Making Arbitrary Relational Calculus Queries Safe-Range

Martin Raszyk   Dmitriy Traytel

March 17, 2025

Abstract

The relational calculus (RC), i.e., first-order logic with equality but without function symbols, is a concise, declarative database query language. In contrast to relational algebra or SQL, which are the traditional query languages of choice in the database community, RC queries can evaluate to an infinite relation. Moreover, even in cases where the evaluation result of an RC query would be finite it is not clear how to efficiently compute it. Safe-range RC is an interesting syntactic subclass of RC, because all safe-range queries evaluate to a finite result and it is well-known [1, §5.4] how to evaluate such queries by translating them to relational algebra. We formalize and prove correct our recent translation [2] of an arbitrary RC query into a pair of safe-range queries. Assuming an infinite domain, the two queries have the following meaning: The first is closed and characterizes the original query's relative safety, i.e., whether given a fixed database (interpretation of atomic predicates with finite relations), the original query evaluates to a finite relation. The second safe-range query is equivalent to the original query, if the latter is relatively safe.

The formalization uses the Refinement Framework to go from the non-deterministic algorithm described in the paper to a deterministic, executable query translation. Our executable query translation is a first step towards a verified tool that efficiently evaluates arbitrary RC queries. This very problem is also solved by the AFP entry Eval_FO with a theoretically incomparable but practically worse time complexity. (The latter is demonstrated by our empirical evaluation [2].)

Contents

1 Preliminaries

1.1 Iterated Function Update

abbreviation *fun_upds* (‹_[_ :=* _]› [*90*, *0*, *0*] *91*) **where**
$f[xs :=^* ys] \equiv fold\ (\lambda(x, y)\ f.\ f(x := y))\ (zip\ xs\ ys)\ f$

fun *restrict* **where**
$restrict\ A\ (x \# xs)\ (y \# ys) = (if\ x \in A\ then\ y \# restrict\ (A - \{x\})\ xs\ ys\ else\ restrict\ A\ xs\ ys)$
$|\ restrict\ A\ \_\ \_ = []$

fun *extend* :: $nat\ set \Rightarrow nat\ list \Rightarrow 'a\ list \Rightarrow 'a\ list\ set$ **where**
$extend\ A\ (x \# xs)\ ys = (if\ x \in A$
$then\ (\bigcup zs \in extend\ (A - \{x\})\ xs\ (tl\ ys).\ \{hd\ ys \# zs\})$
$else\ (\bigcup z \in UNIV.\ \bigcup zs \in extend\ A\ xs\ ys.\ \{z \# zs\}))$
$|\ extend\ A\ \_\ \_ = \{[]\}$

fun *lookup* **where**
$lookup\ (x \# xs)\ (y \# ys)\ z = (if\ x = z\ then\ y\ else\ lookup\ xs\ ys\ z)$
$|\ lookup\ \_\ \_\ \_ = undefined$

lemma *extend_nonempty*: $extend\ A\ xs\ ys \neq \{\}$
⟨*proof*⟩

lemma *length_extend*: $zs \in extend\ A\ xs\ ys \Longrightarrow length\ zs = length\ xs$
⟨*proof*⟩

lemma *ex_lookup_extend*: $x \notin A \Longrightarrow x \in set\ xs \Longrightarrow \exists zs \in extend\ A\ xs\ ys.\ lookup\ xs\ zs\ x = d$
⟨*proof*⟩

lemma *restrict_extend*: $A \subseteq set\ xs \Longrightarrow length\ ys = card\ A \Longrightarrow zs \in extend\ A\ xs\ ys \Longrightarrow restrict\ A\ xs\ zs = ys$
⟨*proof*⟩

lemma *fun_upds_notin*[*simp*]: $length\ xs = length\ ys \Longrightarrow x \notin set\ xs \Longrightarrow (\sigma[xs :=^* ys])\ x = \sigma\ x$
⟨*proof*⟩

lemma *fun_upds_twist*: $length\ xs = length\ ys \Longrightarrow a \notin set\ xs \Longrightarrow \sigma(a := x)[xs :=^* ys] = (\sigma[xs :=^* ys])(a := x)$
⟨*proof*⟩

lemma *fun_upds_twist_apply*: $length\ xs = length\ ys \Longrightarrow a \notin set\ xs \Longrightarrow a \neq b \Longrightarrow (\sigma(a := x)[xs :=^* ys])\ b = (\sigma[xs :=^* ys])\ b$
⟨*proof*⟩

lemma *fun_upds_extend*:
$x \in A \Longrightarrow A \subseteq set\ xs \Longrightarrow distinct\ xs \Longrightarrow sorted\ xs \Longrightarrow length\ ys = card\ A \Longrightarrow zs \in extend\ A\ xs\ ys \Longrightarrow (\sigma[xs :=^* zs])\ x = (\sigma[sorted\_list\_of\_set\ A :=^* ys])\ x$
⟨*proof*⟩

lemma *fun_upds_map_self*: $\sigma[xs :=^* map\ \sigma\ xs] = \sigma$
⟨*proof*⟩

lemma *fun_upds_single*: $distinct\ xs \Longrightarrow \sigma[xs :=^* map\ (\sigma(y := d))\ xs] = (if\ y \in set\ xs\ then\ \sigma(y := d)\ else\ \sigma)$
⟨*proof*⟩

1.2 Lists and Sets

lemma *find_index_less_size*: $\exists\, x \in set\ xs.\ P\ x \Longrightarrow find\_index\ P\ xs < size\ xs$
⟨*proof*⟩

lemma *index_less_size*: $x \in set\ xs \Longrightarrow index\ xs\ x < size\ xs$
⟨*proof*⟩

lemma *fun_upds_in*: $length\ xs = length\ ys \Longrightarrow distinct\ xs \Longrightarrow x \in set\ xs \Longrightarrow (\sigma[xs :=^* ys])\ x = ys\ !\ index\ xs\ x$
⟨*proof*⟩

lemma *remove_nth_index*: $remove\_nth\ (index\ ys\ y)\ ys = remove1\ y\ ys$
⟨*proof*⟩

lemma *index_remove_nth*: $distinct\ xs \Longrightarrow x \in set\ xs \Longrightarrow index\ (remove\_nth\ i\ xs)\ x = (if\ index\ xs\ x < i\ then\ index\ xs\ x\ else\ if\ i = index\ xs\ x\ then\ length\ xs - 1\ else\ index\ xs\ x - 1)$
⟨*proof*⟩

lemma *insert_nth_nth_index*:
$y \neq z \Longrightarrow y \in set\ ys \Longrightarrow z \in set\ ys \Longrightarrow length\ ys = Suc\ (length\ xs) \Longrightarrow distinct\ ys \Longrightarrow$
$insert\_nth\ (index\ ys\ y)\ x\ xs\ !\ index\ ys\ z =$
$xs\ !\ index\ (remove1\ y\ ys)\ z$
⟨*proof*⟩

