Generating Cases from Labeled Subgoals

Lars Noschinski

March 17, 2025

Contents

Abstract

Isabelle/Isar provides *named cases* to structure proofs. This article contains an implementation of a proof method `casify`, which can be used to easily extend proof tools with support for named cases. Such a proof tool must produce labeled subgoals, which are then interpreted by `casify`.

As examples, this work contains verification condition generators producing named cases for three languages: The Hoare language from `HOL/Library`, a monadic language for computations with failure (inspired by the AutoCorres tool), and a language of conditional expressions. These VCGs are demonstrated by a number of example programs.

theory *Case-Labeling*
imports *Main*
keywords *print-nested-cases* :: *diag*
begin

1 Labeling Subgoals

context begin
qualified type-synonym *prg-ctxt-var* = *unit*
qualified type-synonym *prg-ctxt* = *string* × *nat* × *prg-ctxt-var list*

Embed variables in terms

qualified definition *VAR* :: *'v* ⇒ *prg-ctxt-var* **where**
VAR - = ()

Labeling of a subgoal

qualified definition *VC* :: *prg-ctxt list* ⇒ *'a* ⇒ *'a* **where**
VC ct P ≡ *P*

Computing the statement numbers and context

qualified definition *CTXT* :: *nat* ⇒ *prg-ctxt list* ⇒ *nat* ⇒ *'a* ⇒ *'a* **where**
CTXT inp ct outp P ≡ *P*

Labeling of a term binding or assumption

qualified definition *BIND* :: *string* ⇒ *nat* ⇒ *'a* ⇒ *'a* **where**
BIND name inp P ≡ *P*

Hierarchy labeling

qualified definition *HIER* :: *prg-ctxt list* ⇒ *'a* ⇒ *'a* **where**
HIER ct P ≡ *P*

Split Labeling. This is used as an assumption

qualified definition *SPLIT* :: *'a* ⇒ *'a* ⇒ *bool* **where**
SPLIT v w ≡ *v* = *w*

Disambiguation Labeling. This is used as an assumption

qualified definition *DISAMBIG* :: *nat* ⇒ *bool* **where**
DISAMBIG n ≡ *True*

lemmas *LABEL-simps* = *BIND-def CTXT-def HIER-def SPLIT-def VC-def*

lemma *Initial-Label*: *CTXT 0* [] *outp P* ⟹ *P*
⟨*proof*⟩

lemma
BIND-I: *P* ⟹ *BIND name inp P* **and**
BIND-D: *BIND name inp P* ⟹ *P* **and**

VC-I: $P \Longrightarrow VC\ ct\ P$
⟨*proof*⟩

lemma *DISAMBIG-I*: $(DISAMBIG\ n \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemma *DISAMBIG-E*: $(DISAMBIG\ n \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

Lemmas for the tuple postprocessing

lemma *SPLIT-reflection*: $SPLIT\ x\ y \Longrightarrow (x \equiv y)$
⟨*proof*⟩

lemma *rev-SPLIT-reflection*: $(x \equiv y) \Longrightarrow SPLIT\ x\ y$
⟨*proof*⟩

lemma *SPLIT-sym*: $SPLIT\ x\ y \Longrightarrow SPLIT\ y\ x$
⟨*proof*⟩

lemma *SPLIT-thin-refl*: $[\![SPLIT\ x\ x;\ PROP\ W]\!] \Longrightarrow PROP\ W$ ⟨*proof*⟩

lemma *SPLIT-subst*: $[\![SPLIT\ x\ y;\ P\ x]\!] \Longrightarrow P\ y$
⟨*proof*⟩

lemma *SPLIT-prodE*:
assumes $SPLIT\ (x1,\ y1)\ (x2,\ y2)$
obtains *SPLIT x1 x2 SPLIT y1 y2*
⟨*proof*⟩

end

The labeling constants were qualified to not interfere with any other theory. The following locale allows using a nice syntax in other theories

