

Generating Cases from Labeled Subgoals

Lars Noschinski

May 26, 2024

Contents

1	Labeling Subgoals	2
2	Casify	4
3	Examples	4
3.1	A labeling VCG for a monadic language	4
4	Labeled	8
4.1	Decomposing Conditionals	9
4.2	Protecting similar subgoals	9
4.3	Unnamed Cases	10
4.4	A labeling VCG for HOL/Hoare	10
4.4.1	Multiplication by successive addition	11
4.4.2	Euclid’s algorithm for GCD	12
4.4.3	Dijkstra’s extension of Euclid’s algorithm for simultaneous GCD and SCM	12
4.4.4	Power by iterated squaring and multiplication	12
4.4.5	Factorial	13
4.4.6	Quicksort	13

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: $P \implies VC \text{ ct } P$
<proof>

lemma *DISAMBIG-I*: $(DISAMBIG \ n \implies P) \implies P$
<proof>

lemma *DISAMBIG-E*: $(DISAMBIG \ n \implies P) \implies P$
<proof>

Lemmas for the tuple postprocessing

lemma *SPLIT-reflection*: $SPLIT \ x \ y \implies (x \equiv y)$
<proof>

lemma *rev-SPLIT-reflection*: $(x \equiv y) \implies SPLIT \ x \ y$
<proof>

lemma *SPLIT-sym*: $SPLIT \ x \ y \implies SPLIT \ y \ x$
<proof>

lemma *SPLIT-thin-refl*: $\llbracket SPLIT \ x \ x; PROP \ W \rrbracket \implies PROP \ W$ *<proof>*

lemma *SPLIT-subst*: $\llbracket SPLIT \ x \ y; P \ x \rrbracket \implies P \ y$
<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 $VAR \ \mathbf{where} \ VAR \equiv Case-Labeling.VAR$

abbreviation $VC \ (V \langle (2, \cdot, \cdot / \cdot) \rangle) \ \mathbf{where} \ VC \ bl \ ct \equiv Case-Labeling.VC \ (bl \ \# \ ct)$

abbreviation $CTXT \ (C \langle (2, \cdot, \cdot / \cdot) \rangle) \ \mathbf{where} \ CTXT \equiv Case-Labeling.CTXT$

abbreviation $BIND \ (B \langle (2, \cdot, \cdot / \cdot) \rangle) \ \mathbf{where} \ BIND \equiv Case-Labeling.BIND$

abbreviation $HIER \ (H \langle (2, \cdot / \cdot) \rangle) \ \mathbf{where} \ HIER \equiv Case-Labeling.HIER$

abbreviation $SPLIT \ \mathbf{where} \ SPLIT \equiv Case-Labeling.SPLIT$

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* \Rightarrow (*'a* \Rightarrow *'b option*) \Rightarrow *'b option* (**infixr** $|>>$ 4)

consts *return* :: *'a* \Rightarrow *'a option*

consts *while* :: (*'a* \Rightarrow *bool*) \Rightarrow (*'a* \Rightarrow *bool*) \Rightarrow (*'a* \Rightarrow *'a option*) \Rightarrow (*'a* \Rightarrow *'a option*)

consts *valid* :: *bool* \Rightarrow *'a option* \Rightarrow (*'a* \Rightarrow *bool*) \Rightarrow *bool*

named-theorems *vcg*

named-theorems *vcg-comb*

<ML>

axiomatization where

return[*vcg*]: *valid* (*Q x*) (*return x*) *Q* **and**

bind[*vcg*]: $\llbracket \bigwedge x. \text{valid } (R\ x) (c2\ x) Q; \text{valid } P\ c1\ R \rrbracket \Longrightarrow \text{valid } P\ (\text{bind } c1\ c2)\ Q$

and

while[*vcg*]: $\bigwedge c. \llbracket \bigwedge x. \text{valid } (I\ x \wedge b\ x) (c\ x) I; \bigwedge x. I\ x \wedge \neg b\ x \Longrightarrow Q\ x \rrbracket \Longrightarrow \text{valid } (I\ x) (\text{while } b\ I\ c\ x)\ Q$ **and**

cond[*vcg*]: $\bigwedge b\ c1\ c2. \text{valid } P1\ c1\ Q \Longrightarrow \text{valid } P2\ c2\ Q \Longrightarrow \text{valid } (\text{if } b\ \text{then } P1\ \text{else } P2) (\text{if } b\ \text{then } c1\ \text{else } c2)\ Q$ **and**