lemma *index_lt_index_remove*: $index\ xs\ x < index\ xs\ y \Longrightarrow index\ xs\ x = index\ (remove1\ y\ xs)\ x$
⟨*proof*⟩

lemma *index_gt_index_remove*: $index\ xs\ x > index\ xs\ y \Longrightarrow index\ xs\ x = Suc\ (index\ (remove1\ y\ xs)\ x)$
⟨*proof*⟩

lemma *lookup_map*[*simp*]: $x \in set\ xs \Longrightarrow lookup\ xs\ (map\ f\ xs)\ x = f\ x$
⟨*proof*⟩

lemma *in_set_remove_cases*: $P\ z \Longrightarrow (\forall\, x \in set\ (remove1\ z\ xs).\ P\ x) \Longrightarrow x \in set\ xs \Longrightarrow P\ x$
⟨*proof*⟩

lemma *insert_remove_id*: $x \in X \Longrightarrow X = insert\ x\ (X - \{x\})$
⟨*proof*⟩

lemma *infinite_surj*: $infinite\ A \Longrightarrow A \subseteq f\ `\ B \Longrightarrow infinite\ B$
⟨*proof*⟩

class *infinite* =
fixes $to\_nat :: 'a \Rightarrow nat$

assumes *surj_to_nat*: *surj to_nat*
begin

lemma *infinite_UNIV*: *infinite* (*UNIV* :: ′*a set*)
⟨*proof*⟩

end

instantiation *nat* :: *infinite* **begin**
definition *to_nat_nat* :: *nat* ⇒ *nat* **where** *to_nat_nat* = *id*
instance ⟨*proof*⟩
end

instantiation *list* :: (*type*) *infinite* **begin**
definition *to_nat_list* :: ′*a list* ⇒ *nat* **where** *to_nat_list* = *length*
instance ⟨*proof*⟩
end

1.3 Equivalence Closure and Classes

definition *symcl* **where**
symcl r = {(*x*, *y*). (*x*, *y*) ∈ *r* ∨ (*y*, *x*) ∈ *r*}

definition *transymcl* **where**
transymcl r = *trancl* (*symcl r*)

lemma *symclp_symcl_eq*[*pred_set_conv*]: *symclp* (λ*x y*. (*x*, *y*) ∈ *r*) = (λ*x y*. (*x*, *y*) ∈ *symcl r*)
⟨*proof*⟩

definition *classes Qeq* = *quotient* (*Field Qeq*) (*transymcl Qeq*)

lemma *Field_symcl*[*simp*]: *Field* (*symcl r*) = *Field r*
⟨*proof*⟩

lemma *Domain_symcl*[*simp*]: *Domain* (*symcl r*) = *Field r*
⟨*proof*⟩

lemma *Field_trancl*[*simp*]: *Field* (*trancl r*) = *Field r*
⟨*proof*⟩

lemma *Field_transymcl*[*simp*]: *Field* (*transymcl r*) = *Field r*
⟨*proof*⟩

lemma *eqclass_empty_iff*[*simp*]: *r* ‘‘ {*x*} = {} ⟷ *x* ∉ *Domain r*
⟨*proof*⟩

lemma *sym_symcl*[*simp*]: *sym* (*symcl r*)
⟨*proof*⟩

lemma *in_symclI*:
(*a*,*b*) ∈ *r* ⟹ (*a*,*b*) ∈ *symcl r*
(*a*,*b*) ∈ *r* ⟹ (*b*,*a*) ∈ *symcl r*
⟨*proof*⟩

lemma *sym_transymcl*: *sym* (*transymcl r*)
⟨*proof*⟩

lemma *symcl_insert*:

$symcl\ (insert\ (x,\ y)\ Qeq) = insert\ (y,\ x)\ (insert\ (x,\ y)\ (symcl\ Qeq))$
⟨*proof*⟩

lemma *equiv_transymcl*: $Equiv\_Relations.equiv\ (Field\ Qeq)\ (transymcl\ Qeq)$
⟨*proof*⟩

lemma *equiv_quotient_no_empty_class*: $Equiv\_Relations.equiv\ A\ r \Longrightarrow \{\} \notin A\ //\ r$
⟨*proof*⟩

lemma *classes_cover*: $\bigcup(classes\ Qeq) = Field\ Qeq$
⟨*proof*⟩

lemma *classes_disjoint*: $X \in classes\ Qeq \Longrightarrow Y \in classes\ Qeq \Longrightarrow X = Y \lor X \cap Y = \{\}$
⟨*proof*⟩

lemma *classes_nonempty*: $\{\} \notin classes\ Qeq$
⟨*proof*⟩

definition $class\ x\ Qeq = (if\ \exists X \in classes\ Qeq.\ x \in X\ then\ Some\ (THE\ X.\ X \in classes\ Qeq \land x \in X)$ $else\ None)$

lemma *class_Some_eq*: $class\ x\ Qeq = Some\ X \longleftrightarrow X \in classes\ Qeq \land x \in X$
⟨*proof*⟩

lemma *class_None_eq*: $class\ x\ Qeq = None \longleftrightarrow x \notin Field\ Qeq$
⟨*proof*⟩

lemma *insert_Image_triv*: $x \notin r \Longrightarrow insert\ (x,\ y)\ Qeq\ ``\ r = Qeq\ ``\ r$
⟨*proof*⟩

lemma *Un1_Image_triv*: $Domain\ B \cap r = \{\} \Longrightarrow (A \cup B)\ ``\ r = A\ ``\ r$
⟨*proof*⟩

lemma *Un2_Image_triv*: $Domain\ A \cap r = \{\} \Longrightarrow (A \cup B)\ ``\ r = B\ ``\ r$
⟨*proof*⟩

lemma *classes_empty*: $classes\ \{\} = \{\}$
⟨*proof*⟩

lemma *ex_class*: $x \in Field\ Qeq \Longrightarrow \exists X.\ class\ x\ Qeq = Some\ X \land x \in X$
⟨*proof*⟩

lemma *equivD*:
$Equiv\_Relations.equiv\ A\ r \Longrightarrow refl\_on\ A\ r$
$Equiv\_Relations.equiv\ A\ r \Longrightarrow sym\ r$
$Equiv\_Relations.equiv\ A\ r \Longrightarrow trans\ r$
⟨*proof*⟩

lemma *transymcl_into*:
$(x,\ y) \in r \Longrightarrow (x,\ y) \in transymcl\ r$
$(x,\ y) \in r \Longrightarrow (y,\ x) \in transymcl\ r$
⟨*proof*⟩