locale *Labeling-Syntax* **begin**
abbreviation VAR **where** $VAR \equiv Case\text{-}Labeling.VAR$
abbreviation $VC\ (‹V⟨(2\text{-},\text{-}:/\ \text{-})⟩›)$ **where** $VC\ bl\ ct \equiv Case\text{-}Labeling.VC\ (bl\ \#\ ct)$
abbreviation $CTXT\ (‹C⟨(2\text{-},\text{-},\text{-}:/\ \text{-})⟩›)$ **where** $CTXT \equiv Case\text{-}Labeling.CTXT$
abbreviation $BIND\ (‹B⟨(2\text{-},\text{-}:/\ \text{-})⟩›)$ **where** $BIND \equiv Case\text{-}Labeling.BIND$
abbreviation $HIER\ (‹H⟨(2\text{-}:/\ \text{-})⟩›)$ **where** $HIER \equiv Case\text{-}Labeling.HIER$
abbreviation $SPLIT$ **where** $SPLIT \equiv Case\text{-}Labeling.SPLIT$
end

Lemmas for converting terms from *Suc/0* notation to numerals

lemma *Suc-numerals-conv*:
$Suc\ 0 = Numeral1$
$Suc\ (numeral\ n) = numeral\ (n + num.One)$
⟨*proof*⟩

lemmas *Suc-numeral-simps = Suc-numerals-conv add-num-simps*

2 Casify

Introduces a command **print-nested-cases**. This is similar to **print-cases**, but shows also the nested cases.

⟨*ML*⟩
Introduces the proof method.

⟨*ML*⟩

end

3 Examples

3.1 A labeling VCG for a monadic language

theory *Monadic-Language*
imports
Complex-Main
../Case-Labeling
HOL−Eisbach.Eisbach
begin

⟨*ML*⟩

This language is inspired by the languages used in AutoCorres [1]

consts *bind* :: *'a option* ⇒ (*'a* ⇒ *'b option*) ⇒ *'b option* (**infixr** ‹|>>› *4*)
consts *return* :: *'a* ⇒ *'a option*
consts *while* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *'a option*) ⇒ (*'a* ⇒ *'a option*)
consts *valid* :: *bool* ⇒ *'a option* ⇒ (*'a* ⇒ *bool*) ⇒ *bool*

named-theorems *vcg*
named-theorems *vcg-comb*

⟨*ML*⟩

axiomatization where
return[*vcg*]: *valid* (*Q x*) (*return x*) *Q* **and**
bind[*vcg*]: ⟦⋀*x*. *valid* (*R x*) (*c2 x*) *Q*; *valid P c1 R*⟧ ⟹ *valid P* (*bind c1 c2*) *Q* **and**
while[*vcg*]: ⋀*c*. ⟦⋀*x*. *valid* (*I x* ∧ *b x*) (*c x*) *I*; ⋀*x*. *I x* ∧ ¬*b x* ⟹ *Q x*⟧ ⟹ *valid* (*I x*) (*while b I c x*) *Q* **and**
cond[*vcg*]: ⋀*b c1 c2*. *valid P1 c1 Q* ⟹ *valid P2 c2 Q* ⟹ *valid* (*if b then P1 else P2*) (*if b then c1 else c2*) *Q* **and**

case-prod[*vcg*]: $\bigwedge P.\ [\![\bigwedge x\ y.\ v = (x,y) \Longrightarrow \mathit{valid}\ (P\ x\ y)\ (B\ x\ y)\ Q]\!]$
$\Longrightarrow \mathit{valid}\ (\mathit{case}\ v\ \mathit{of}\ (x,y) \Rightarrow P\ x\ y)\ (\mathit{case}\ v\ \mathit{of}\ (x,y) \Rightarrow B\ x\ y)\ Q$ **and**
conseq[*vcg-comb*]: $[\![\mathit{valid}\ P'\ c\ Q;\ P \Longrightarrow P']\!] \Longrightarrow \mathit{valid}\ P\ c\ Q$

