **where** *examples-fac-16 res* = (λ(*res*, *iter*). *let t-8* =
core-iter-ops-Range-A--Iterator-next in
let t-10 = *iter in*
let t-9 = *t-10 in*
let (*t-7*, *t-9*) = (*t-8 t-9*) *in*
case t-7 of core-option-Option-None => ((*res*, *iter*), *False*) | *core-option-Option-Some*
- => *let i* = (*case t-7 of core-option-Option-Some x* => *x*) *in*
let t-12 = *i in*
let res = (*res* ∗ *t-12*) *in*
let t-10 = *t-9 in*
let iter = *t-10 in*
((*res*, *iter*), *True*))

definition *examples-fac* :: *u32* => *u32* **where**
examples-fac n = (*let n* = *n in*
let res = *1 in*
let t-3 = *core-iter-I-IntoIterator-into-iter in*
let t-6 = *n in*
let t-5 = (*t-6* + *1*) *in*
let t-4 = (*core-ops-Range 2 t-5*) *in*
let (*t-2*) = (*t-3 t-4*) *in*
let iter = *t-2 in*
let (*res*, *iter*) = *loop* (*examples-fac-16 res*) (*res*, *iter*) *in*
let t-14 = *res in*
let ret = *t-14 in*
ret)

end
theory *examples*
imports *../export/examples/lib Binomial*
begin

lemma *fac*: *examples-fac* (*n*::*u32*) = *fact n*
proof−
show *?thesis*
unfolding *examples-fac-def*
apply *auto*
apply (*rule loop-rule*[**where** *P*=λ*s*. *case s of* (*res*,*core-ops-Range* (*Suc l*) *r*) ⇒
res = *fact l* ∧ (*n* = *0* ∧ *l* = *1* ∨ *l* < *r*) ∧ *r* = *n*+*1* | - ⇒ *False*])
unfolding *examples-fac-16-def*
apply *auto*[*1*]
apply (*auto simp add*: *le-less-Suc-eq split*:*prod.splits core-ops-Range.splits split-if-asm*

option.splits nat.splits)[*2*]
apply (*rule wf-measure*[*of* $\lambda s.$ *case s of* (-,*core-ops-Range l r*) $\Rightarrow$ $r-l$])
by (*auto simp add*: *le-less-Suc-eq split*:*prod.splits core-ops-Range.splits split-if-asm option.splits nat.splits*)
qed

end