lemma *transymcl_self*:
$(x,\ y) \in r \Longrightarrow (x,\ x) \in transymcl\ r$
$(x,\ y) \in r \Longrightarrow (y,\ y) \in transymcl\ r$
⟨*proof*⟩

lemma *transymcl_trans*: $(x,\ y) \in transymcl\ r \Longrightarrow (y,\ z) \in transymcl\ r \Longrightarrow (x,\ z) \in transymcl\ r$
⟨*proof*⟩

lemma *transymcl_sym*: $(x,\ y) \in transymcl\ r \Longrightarrow (y,\ x) \in transymcl\ r$
⟨*proof*⟩

lemma *edge_same_class*: $X \in classes\ Qeq \Longrightarrow (a,\ b) \in Qeq \Longrightarrow a \in X \longleftrightarrow b \in X$
⟨*proof*⟩

lemma *Field_transymcl_self*: $a \in Field\ Qeq \Longrightarrow (a,\ a) \in transymcl\ Qeq$
⟨*proof*⟩

lemma *transymcl_insert*: $transymcl\ (insert\ (a,\ b)\ Qeq) = transymcl\ Qeq \cup \{(a,a),(b,b)\} \cup ((transymcl\ Qeq \cup \{(a,\ a),\ (b,\ b)\})\ O\ \{(a,\ b),\ (b,\ a)\}\ O\ (transymcl\ Qeq \cup \{(a,\ a),\ (b,\ b)\}) - transymcl\ Qeq)$
⟨*proof*⟩

lemma *transymcl_insert_both_new*: $a \notin Field\ Qeq \Longrightarrow b \notin Field\ Qeq \Longrightarrow$
$transymcl\ (insert\ (a,\ b)\ Qeq) = transymcl\ Qeq \cup \{(a,a),(b,b),(a,b),(b,a)\}$
⟨*proof*⟩

lemma *transymcl_insert_same_class*: $(x,\ y) \in transymcl\ Qeq \Longrightarrow transymcl\ (insert\ (x,\ y)\ Qeq) = transymcl\ Qeq$
⟨*proof*⟩

lemma *classes_insert*: $classes\ (insert\ (x,\ y)\ Qeq) =$
$(case\ (class\ x\ Qeq,\ class\ y\ Qeq)\ of$
$(Some\ X,\ Some\ Y) \Rightarrow if\ X = Y\ then\ classes\ Qeq\ else\ classes\ Qeq - \{X,\ Y\} \cup \{X \cup Y\}$
$|\ (Some\ X,\ None) \Rightarrow classes\ Qeq - \{X\} \cup \{insert\ y\ X\}$
$|\ (None,\ Some\ Y) \Rightarrow classes\ Qeq - \{Y\} \cup \{insert\ x\ Y\}$
$|\ (None,\ None) \Rightarrow classes\ Qeq \cup \{\{x,y\}\})$
⟨*proof*⟩

lemma *classes_intersect_find_not_None*:
assumes $\forall\ V{\in}classes\ (set\ xys).\ V \cap A \neq \{\}\ xys \neq []$
shows $find\ (\lambda(x,\ y).\ x \in A \lor y \in A)\ xys \neq None$
⟨*proof*⟩

2 Relational Calculus

2.1 First-order Terms

datatype *'a term = Const 'a | Var nat*

type_synonym *'a val = nat ⇒ 'a*

fun *fv_term_set* :: *'a term ⇒ nat set* **where**
fv_term_set (Var n) = {n}
| *fv_term_set _ = {}*

fun *fv_fo_term_list* :: *'a term ⇒ nat list* **where**
fv_fo_term_list (Var n) = [n]
| *fv_fo_term_list _ = []*

definition *fv_terms_set* :: *('a term) list ⇒ nat set* **where**
fv_terms_set ts = $\bigcup$ *(set (map fv_term_set ts))*

fun *eval_term* :: *'a val ⇒ 'a term ⇒ 'a* (**infix** ‹·› *60*) **where**

eval_term σ (*Const c*) = *c*
| *eval_term* σ (*Var n*) = σ *n*

definition *eval_terms* :: *'a val* ⇒ (*'a term*) *list* ⇒ *'a list* (**infix** ‹⊙› *60*) **where**
eval_terms σ *ts* = *map* (*eval_term* σ) *ts*

lemma *finite_set_term*: *finite* (*set_term t*)
⟨*proof*⟩

lemma *finite_fv_term_set*: *finite* (*fv_term_set t*)
⟨*proof*⟩

lemma *fv_term_setD*: *n* ∈ *fv_term_set t* ⟹ *t* = *Var n*
⟨*proof*⟩

lemma *fv_term_set_cong*: *fv_term_set t* = *fv_term_set* (*map_term f t*)
⟨*proof*⟩

lemma *fv_terms_setI*: *Var m* ∈ *set ts* ⟹ *m* ∈ *fv_terms_set ts*
⟨*proof*⟩

lemma *fv_terms_setD*: *m* ∈ *fv_terms_set ts* ⟹ *Var m* ∈ *set ts*
⟨*proof*⟩

lemma *finite_fv_terms_set*: *finite* (*fv_terms_set ts*)
⟨*proof*⟩

lemma *fv_terms_set_cong*: *fv_terms_set ts* = *fv_terms_set* (*map* (*map_term f*) *ts*)
⟨*proof*⟩

lemma *eval_term_cong*: (⋀*n*. *n* ∈ *fv_term_set t* ⟹ σ *n* = σ' *n*) ⟹
eval_term σ *t* = *eval_term* σ' *t*
⟨*proof*⟩

lemma *eval_terms_fv_terms_set*: σ ⊙ *ts* = σ' ⊙ *ts* ⟹ *n* ∈ *fv_terms_set ts* ⟹ σ *n* = σ' *n*
⟨*proof*⟩

lemma *eval_terms_cong*: (⋀*n*. *n* ∈ *fv_terms_set ts* ⟹ σ *n* = σ' *n*) ⟹
eval_terms σ *ts* = *eval_terms* σ' *ts*
⟨*proof*⟩