Labeled rules

named-theorems *vcg-l*
named-theorems *vcg-l-comb*
named-theorems *vcg-elim*

⟨*ML*⟩

method *vcg-l'* = (*vcg-l*; (*elim vcg-elim*)*?*)

context begin
interpretation *Labeling-Syntax* ⟨*proof*⟩

lemma *L-return*[*vcg-l*]: *CTXT inp ct* (*Suc inp*) (*valid* (*P x*) (*return x*) *P*)
⟨*proof*⟩

lemma *L-bind*[*vcg-l*]:
assumes $\bigwedge x.$ *CTXT* (*Suc outp'*) ((''*bind*'',*outp'*, [*VAR x*]) # *ct*) *outp* (*valid* (*R x*) (*c2 x*) *Q*)
assumes *CTXT inp ct outp'* (*valid P c1 R*)
shows *CTXT inp ct outp* (*valid P* (*bind c1 c2*) *Q*)
⟨*proof*⟩

lemma *L-while*[*vcg-l*]:
fixes *inp ct* **defines** *ct'* ≡ $\lambda x.$ (''*while*'', *inp*, [*VAR x*]) # *ct*
assumes $\bigwedge x.$ *CTXT* (*Suc inp*) (*ct' x*) *outp'*
(*valid* (*BIND* ''*inv-pre*'' *inp* (*I x*) ∧ *BIND* ''*lcond*'' *inp* (*b x*)) (*c x*) (λ*x*. *BIND* ''*inv-post*'' *inp* (*I x*)))
assumes $\bigwedge x.$ *B*⟨''*inv-pre*'',*inp*: *I x*⟩ ∧ *B*⟨''*lcond*'',*inp*: ¬*b x*⟩ ⟹ *VC* (''*post*'',*outp'*, []) (*ct' x*) (*P x*)
shows *CTXT inp ct* (*Suc outp'*) (*valid* (*I x*) (*while b I c x*) *P*)
⟨*proof*⟩

lemma *L-cond*[*vcg-l*]:
fixes *inp ct* **defines** *ct'* ≡ (''*if*'',*inp*,[]) # *ct*
assumes *C*⟨*Suc inp*, (''*then*'',*inp*,[]) # *ct'*,*outp*: *valid P1 c1 Q*⟩
assumes *C*⟨*Suc outp*, (''*else*'',*outp*,[]) # *ct'*,*outp'*: *valid P2 c2 Q*⟩
shows *C*⟨*inp*,*ct*,*outp'*: *valid* (*if B*⟨''*cond*'',*inp*: *b*⟩ *then B*⟨''*then*'',*inp*: *P1*⟩ *else B*⟨''*else*'',*inp*: *P2*⟩) (*if b then c1 else c2*) *Q*⟩
⟨*proof*⟩

lemma *L-case-prod*[*vcg-l*]:
assumes $\bigwedge x\ y.\ v = (x,y) \Longrightarrow$ *CTXT inp ct outp* (*valid* (*P x y*) (*B x y*) *Q*)
shows *CTXT inp ct outp* (*valid* (*case v of* (*x*,*y*) ⇒ *P x y*) (*case v of* (*x*,*y*) ⇒ *B x y*) *Q*)
⟨*proof*⟩

lemma *L-conseq*[*vcg-l-comb*]:
assumes *CTXT* (*Suc inp*) *ct outp* (*valid P′ c Q*)
assumes *P* ⟹ *VC* (*''conseq''*,*inp*,[]) *ct P′*
shows *CTXT inp ct outp* (*valid P c Q*)
⟨*proof*⟩

lemma *L-assm-conjE*[*vcg-elim*]:
assumes *BIND name inp* (*P* ∧ *Q*) **obtains** *BIND name inp P BIND name inp Q*
⟨*proof*⟩

declare *conjE*[*vcg-elim*]

end

lemma *dvd-div*:
fixes *a b c* :: *int*
assumes *a dvd b c dvd b coprime a c*
shows *a dvd* (*b div c*)
⟨*proof*⟩

lemma *divides*:
valid
(*0* < (*a* :: *int*))
(
return a
|>> (λ*n*.
while
(λ*n*. *even n*)
(λ*n*. *0* < *n* ∧ *n dvd a* ∧ (∀ *m*. *odd m* ∧ *m dvd a* ⟶ *m dvd n*))
(λ*n*. *return* (*n div 2*))
n
)
)
(λ*r*. *odd r* ∧ *r dvd a* ∧ (∀ *m*. *odd m* ∧ *m dvd a* ⟶ *m* ≤ *r*))