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

Labeled rules

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

$\langle ML \rangle$

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

context begin

interpretation *Labeling-Syntax* $\langle proof \rangle$

lemma *L-return*[*vcg-l*]: *CTXT inp ct (Suc inp) (valid (P x) (return x) P)*
 $\langle proof \rangle$

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)*
 $\langle proof \rangle$

lemma *L-while*[*vcg-l*]:
fixes *inp ct defines ct' $\equiv \lambda x. ('while'', inp, [VAR\ x]) \# ct$*
assumes $\bigwedge x. CTXT (Suc\ inp) (ct'\ x) outp'$
 $(valid (BIND\ ''inv\text{-}pre''\ inp\ (I\ x) \wedge BIND\ ''lcond''\ inp\ (b\ x)) (c\ x) (\lambda x. BIND$
 $''inv\text{-}post''\ inp\ (I\ x)))$
assumes $\bigwedge x. B(''inv\text{-}pre'', inp: I\ x) \wedge B(''lcond'', inp: \neg b\ x) \implies VC (''post'', outp'$
 $, []) (ct'\ x) (P\ x)$
shows *CTXT inp ct (Suc outp') (valid (I x) (while b I c x) P)*
 $\langle proof \rangle$

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

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

lemma *L-conseq*[*vcg-l-comb*]:
assumes *CTXT* (*Suc inp*) *ct outp* (*valid P' c Q*)
assumes $P \implies VC$ ("*conseq''*",*inp*,[]) *ct P'*
shows *CTXT inp ct outp* (*valid P c Q*)
<proof>

lemma *L-assm-conjE*[*vcg-elim*]:
assumes *BIND name inp* ($P \wedge Q$) **obtains** *BIND name inp P BIND name inp Q*
<proof>

declare *conjE*[*vcg-elim*]

end

lemma *dvd-div*:
fixes $a b c :: int$
assumes $a \text{ dvd } b \text{ } c \text{ dvd } b \text{ coprime } a \text{ } c$
shows $a \text{ dvd } (b \text{ div } c)$
<proof>

lemma *divides*:
valid
 $(0 < (a :: int))$
(
 return a
 |>> ($\lambda n.$
 while
 ($\lambda n. \text{even } n$)
 ($\lambda n. 0 < n \wedge n \text{ dvd } a \wedge (\forall m. \text{odd } m \wedge m \text{ dvd } a \longrightarrow m \text{ dvd } n)$)
 ($\lambda n. \text{return } (n \text{ div } 2)$)
 n
)
)
 $(\lambda r. \text{odd } r \wedge r \text{ dvd } a \wedge (\forall m. \text{odd } m \wedge m \text{ dvd } a \longrightarrow m \leq r))$
<proof>

lemma *L-divides*:
valid
 $(0 < (a :: int))$
(
 return a
 |>> ($\lambda n.$
 while
 ($\lambda n. \text{even } n$)
)
)
 $(\lambda r. \text{odd } r \wedge r \text{ dvd } a \wedge (\forall m. \text{odd } m \wedge m \text{ dvd } a \longrightarrow m \leq r))$

```

      (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: $(r, 0)$
 $|>> (\lambda(r,-). return (r, i + 1))$
)
 — START: $(0, 0)$
 $|>> (\lambda(r,-). return r)$
)
 $(\lambda r. r = a * b)$

<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: $(path, [])$
 $|>> (\lambda(-,r). return\ r)$
)
 $(\lambda r. distinct\ r \wedge hd\ r = hd\ path \wedge last\ r = last\ path)$

<proof>

end

4.1 Decomposing Conditionals

theory *Conditionals*

imports

Complex-Main

../Case-Labeling

HOL-Eisbach.Eisbach

begin

context begin

interpretation *Labeling-Syntax* $\langle proof \rangle$

lemma *DC-conj*:

assumes $C\langle inp, ct, outp': a \rangle C\langle outp', ct, outp: b \rangle$

shows $C\langle inp, ct, outp: a \wedge b \rangle$

$\langle proof \rangle$

lemma *DC-if*:

fixes ct **defines** $ct' \equiv \lambda pos\ name. (name, pos, []) \# ct$

assumes $H\langle ct' inp\ "then": a \rangle \implies C\langle Suc\ inp, ct' inp\ "then", outp': b \rangle$

assumes $H\langle ct' outp'\ "else": \neg a \rangle \implies C\langle Suc\ outp', ct' outp'\ "else", outp: c \rangle$

shows $C\langle inp, ct, outp: if\ a\ then\ b\ else\ c \rangle$

$\langle proof \rangle$

lemma *DC-final*:

assumes $V\langle ("g", inp, []), ct: a \rangle$

shows $C\langle inp, ct, Suc\ inp: a \rangle$

$\langle proof \rangle$

end

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

lemma

assumes $a: a$

and $d: b \implies c \implies d$

and $d': b \implies c \implies d'$

and $e: b \implies \neg c \implies e$

and $f: \neg b \implies f$

shows $a \wedge (if\ b\ then\ (if\ c\ then\ d \wedge d'\ else\ e)\ else\ f)$

$\langle proof \rangle$

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`

$\implies b$, 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  $\implies$  b  $\neg$ a  $\implies$  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 `LAbsortRule = 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^ $\wedge$ )>
```