2.2 Relational Calculus Syntax and Semantics

datatype (*discs_sels*) (*'a*, *'b*) *fmla* =
Pred 'b (*'a term*) *list*
| *Bool bool*
| *Eq nat 'a term*
| *Neg* (*'a*, *'b*) *fmla*
| *Conj* (*'a*, *'b*) *fmla* (*'a*, *'b*) *fmla*
| *Disj* (*'a*, *'b*) *fmla* (*'a*, *'b*) *fmla*
| *Exists nat* (*'a*, *'b*) *fmla*

derive *linorder term*
derive *linorder fmla*

fun *fv* :: (*'a*, *'b*) *fmla* ⇒ *nat set* **where**
fv (*Pred _ ts*) = *fv_terms_set ts*
| *fv* (*Bool b*) = {}

| fv $(Eq\ x\ t') = \{x\} \cup fv\_term\_set\ t'$
| fv $(Neg\ \varphi) = fv\ \varphi$
| fv $(Conj\ \varphi\ \psi) = fv\ \varphi \cup fv\ \psi$
| fv $(Disj\ \varphi\ \psi) = fv\ \varphi \cup fv\ \psi$
| fv $(Exists\ z\ \varphi) = fv\ \varphi - \{z\}$

definition *exists* **where** $exists\ x\ Q = (if\ x \in fv\ Q\ then\ Exists\ x\ Q\ else\ Q)$
abbreviation $Forall\ x\ Q \equiv Neg\ (Exists\ x\ (Neg\ Q))$
abbreviation $forall\ x\ Q \equiv Neg\ (exists\ x\ (Neg\ Q))$
abbreviation $Impl\ Q1\ Q2 \equiv Disj\ (Neg\ Q1)\ Q2$

definition $EXISTS\ xs\ Q = fold\ Exists\ xs\ Q$

abbreviation *close* **where**
$close\ Q \equiv EXISTS\ (sorted\_list\_of\_set\ (fv\ Q))\ Q$

lemma *fv_exists*[*simp*]: $fv\ (exists\ x\ Q) = fv\ Q - \{x\}$
⟨*proof*⟩

lemma *fv_EXISTS*: $fv\ (EXISTS\ xs\ Q) = fv\ Q - set\ xs$
⟨*proof*⟩

lemma *exists_Exists*: $x \in fv\ Q \Longrightarrow exists\ x\ Q = Exists\ x\ Q$
⟨*proof*⟩

lemma *is_Bool_exists*[*simp*]: $is\_Bool\ (exists\ x\ Q) = is\_Bool\ Q$
⟨*proof*⟩

lemma *finite_fv*[*simp*]: $finite\ (fv\ \varphi)$
⟨*proof*⟩

lemma *fv_close*[*simp*]: $fv\ (close\ Q) = \{\}$
⟨*proof*⟩

type_synonym $'a\ table = ('a\ list)\ set$
type_synonym $('a, 'b)\ intp = 'b \times nat \Rightarrow 'a\ table$

definition $adom :: ('a, 'b)\ intp \Rightarrow 'a\ set$ **where**
$adom\ I = (\bigcup rn.\ \bigcup xs \in I\ rn.\ set\ xs)$

fun $sat :: ('a, 'b)\ fmla \Rightarrow ('a, 'b)\ intp \Rightarrow 'a\ val \Rightarrow bool$ **where**
$sat\ (Pred\ r\ ts)\ I\ \sigma \longleftrightarrow \sigma \odot ts \in I\ (r, length\ ts)$
| $sat\ (Bool\ b)\ I\ \sigma \longleftrightarrow b$
| $sat\ (Eq\ x\ t')\ I\ \sigma \longleftrightarrow \sigma\ x = \sigma \cdot t'$
| $sat\ (Neg\ \varphi)\ I\ \sigma \longleftrightarrow \neg sat\ \varphi\ I\ \sigma$
| $sat\ (Conj\ \varphi\ \psi)\ I\ \sigma \longleftrightarrow sat\ \varphi\ I\ \sigma \wedge sat\ \psi\ I\ \sigma$
| $sat\ (Disj\ \varphi\ \psi)\ I\ \sigma \longleftrightarrow sat\ \varphi\ I\ \sigma \vee sat\ \psi\ I\ \sigma$
| $sat\ (Exists\ z\ \varphi)\ I\ \sigma \longleftrightarrow (\exists x.\ sat\ \varphi\ I\ (\sigma(z := x)))$

lemma *sat_fv_cong*: $(\bigwedge n.\ n \in fv\ \varphi \Longrightarrow \sigma\ n = \sigma'\ n) \Longrightarrow$
$sat\ \varphi\ I\ \sigma \longleftrightarrow sat\ \varphi\ I\ \sigma'$
⟨*proof*⟩

lemma *sat_fun_upd*: $n \notin fv\ Q \Longrightarrow sat\ Q\ I\ (\sigma(n := z)) = sat\ Q\ I\ \sigma$
⟨*proof*⟩

lemma *sat_exists*[*simp*]: $sat\ (exists\ n\ Q)\ I\ \sigma = (\exists x.\ sat\ Q\ I\ (\sigma(n := x)))$
⟨*proof*⟩

abbreviation *eq* (**infix** ‹≈› *80*) **where**
$x \approx y \equiv Eq\ x\ (Var\ y)$

definition *equiv* (**infix** ‹≜› *100*) **where**
$Q1 \triangleq Q2 = (\forall I\ \sigma.\ finite\ (adom\ I) \longrightarrow sat\ Q1\ I\ \sigma \longleftrightarrow sat\ Q2\ I\ \sigma)$

lemma *equiv_refl*[*iff*]: $Q \triangleq Q$
⟨*proof*⟩

lemma *equiv_sym*[*sym*]: $Q1 \triangleq Q2 \Longrightarrow Q2 \triangleq Q1$
⟨*proof*⟩

lemma *equiv_trans*[*trans*]: $Q1 \triangleq Q2 \Longrightarrow Q2 \triangleq Q3 \Longrightarrow Q1 \triangleq Q3$
⟨*proof*⟩

lemma *equiv_Neg_cong*[*simp*]: $Q \triangleq Q' \Longrightarrow Neg\ Q \triangleq Neg\ Q'$
⟨*proof*⟩

lemma *equiv_Conj_cong*[*simp*]: $Q1 \triangleq Q1' \Longrightarrow Q2 \triangleq Q2' \Longrightarrow Conj\ Q1\ Q2 \triangleq Conj\ Q1'\ Q2'$
⟨*proof*⟩

lemma *equiv_Disj_cong*[*simp*]: $Q1 \triangleq Q1' \Longrightarrow Q2 \triangleq Q2' \Longrightarrow Disj\ Q1\ Q2 \triangleq Disj\ Q1'\ Q2'$
⟨*proof*⟩

lemma *equiv_Exists_cong*[*simp*]: $Q \triangleq Q' \Longrightarrow Exists\ x\ Q \triangleq Exists\ x\ Q'$
⟨*proof*⟩

lemma *equiv_Exists_exists_cong*[*simp*]: $Q \triangleq Q' \Longrightarrow Exists\ x\ Q \triangleq exists\ x\ Q'$
⟨*proof*⟩

lemma *equiv_Exists_Disj*: $Exists\ x\ (Disj\ Q1\ Q2) \triangleq Disj\ (Exists\ x\ Q1)\ (Exists\ x\ Q2)$
⟨*proof*⟩

lemma *equiv_Disj_Assoc*: $Disj\ (Disj\ Q1\ Q2)\ Q3 \triangleq Disj\ Q1\ (Disj\ Q2\ Q3)$
⟨*proof*⟩

lemma *foldr_Disj_equiv_cong*[*simp*]:
$list\_all2\ (\triangleq)\ xs\ ys \Longrightarrow b \triangleq c \Longrightarrow foldr\ Disj\ xs\ b \triangleq foldr\ Disj\ ys\ c$
⟨*proof*⟩

lemma *Exists_nonfree_equiv*: $x \notin fv\ Q \Longrightarrow Exists\ x\ Q \triangleq Q$
⟨*proof*⟩