⟨*proof*⟩

lemma *L-divides*:
valid
(*0* < (*a* :: *int*))
(
return a
|>> (λ*n*.
while
(λ*n*. *even n*)

```
      (λn. 0 < n ∧ n dvd a ∧ (∀ m. odd m ∧ m dvd a ⟶ m dvd n))
      (λn. return (n div 2))
      n
  )
 )
 (λr. odd r ∧ r dvd a ∧ (∀ m. odd m ∧ m dvd a ⟶ m ≤ r))

 ⟨proof⟩
```

lemma *add*:

```
valid
 True
 (
  while
   — COND: (λ(r,j). j < (b :: nat))
   — INV: (λ(r,j). j ≤ b ∧ r = a + j)
   — BODY: (λ(r,j). return (r + 1, j + 1))
   — START: (a,0)
  |>> (λ(r,-). return r)
 )
 (λr. r = a + b)

 ⟨proof⟩
```

lemma *mult*:

```
valid
 True
 (
  while
   — COND: (λ(r,i). i < (a :: nat))
   — INV: (λ(r,i). i ≤ a ∧ r = i * b)
   — BODY: (λ(r,i).
     while
      — COND: (λ(r,j). j < b)
      — INV: (λ(r,j). i < a ∧ j ≤ b ∧ r = i * b + j)
      — BODY: (λ(r,j). return (r + 1, j + 1))
      — START: (r,0)
     |>> (λ(r,-). return (r, i + 1))
   )
   — START: (0,0)
  |>> (λ(r,-). return r)
 )
 (λr. r = a * b)

 ⟨proof⟩
```

4 Labeled

lemma *L-mult*:
valid
  True
  (
    while
      — COND: ($\lambda(r,i).\ i < (a :: nat)$)
      — INV: ($\lambda(r,i).\ i \leq a \wedge r = i * b$)
      — BODY: ($\lambda(r,i)$.
        while
          — COND: ($\lambda(r,j).\ j < b$)
          — INV: ($\lambda(r,j).\ i < a \wedge j \leq b \wedge r = i * b + j$)
          — BODY: ($\lambda(r,j).$ *return* $(r + 1, j + 1)$)
          — START: $(r,0)$
        |>> ($\lambda(r,\text{-}).$ *return* $(r,\ i + 1)$)
)
      — START: $(0,0)$
    |>> ($\lambda(r,\text{-}).$ *return* r)
)
  ($\lambda r.\ r = a * b$)

  ⟨*proof*⟩

lemma *L-paths*:
valid
  (*path* $\neq$ [])
  (*while*
      — COND: ($\lambda(p,r).\ p \neq []$)
      — INV: ($\lambda(p,r).$ *distinct* $r \wedge$ *hd* $(r$ @ $p) =$ *hd path* $\wedge$ *last* $(r$ @ $p) =$ *last path*)
      — BODY: ($\lambda(p,r)$.
        return (*hd p*)
        |>> ($\lambda x.$
          if $(r \neq [] \wedge x =$ *hd* $r)$
          then return []
          else (*if* $x \in$ *set* r
            then return (*takeWhile* $(\lambda y.\ y \neq x)\ r$)
            else return (r))
        |>> ($\lambda r'.$ *return* (*tl* $p,\ r'$ @ $[x]$)
)
)
)
      — START: (*path*, [])
    |>> ($\lambda(\text{-},r).$ *return* r)
)
  ($\lambda r.$ *distinct* $r \wedge$ *hd* $r =$ *hd path* $\wedge$ *last* $r =$ *last path*)