and $C\langle \text{Suc } IC, ("then", IC, []) \# ("cond", IC, []) \# CT, OC1: \text{Valid } w \text{ c1 a1 } q \rangle$
and $C\langle \text{Suc } OC1, ("else", \text{Suc } OC1, []) \# ("cond", IC, []) \# CT, OC: \text{Valid } w' \text{ c2 a2 } q \rangle$
shows $C\langle IC, CT, OC: \text{Valid } p \text{ (Cond } b \text{ c1 c2) (Acond a1 a2) } q \rangle$
 $\langle \text{proof} \rangle$

lemma *LWhileRule:*

fixes $IC \ CT$ **defines** $CT' \equiv ("while", IC, []) \# CT$
assumes $V\langle ("precondition", IC, []), ("while", IC, []) \# CT: p \subseteq i \rangle$
and $C\langle \text{Suc } IC, ("invariant", \text{Suc } IC, []) \# ("while", IC, []) \# CT, OC: \text{Valid } (i \cap b) \text{ c (A } \theta) \text{ i} \rangle$
and $V\langle ("postcondition", IC, []), ("while", IC, []) \# CT: i \cap - b \subseteq q \rangle$
shows $C\langle IC, CT, OC: \text{Valid } p \text{ (While } b \text{ c) (Awhile } i \text{ v } A) \text{ } q \rangle$
 $\langle \text{proof} \rangle$

lemma *LABELs-to-prems:*

$(C\langle IC, CT, OC: \text{True} \rangle \implies P) \implies C\langle IC, CT, OC: P \rangle$
 $(V\langle x, ct: \text{True} \rangle \implies P) \implies V\langle x, ct: P \rangle$
 $\langle \text{proof} \rangle$

lemma *LABELs-to-concl:*

$C\langle IC, CT, OC: \text{True} \rangle \implies C\langle IC, CT, OC: P \rangle \implies P$
 $V\langle x, ct: \text{True} \rangle \implies V\langle x, ct: P \rangle \implies P$
 $\langle \text{proof} \rangle$

end

$\langle ML \rangle$

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*

$\{a=A \wedge b=B\}$
 $m := 0; s := 0;$
 $WHILE \ m \neq a$
 $INV \ \{s=m*b \wedge a=A \wedge b=B\}$
 $DO \ s := s+b; m := m+(1::nat) \ OD$
 $\{s = A*B\}$
 $\langle \text{proof} \rangle$

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 $\{0 \leq M \wedge (\exists p. p = (\text{if } m < 0 \text{ then } -m \text{ else } m) \wedge p * N = m * n \wedge P = (p - M) * N)\}$
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 $\{0 < a \wedge 0 < b \wedge \text{gcd } A B = \text{gcd } a b\}$
DO IF $a < b$ *THEN* $b := b - a$ *ELSE* $a := a - b$ *FI OD*
 $\{a = \text{gcd } A B\}$
<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 $\text{scm } x y = x * y / \text{gcd } x y$ and avoided division by multiplying with $\text{gcd } x y$.

lemmas *distrib =*
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 $\{0 < a \wedge 0 < b \wedge \text{gcd } A B = \text{gcd } a b \wedge 2 * A * B = a * x + b * y\}$
DO IF $a < b$ *THEN* $(b := b - a; x := x + y)$ *ELSE* $(a := a - b; y := y + x)$ *FI OD*
 $\{a = \text{gcd } A B \wedge 2 * A * B = a * (x + y)\}$
<proof>

4.4.4 Power by iterated squaring and multiplication

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

```

    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}

```

```
DO u := u - 1 OD;  
IF l ≤ u THEN A := A[l := A!u, u := A!l] ELSE SKIP FI  
OD  
{leq A u ∧ (∀ k. u < k ∧ k < l → A!k = pivot) ∧ geq A l}  
⟨proof⟩
```

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.