2.3 Constant Propagation

fun *cp* **where**
cp (Eq x t) = (case t of Var y ⇒ if x = y then Bool True else x ≈ y | _ ⇒ Eq x t)
| cp (Neg Q) = (let Q′ = cp Q in if is_Bool Q′ then Bool (¬ un_Bool Q′) else Neg Q′)
| cp (Conj Q1 Q2) =
(let Q1′ = cp Q1; Q2′ = cp Q2 in
if is_Bool Q1′ then if un_Bool Q1′ then Q2′ else Bool False
else if is_Bool Q2′ then if un_Bool Q2′ then Q1′ else Bool False
else Conj Q1′ Q2′)
| cp (Disj Q1 Q2) =
(let Q1′ = cp Q1; Q2′ = cp Q2 in
if is_Bool Q1′ then if un_Bool Q1′ then Bool True else Q2′
else if is_Bool Q2′ then if un_Bool Q2′ then Bool True else Q1′

else Disj Q1′ Q2′)
| *cp* (*Exists x Q*) = *exists x* (*cp Q*)
| *cp Q* = *Q*

lemma *fv_cp*: *fv* (*cp Q*) ⊆ *fv Q*
⟨*proof*⟩

lemma *cp_exists*[*simp*]: *cp* (*exists x Q*) = *exists x* (*cp Q*)
⟨*proof*⟩

fun *nocp* **where**
nocp (*Bool b*) = *False*
| *nocp* (*Pred p ts*) = *True*
| *nocp* (*Eq x t*) = (*t* ≠ *Var x*)
| *nocp* (*Neg Q*) = *nocp Q*
| *nocp* (*Conj Q1 Q2*) = (*nocp Q1* ∧ *nocp Q2*)
| *nocp* (*Disj Q1 Q2*) = (*nocp Q1* ∧ *nocp Q2*)
| *nocp* (*Exists x Q*) = (*x* ∈ *fv Q* ∧ *nocp Q*)

lemma *nocp_exists*[*simp*]: *nocp* (*exists x Q*) = *nocp Q*
⟨*proof*⟩

lemma *nocp_cp_triv*: *nocp Q* ⟹ *cp Q* = *Q*
⟨*proof*⟩

lemma *is_Bool_cp_triv*: *is_Bool Q* ⟹ *cp Q* = *Q*
⟨*proof*⟩

lemma *nocp_cp_or_is_Bool*: *nocp* (*cp Q*) ∨ *is_Bool* (*cp Q*)
⟨*proof*⟩

lemma *cp_idem*[*simp*]: *cp* (*cp Q*) = *cp Q*
⟨*proof*⟩

lemma *sat_cp*[*simp*]: *sat* (*cp Q*) *I σ* = *sat Q I σ*
⟨*proof*⟩

lemma *equiv_cp_cong*[*simp*]: *Q* ≜ *Q′* ⟹ *cp Q* ≜ *cp Q′*
⟨*proof*⟩

lemma *equiv_cp*[*simp*]: *cp Q* ≜ *Q*
⟨*proof*⟩

definition *cpropagated* **where** *cpropagated Q* = (*nocp Q* ∨ *is_Bool Q*)

lemma *cpropagated_cp*[*simp*]: *cpropagated* (*cp Q*)
⟨*proof*⟩

lemma *nocp_cpropagated*[*simp*]: *nocp Q* ⟹ *cpropagated Q*
⟨*proof*⟩

lemma *cpropagated_cp_triv*: *cpropagated Q* ⟹ *cp Q* = *Q*
⟨*proof*⟩

lemma *cpropagated_nocp*: *cpropagated Q* ⟹ *x* ∈ *fv Q* ⟹ *nocp Q*
⟨*proof*⟩

lemma *cpropagated_simps*[*simp*]:

$cpropagated\ (Bool\ b) \longleftrightarrow True$
$cpropagated\ (Pred\ p\ ts) \longleftrightarrow True$
$cpropagated\ (Eq\ x\ t) \longleftrightarrow t \neq Var\ x$
$cpropagated\ (Neg\ Q) \longleftrightarrow nocp\ Q$
$cpropagated\ (Conj\ Q1\ Q2) \longleftrightarrow nocp\ Q1 \wedge nocp\ Q2$
$cpropagated\ (Disj\ Q1\ Q2) \longleftrightarrow nocp\ Q1 \wedge nocp\ Q2$
$cpropagated\ (Exists\ x\ Q) \longleftrightarrow x \in fv\ Q \wedge nocp\ Q$
⟨*proof*⟩

2.4 Big Disjunction

fun *foldr1* **where**
foldr1 f (x # xs) z = foldr f xs x
| *foldr1 f [] z = z*

definition *DISJ* **where**
DISJ G = foldr1 Disj (sorted_list_of_set G) (Bool False)

lemma *sat_foldr_Disj*[*simp*]: $sat\ (foldr\ Disj\ xs\ Q)\ I\ \sigma = (\exists\, Q \in set\ xs \cup \{Q\}.\ sat\ Q\ I\ \sigma)$
⟨*proof*⟩

lemma *sat_foldr1_Disj*[*simp*]: $sat\ (foldr1\ Disj\ xs\ Q)\ I\ \sigma = (if\ xs = []\ then\ sat\ Q\ I\ \sigma\ else\ \exists\, Q \in set\ xs.\ sat\ Q\ I\ \sigma)$
⟨*proof*⟩

lemma *sat_DISJ*[*simp*]: $finite\ G \Longrightarrow sat\ (DISJ\ G)\ I\ \sigma = (\exists\, Q \in G.\ sat\ Q\ I\ \sigma)$
⟨*proof*⟩

lemma *foldr_Disj_equiv*: $insert\ Q\ (set\ Qs) = insert\ Q'\ (set\ Qs') \Longrightarrow foldr\ Disj\ Qs\ Q \triangleq foldr\ Disj\ Qs'\ Q'$
⟨*proof*⟩