  ⟨*proof*⟩

end

4.1 Decomposing Conditionals

theory *Conditionals*
imports
Complex-Main
../Case-Labeling
HOL−Eisbach.Eisbach
begin

context begin
interpretation *Labeling-Syntax* ⟨*proof*⟩

lemma *DC-conj*:
assumes $C\langle inp,ct,outp'\text{: } a\rangle$ $C\langle outp',ct,outp\text{: } b\rangle$
shows $C\langle inp,ct,outp\text{: } a \wedge b\rangle$
⟨*proof*⟩

lemma *DC-if*:
fixes *ct* **defines** $ct' \equiv \lambda pos\ name.\ (name,\ pos,[])\ \#\ ct$
assumes $H\langle ct'\ inp\ ''then''\text{: } a\rangle \Longrightarrow C\langle Suc\ inp,ct'\ inp\ ''then'',\ outp'\text{: } b\rangle$
assumes $H\langle ct'\ outp'\ ''else''\text{: } \neg a\rangle \Longrightarrow C\langle Suc\ outp',ct'\ outp'\ ''else'',\ outp\text{: } c\rangle$
shows $C\langle inp,ct,outp\text{: } if\ a\ then\ b\ else\ c\rangle$
⟨*proof*⟩

lemma *DC-final*:
assumes $V\langle(''g'',inp,[]),\ ct\text{: } a\rangle$
shows $C\langle inp,ct,Suc\ inp\text{: } a\rangle$
⟨*proof*⟩

end

method *vcg-dc* = (*intro DC-conj DC-if*; *rule DC-final*)

lemma
assumes *a*: *a*
and *d*: $b \Longrightarrow c \Longrightarrow d$
and *d'*: $b \Longrightarrow c \Longrightarrow d'$
and *e*: $b \Longrightarrow \neg c \Longrightarrow e$
and *f*: $\neg b \Longrightarrow f$
shows $a \wedge (if\ b\ then\ (if\ c\ then\ d \wedge d'\ else\ e)\ else\ f)$
⟨*proof*⟩

4.2 Protecting similar subgoals

The proof below fails if the `disambig_subgoals` option is omitted: all three subgoals have the same conclusion and can be discharged without using their assumptions. If the case `g` is solved first, it discharges instead the subgoal *a*

$\Longrightarrow b$, making the case **then** fail afterwards.

The `disambig_subgoals` options prevents this by inserting vacuous assumptions.

lemma
assumes *b*
shows (*if a then b else b*) $\wedge$ *b*
⟨*proof*⟩

4.3 Unnamed Cases

lemma
assumes $a \Longrightarrow b$ $\neg a \Longrightarrow c$ *d*
shows (*if a then b else c*) $\wedge$ *d*
⟨*proof*⟩

end
theory *Labeled-Hoare*
imports
../../Case-Labeling
HOL−Hoare.Hoare-Logic
begin

4.4 A labeling VCG for HOL/Hoare

context begin
interpretation *Labeling-Syntax* ⟨*proof*⟩

lemma *LSeqRule*:
assumes *C*⟨*IC*,*CT*,*OC1*: *Valid P c1 a1 Q*⟩
and *C*⟨*Suc OC1*,*CT*,*OC*: *Valid Q c2 a2 R*⟩
shows *C*⟨*IC*,*CT*,*OC*: *Valid P* (*Seq c1 c2*) (*Aseq a1 a2*) *R*⟩
⟨*proof*⟩

lemma *LSkipRule*:
assumes *V*⟨(*''weaken''*, *IC*, []),*CT*: $p \subseteq q$⟩
shows *C*⟨*IC*,*CT*,*IC*: *Valid p SKIP a q*⟩
⟨*proof*⟩

lemmas *LAbortRule* = *LSkipRule* — dummy version

lemma *LBasicRule*:
assumes *V*⟨(*''basic''*, *IC*, []),*CT*: $p \subseteq \{s.\ f\ s \in q\}$⟩
shows *C*⟨*IC*,*CT*,*IC*: *Valid p* (*Basic f*) *a q*⟩
⟨*proof*⟩

lemma *LCondRule*:
fixes *IC CT* **defines** $CT' \equiv (''cond'', IC, []) \#\ CT$
assumes *V*⟨(*''vc''*, *IC*, []),(*''cond''*, *IC*, []) # *CT*: $p \subseteq \{s.\ (s \in b \longrightarrow s \in w) \wedge (s \notin b \longrightarrow s \in w')\}$⟩

and $C\langle Suc\ IC,(''then'',\ IC,\ [])\ \#\ (''cond'',\ IC,\ [])\ \#\ CT,OC1\colon\ Valid\ w\ c1\ a1\ q\rangle$
and $C\langle Suc\ OC1,(''else'',\ Suc\ OC1,\ [])\ \#\ (''cond'',\ IC,\ [])\ \#\ CT,OC\colon\ Valid\ w'\ c2\ a2\ q\rangle$
shows $C\langle IC,CT,OC\colon\ Valid\ p\ (Cond\ b\ c1\ c2)\ (Acond\ a1\ a2)\ q\rangle$
⟨*proof*⟩

lemma *LWhileRule*:
fixes *IC CT* **defines** $CT' \equiv (''while'',\ IC,\ [])\ \#\ CT$
assumes $V\langle(''precondition'',\ IC,\ []),(''while'',\ IC,\ [])\ \#\ CT\colon\ p \subseteq i\rangle$
and $C\langle Suc\ IC,(''invariant'',\ Suc\ IC,\ [])\ \#\ (''while'',\ IC,\ [])\ \#\ CT,OC\colon\ Valid\ (i \cap b)\ c\ (A\ 0)\ i\rangle$
and $V\langle(''postcondition'',\ IC,\ []),(''while'',\ IC,\ [])\ \#\ CT\colon\ i \cap -\ b \subseteq q\rangle$
shows $C\langle IC,CT,OC\colon\ Valid\ p\ (While\ b\ c)\ (Awhile\ i\ v\ A)\ q\rangle$
⟨*proof*⟩

lemma *LABELs-to-prems*:
$(C\langle IC,\ CT,\ OC\colon\ True\rangle \Longrightarrow P) \Longrightarrow C\langle IC,\ CT,\ OC\colon\ P\rangle$
$(V\langle x,\ ct\colon\ True\rangle \Longrightarrow P) \Longrightarrow V\langle x,\ ct\colon\ P\rangle$
⟨*proof*⟩

lemma *LABELs-to-concl*:
$C\langle IC,\ CT,\ OC\colon\ True\rangle \Longrightarrow C\langle IC,\ CT,\ OC\colon\ P\rangle \Longrightarrow P$
$V\langle x,\ ct\colon\ True\rangle \Longrightarrow V\langle x,\ ct\colon\ P\rangle \Longrightarrow P$
⟨*proof*⟩

end

⟨*ML*⟩

end

theory *Labeled-Hoare-Examples*
imports
Labeled-Hoare
HOL−Hoare.Arith2
begin

4.4.1 Multiplication by successive addition

lemma *multiply-by-add*: *VARS m s a b*
$\{a{=}A \wedge b{=}B\}$
$m := 0;\ s := 0;$
WHILE $m{\neq}a$
INV $\{s{=}m{*}b \wedge a{=}A \wedge b{=}B\}$
DO $s := s{+}b;\ m := m{+}(1{::}nat)$ *OD*
$\{s = A{*}B\}$
⟨*proof*⟩

lemma *VARS M N P :: int*
$\{m{=}M \wedge n{=}N\}$
IF $M < 0$ *THEN* $M := -M$; $N := -N$ *ELSE SKIP FI*;
$P := 0$;
WHILE $0 < M$
INV $\{0 \leq M \wedge (\exists p.\ p = (\mathit{if}\ m{<}0\ \mathit{then}\ -m\ \mathit{else}\ m) \wedge p{*}N = m{*}n \wedge P = (p{-}M){*}N)\}$
DO $P := P{+}N$; $M := M - 1$ *OD*
$\{P = m{*}n\}$
⟨*proof*⟩

4.4.2 Euclid's algorithm for GCD

lemma *Euclid-GCD*: *VARS a b*
$\{0{<}A \wedge 0{<}B\}$
$a := A$; $b := B$;
WHILE $a \neq b$
INV $\{0{<}a \wedge 0{<}b \wedge \mathit{gcd}\ A\ B = \mathit{gcd}\ a\ b\}$
DO IF $a{<}b$ *THEN* $b := b{-}a$ *ELSE* $a := a{-}b$ *FI OD*
$\{a = \mathit{gcd}\ A\ B\}$
⟨*proof*⟩