lemma *foldr1_Disj_equiv*: $set\ Qs = set\ Qs' \Longrightarrow foldr1\ Disj\ Qs\ (Bool\ False) \triangleq foldr1\ Disj\ Qs'\ (Bool\ False)$
⟨*proof*⟩

lemma *foldr1_Disj_equiv_cong*[*simp*]:
$list\_all2\ (\triangleq)\ xs\ ys \Longrightarrow b \triangleq c \Longrightarrow foldr1\ Disj\ xs\ b \triangleq foldr1\ Disj\ ys\ c$
⟨*proof*⟩

lemma *Exists_foldr_Disj*:
$Exists\ x\ (foldr\ Disj\ xs\ b) \triangleq foldr\ Disj\ (map\ (exists\ x)\ xs)\ (exists\ x\ b)$
⟨*proof*⟩

lemma *Exists_foldr1_Disj*:
$Exists\ x\ (foldr1\ Disj\ xs\ b) \triangleq foldr1\ Disj\ (map\ (exists\ x)\ xs)\ (exists\ x\ b)$
⟨*proof*⟩

lemma *Exists_DISJ*:
$finite\ \mathcal{Q} \Longrightarrow Exists\ x\ (DISJ\ \mathcal{Q}) \triangleq DISJ\ (exists\ x\ `\ \mathcal{Q})$
⟨*proof*⟩

lemma *Exists_cp_DISJ*:
$finite\ \mathcal{Q} \Longrightarrow Exists\ x\ (cp\ (DISJ\ \mathcal{Q})) \triangleq DISJ\ (exists\ x\ `\ \mathcal{Q})$
⟨*proof*⟩

lemma *Disj_empty*[*simp*]: $DISJ\ \{\} = Bool\ False$
⟨*proof*⟩

lemma *Disj_single*[*simp*]: $DISJ\ \{x\} = x$
⟨*proof*⟩

lemma *DISJ_insert*[*simp*]: $finite\ X \Longrightarrow DISJ\ (insert\ x\ X) \triangleq Disj\ x\ (DISJ\ X)$
⟨*proof*⟩

lemma *DISJ_union*[*simp*]: $finite\ X \Longrightarrow finite\ Y \Longrightarrow DISJ\ (X \cup Y) \triangleq Disj\ (DISJ\ X)\ (DISJ\ Y)$
⟨*proof*⟩

lemma *DISJ_exists_pull_out*: $finite\ \mathcal{Q} \Longrightarrow Q \in \mathcal{Q} \Longrightarrow$
$DISJ\ (exists\ x\ `\ \mathcal{Q}) \triangleq Disj\ (Exists\ x\ Q)\ (DISJ\ (exists\ x\ `\ (\mathcal{Q} - \{Q\})))$
⟨*proof*⟩

lemma *DISJ_push_in*: $finite\ \mathcal{Q} \Longrightarrow Disj\ Q\ (DISJ\ \mathcal{Q}) \triangleq DISJ\ (insert\ Q\ \mathcal{Q})$
⟨*proof*⟩

lemma *DISJ_insert_reorder*: $finite\ \mathcal{Q} \Longrightarrow DISJ\ (insert\ (Disj\ Q1\ Q2)\ \mathcal{Q}) \triangleq DISJ\ (insert\ Q2\ (insert\ Q1\ \mathcal{Q}))$
⟨*proof*⟩

lemma *DISJ_insert_reorder'*: $finite\ \mathcal{Q} \Longrightarrow finite\ \mathcal{Q}' \Longrightarrow DISJ\ (insert\ (Disj\ (DISJ\ \mathcal{Q}')\ Q2)\ \mathcal{Q}) \triangleq DISJ\ (insert\ Q2\ (\mathcal{Q}' \cup \mathcal{Q}))$
⟨*proof*⟩

lemma *fv_foldr_Disj*[*simp*]: $fv\ (foldr\ Disj\ Qs\ Q) = (fv\ Q \cup (\bigcup Q \in set\ Qs.\ fv\ Q))$
⟨*proof*⟩

lemma *fv_foldr1_Disj*[*simp*]: $fv\ (foldr1\ Disj\ Qs\ Q) = (if\ Qs = []\ then\ fv\ Q\ else\ (\bigcup Q \in set\ Qs.\ fv\ Q))$
⟨*proof*⟩

lemma *fv_DISJ*: $finite\ \mathcal{Q} \Longrightarrow fv\ (DISJ\ \mathcal{Q}) \subseteq (\bigcup Q \in \mathcal{Q}.\ fv\ Q)$
⟨*proof*⟩

lemma *fv_DISJ_close*[*simp*]: $finite\ \mathcal{Q} \Longrightarrow fv\ (DISJ\ (close\ `\ \mathcal{Q})) = \{\}$
⟨*proof*⟩

lemma *fv_cp_foldr_Disj*: $\forall\ Q{\in}set\ Qs \cup \{Q\}.\ cpropagated\ Q \wedge fv\ Q = A \Longrightarrow fv\ (cp\ (foldr\ Disj\ Qs\ Q)) = A$
⟨*proof*⟩

lemma *fv_cp_foldr1_Disj*: $cp\ (foldr1\ Disj\ Qs\ (Bool\ False)) \neq Bool\ False \Longrightarrow$
$\forall\ Q{\in}set\ Qs.\ cpropagated\ Q \wedge fv\ Q = A \Longrightarrow$
$fv\ (cp\ (foldr1\ Disj\ Qs\ (Bool\ False))) = A$
⟨*proof*⟩

lemma *fv_cp_DISJ_eq*: $finite\ \mathcal{Q} \Longrightarrow cp\ (DISJ\ \mathcal{Q}) \neq Bool\ False \Longrightarrow \forall\ Q \in \mathcal{Q}.\ cpropagated\ Q \wedge fv\ Q = A \Longrightarrow fv\ (cp\ (DISJ\ \mathcal{Q})) = A$
⟨*proof*⟩

fun *sub* **where**
$sub\ (Bool\ t) = \{Bool\ t\}$
$|\ sub\ (Pred\ p\ ts) = \{Pred\ p\ ts\}$
$|\ sub\ (Eq\ x\ t) = \{Eq\ x\ t\}$
$|\ sub\ (Neg\ Q) = insert\ (Neg\ Q)\ (sub\ Q)$
$|\ sub\ (Conj\ Q1\ Q2) = insert\ (Conj\ Q1\ Q2)\ (sub\ Q1 \cup sub\ Q2)$
$|\ sub\ (Disj\ Q1\ Q2) = insert\ (Disj\ Q1\ Q2)\ (sub\ Q1 \cup sub\ Q2)$
$|\ sub\ (Exists\ z\ Q) = insert\ (Exists\ z\ Q)\ (sub\ Q)$

lemma *cpropagated_sub*: *cpropagated Q* $\Longrightarrow$ *Q′* $\in$ *sub Q* $\Longrightarrow$ *cpropagated Q′*
⟨*proof*⟩

lemma *Exists_in_sub_cp_foldr_Disj*:
Exists x Q′ $\in$ *sub* (*cp* (*foldr Disj Qs Q*)) $\Longrightarrow$ *Exists x Q′* $\in$ *sub* (*cp Q*) $\vee$ ($\exists$ *Q* $\in$ *set Qs*. *Exists x Q′* $\in$ *sub* (*cp Q*))
⟨*proof*⟩

lemma *Exists_in_sub_cp_foldr1_Disj*:
Exists x Q′ $\in$ *sub* (*cp* (*foldr1 Disj Qs Q*)) $\Longrightarrow$ *Qs* = [] $\wedge$ *Exists x Q′* $\in$ *sub* (*cp Q*) $\vee$ ($\exists$ *Q* $\in$ *set Qs*. *Exists x Q′* $\in$ *sub* (*cp Q*))
⟨*proof*⟩

lemma *Exists_in_sub_cp_DISJ*: *Exists x Q′* $\in$ *sub* (*cp* (*DISJ* $\mathcal{Q}$)) $\Longrightarrow$ *finite* $\mathcal{Q}$ $\Longrightarrow$ ($\exists$ *Q* $\in$ $\mathcal{Q}$. *Exists x Q′* $\in$ *sub* (*cp Q*))
⟨*proof*⟩

lemma *Exists_in_sub_foldr_Disj*:
Exists x Q′ $\in$ *sub* (*foldr Disj Qs Q*) $\Longrightarrow$ *Exists x Q′* $\in$ *sub Q* $\vee$ ($\exists$ *Q* $\in$ *set Qs*. *Exists x Q′* $\in$ *sub Q*)
⟨*proof*⟩

lemma *Exists_in_sub_foldr1_Disj*:
Exists x Q′ $\in$ *sub* (*foldr1 Disj Qs Q*) $\Longrightarrow$ *Qs* = [] $\wedge$ *Exists x Q′* $\in$ *sub Q* $\vee$ ($\exists$ *Q* $\in$ *set Qs*. *Exists x Q′* $\in$ *sub Q*)
⟨*proof*⟩

lemma *Exists_in_sub_DISJ*: *Exists x Q′* $\in$ *sub* (*DISJ* $\mathcal{Q}$) $\Longrightarrow$ *finite* $\mathcal{Q}$ $\Longrightarrow$ ($\exists$ *Q* $\in$ $\mathcal{Q}$. *Exists x Q′* $\in$ *sub Q*)
⟨*proof*⟩

2.5 Substitution

fun *subst_term* (‹_[_ →t _]› [*90*, *0*, *0*] *91*) **where**
Var z[*x* →*t y*] = *Var* (*if x* = *z then y else z*)
| *Const c*[*x* →*t y*] = *Const c*

abbreviation *substs_term* (‹_[_ →t* _]› [*90*, *0*, *0*] *91*) **where**
t[*xs* →*t** *ys*] ≡ *fold* (λ(*x*, *y*) *t*. *t*[*x* →*t y*]) (*zip xs ys*) *t*

lemma *size_subst_term*[*simp*]: *size* (*t*[*x* →*t y*]) = *size t*
⟨*proof*⟩

lemma *fv_subst_term*[*simp*]: *fv_term_set* (*t*[*x* →*t y*]) =
(*if x* $\in$ *fv_term_set t then insert y* (*fv_term_set t* − {*x*}) *else fv_term_set t*)
⟨*proof*⟩

definition *fresh2 x y Q* = *Suc* (*Max* (*insert x* (*insert y* (*fv Q*))))

function (*sequential*) *subst* :: (′*a*, ′*b*) *fmla* ⇒ *nat* ⇒ *nat* ⇒ (′*a*, ′*b*) *fmla* (‹_[_ → _]› [*90*, *0*, *0*] *91*)
where
Bool t[*x* → *y*] = *Bool t*
| *Pred p ts*[*x* → *y*] = *Pred p* (*map* (λ*t*. *t*[*x* →*t y*]) *ts*)
| *Eq z t*[*x* → *y*] = *Eq* (*if z* = *x then y else z*) (*t*[*x* →*t y*])
| *Neg Q*[*x* → *y*] = *Neg* (*Q*[*x* → *y*])
| *Conj Q1 Q2*[*x* → *y*] = *Conj* (*Q1*[*x* → *y*]) (*Q2*[*x* → *y*])
| *Disj Q1 Q2*[*x* → *y*] = *Disj* (*Q1*[*x* → *y*]) (*Q2*[*x* → *y*])
| *Exists z Q*[*x* → *y*] = (*if x* = *z then Exists x Q else*
if z = *y then let z′* = *fresh2 x y Q in Exists z′* (*Q*[*z* → *z′*][*x* → *y*]) *else Exists z* (*Q*[*x* → *y*]))

⟨*proof*⟩

abbreviation *substs* (‹_[_ →* _]› [*90*, *0*, *0*] *91*) **where**
Q[*xs* →* *ys*] ≡ *fold* (λ(*x*, *y*) *Q*. *Q*[*x* → *y*]) (*zip xs ys*) *Q*

lemma *size_subst_p*[*simp*]: *subst_dom* (*Q*, *x*, *y*) ⟹ *size* (*Q*[*x* → *y*]) = *size Q*
⟨*proof*⟩

termination ⟨*proof*⟩

lemma *size_subst*[*simp*]: *size* (*Q*[*x* → *y*]) = *size Q*
⟨*proof*⟩

lemma *fresh2_gt*:
x < *fresh2 x y Q*
y < *fresh2 x y Q*
z ∈ *fv Q* ⟹ *z* < *fresh2 x y Q*
⟨*proof*⟩

lemma *fresh2*:
x ≠ *fresh2 x y Q*
y ≠ *fresh2 x y Q*
fresh2 x y Q ∉ *fv Q*
⟨*proof*⟩

lemma *fv_subst*:
fv (*Q*[*x* → *y*]) = (*if x* ∈ *fv Q then insert y* (*fv Q* − {*x*}) *else fv Q*)
⟨*proof*⟩

lemma *subst_term_triv*: *x* ∉ *fv_term_set t* ⟹ *t*[*x* →*t* *y*] = *t*
⟨*proof*⟩

lemma *subst_exists*: *exists z Q*[*x* → *y*] = (*if z* ∈ *fv Q then if x* = *z then exists x Q else*
if z = *y then let z'* = *fresh2 x y Q in exists z'* (*Q*[*z* → *z'*][*x* → *y*]) *else exists z* (*Q*[*x* → *y*]) *else Q*[*x* → *y*])
⟨*proof*⟩