4.4.3 Dijkstra's extension of Euclid's algorithm for simultaneous GCD and SCM

From E.W. Disjkstra. Selected Writings on Computing, p 98 (EWD474), where it is given without the invariant. Instead of defining scm explicitly we have used the theorem scm x y = x*y/gcd x y and avoided division by mupltiplying with gcd x y.

lemmas *distribs* =
diff-mult-distrib diff-mult-distrib2 add-mult-distrib add-mult-distrib2

lemma *gcd-scm*: *VARS a b x y*
$\{0{<}A \wedge 0{<}B \wedge a{=}A \wedge b{=}B \wedge x{=}B \wedge y{=}A\}$
WHILE $a \neq b$
INV $\{0{<}a \wedge 0{<}b \wedge \mathit{gcd}\ A\ B = \mathit{gcd}\ a\ b \wedge 2{*}A{*}B = a{*}x + b{*}y\}$
DO IF $a{<}b$ *THEN* $(b := b{-}a;\ x := x{+}y)$ *ELSE* $(a := a{-}b;\ y := y{+}x)$ *FI OD*
$\{a = \mathit{gcd}\ A\ B \wedge 2{*}A{*}B = a{*}(x{+}y)\}$
⟨*proof*⟩

4.4.4 Power by iterated squaring and multiplication

lemma *power-by-mult*: *VARS a b c*
$\{a{=}A \wedge b{=}B\}$
$c := (1{::}\mathit{nat})$;
WHILE $b \neq 0$
INV $\{A\hat{}B = c * a\hat{}b\}$
DO WHILE $b\ \mathit{mod}\ 2 = 0$

```
    INV {A^B = c * a^b}
    DO  a := a*a; b := b div 2 OD;
    c := c*a; b := b - 1
 OD
 {c = A^B}
⟨proof⟩
```

4.4.5 Factorial

```
lemma factorial: VARS a b
 {a=A}
 b := 1;
 WHILE a ≠ 0
 INV {fac A = b * fac a}
 DO b := b*a; a := a - 1 OD
 {b = fac A}
 ⟨proof⟩

lemma VARS i f
 {True}
 i := (1::nat); f := 1;
 WHILE i ≤ n INV {f = fac(i - 1) ∧ 1 ≤ i ∧ i ≤ n+1}
 DO f := f*i; i := i+1 OD
 {f = fac n}
⟨proof⟩
```

4.4.6 Quicksort

The 'partition' procedure for quicksort. 'A' is the array to be sorted (modelled as a list). Elements of A must be of class order to infer at the end that the elements between u and l are equal to pivot.

Ambiguity warnings of parser are due to := being used both for assignment and list update.

```
lemma Partition:
  fixes pivot
  defines leq ≡ λA i. ∀k. k<i ⟶ A!k ≤ pivot
  defines geq ≡ λA i. ∀k. i<k ∧ k<length A ⟶ pivot ≤ A!k
  shows
   VARS A u l
   {0 < length(A::('a::order)list)}
   l := 0; u := length A - Suc 0;
   WHILE l ≤ u
    INV {leq A l ∧ geq A u ∧ u<length A ∧ l≤length A}
    DO WHILE l < length A ∧ A!l ≤ pivot
        INV {leq A l ∧ geq A u ∧ u<length A ∧ l≤length A}
        DO l := l+1 OD;
       WHILE 0 < u ∧ pivot ≤ A!u
        INV {leq A l ∧ geq A u  ∧ u<length A ∧ l≤length A}
```

$$
\begin{array}{l}
\quad DO\ u := u - 1\ OD; \\
\quad IF\ l \leq u\ THEN\ A := A[l := A!u,\ u := A!l]\ ELSE\ SKIP\ FI \\
\ OD \\
\{leq\ A\ u \wedge (\forall k.\ u{<}k \wedge k{<}l \longrightarrow A!k = pivot) \wedge geq\ A\ l\} \\
\langle proof \rangle
\end{array}
$$

end

References

[1] D. Greenaway, J. Andronick, and G. Klein. Bridging the gap: Automatic verified abstraction of C. In *Interactive Theorem Proving*, Lecture Notes in Computer Science, pages 99–115. Springer, Jan. 2012.