lemma *eval_subst*[*simp*]: σ · *t*[*x* →*t* *y*] = σ(*x* := σ *y*) · *t*
⟨*proof*⟩

lemma *sat_subst*[*simp*]: *sat* (*Q*[*x* → *y*]) *I* σ = *sat Q I* (σ(*x* := σ *y*))
⟨*proof*⟩

lemma *substs_Bool*[*simp*]: *length xs* = *length ys* ⟹ *Bool b*[*xs* →* *ys*] = *Bool b*
⟨*proof*⟩

lemma *substs_Neg*[*simp*]: *length xs* = *length ys* ⟹ *Neg Q*[*xs* →* *ys*] = *Neg* (*Q*[*xs* →* *ys*])
⟨*proof*⟩

lemma *substs_Conj*[*simp*]: *length xs* = *length ys* ⟹ *Conj Q1 Q2*[*xs* →* *ys*] = *Conj* (*Q1*[*xs* →* *ys*]) (*Q2*[*xs* →* *ys*])
⟨*proof*⟩

lemma *substs_Disj*[*simp*]: *length xs* = *length ys* ⟹ *Disj Q1 Q2*[*xs* →* *ys*] = *Disj* (*Q1*[*xs* →* *ys*]) (*Q2*[*xs* →* *ys*])
⟨*proof*⟩

fun *substs_bd* **where**

substs_bd z (x # xs) (y # ys) Q = (if x = z then substs_bd z xs ys Q else
if z = y then substs_bd (fresh2 x y Q) xs ys (Q[y → fresh2 x y Q][x → y]) else substs_bd z xs ys (Q[x → y]))
| substs_bd z _ _ _ = z

fun substs_src **where**
substs_src z (x # xs) (y # ys) Q = (if x = z then substs_src z xs ys Q else
if z = y then [y, x] @ substs_src (fresh2 x y Q) xs ys (Q[y → fresh2 x y Q][x → y]) else x # substs_src z xs ys (Q[x → y]))
| substs_src _ _ _ _ = []

fun substs_dst **where**
substs_dst z (x # xs) (y # ys) Q = (if x = z then substs_dst z xs ys Q else
if z = y then [fresh2 x y Q, y] @ substs_dst (fresh2 x y Q) xs ys (Q[y → fresh2 x y Q][x → y]) else y # substs_dst z xs ys (Q[x → y]))
| substs_dst _ _ _ _ = []

lemma length_substs[simp]: length xs = length ys ⟹ length (substs_src z xs ys Q) = length (substs_dst z xs ys Q)
⟨proof⟩

lemma substs_Exists[simp]: length xs = length ys ⟹
Exists z Q[xs →* ys] = Exists (substs_bd z xs ys Q) (Q[substs_src z xs ys Q →* substs_dst z xs ys Q])
⟨proof⟩

fun subst_var **where**
subst_var (x # xs) (y # ys) z = (if x = z then subst_var xs ys y else subst_var xs ys z)
| subst_var _ _ z = z

lemma substs_Eq[simp]: length xs = length ys ⟹ (Eq x t)[xs →* ys] = Eq (subst_var xs ys x) (t[xs →t* ys])
⟨proof⟩

lemma substs_term_Var[simp]: length xs = length ys ⟹ (Var x)[xs →t* ys] = Var (subst_var xs ys x)
⟨proof⟩

lemma substs_term_Const[simp]: length xs = length ys ⟹ (Const c)[xs →t* ys] = Const c
⟨proof⟩

lemma in_fv_substs:
length xs = length ys ⟹ x ∈ fv Q ⟹ subst_var xs ys x ∈ fv (Q[xs →* ys])
⟨proof⟩

lemma exists_cp_subst: x ≠ y ⟹ exists x (cp (Q[x → y])) = cp (Q[x → y])
⟨proof⟩

2.6 Generated Variables

inductive ap **where**
Pred: ap (Pred p ts)
| Eqc: ap (Eq x (Const c))

inductive gen **where**
gen x (Bool False) {}
| ap Q ⟹ x ∈ fv Q ⟹ gen x Q {Q}
| gen x Q G ⟹ gen x (Neg (Neg Q)) G
| gen x (Conj (Neg Q1) (Neg Q2)) G ⟹ gen x (Neg (Disj Q1 Q2)) G
| gen x (Disj (Neg Q1) (Neg Q2)) G ⟹ gen x (Neg (Conj Q1 Q2)) G

| $gen\ x\ Q1\ G1 \Longrightarrow gen\ x\ Q2\ G2 \Longrightarrow gen\ x\ (Disj\ Q1\ Q2)\ (G1 \cup G2)$
| $gen\ x\ Q1\ G \vee gen\ x\ Q2\ G \Longrightarrow gen\ x\ (Conj\ Q1\ Q2)\ G$
| $gen\ y\ Q\ G \Longrightarrow gen\ x\ (Conj\ Q\ (x \approx y))\ ((\lambda Q.\ cp\ (Q[y \to x]))\ `\ G)$
| $gen\ y\ Q\ G \Longrightarrow gen\ x\ (Conj\ Q\ (y \approx x))\ ((\lambda Q.\ cp\ (Q[y \to x]))\ `\ G)$
| $x \neq y \Longrightarrow gen\ x\ Q\ G \Longrightarrow\ gen\ x\ (Exists\ y\ Q)\ (exists\ y\ `\ G)$

inductive *gen′* **where**
$gen'\ x\ (Bool\ False)\ \{\}$
| $ap\ Q \Longrightarrow x \in fv\ Q \Longrightarrow gen'\ x\ Q\ \{Q\}$
| $gen'\ x\ Q\ G \Longrightarrow gen'\ x\ (Neg\ (Neg\ Q))\ G$
| $gen'\ x\ (Conj\ (Neg\ Q1)\ (Neg\ Q2))\ G \Longrightarrow gen'\ x\ (Neg\ (Disj\ Q1\ Q2))\ G$
| $gen'\ x\ (Disj\ (Neg\ Q1)\ (Neg\ Q2))\ G \Longrightarrow gen'\ x\ (Neg\ (Conj\ Q1\ Q2))\ G$
| $gen'\ x\ Q1\ G1 \Longrightarrow gen'\ x\ Q2\ G2 \Longrightarrow gen'\ x\ (Disj\ Q1\ Q2)\ (G1 \cup G2)$
| $gen'\ x\ Q1\ G \vee gen'\ x\ Q2\ G \Longrightarrow gen'\ x\ (Conj\ Q1\ Q2)\ G$
| $gen'\ y\ Q\ G \Longrightarrow gen'\ x\ (Conj\ Q\ (x \approx y))\ ((\lambda Q.\ Q[y \to x])\ `\ G)$
| $gen'\ y\ Q\ G \Longrightarrow gen'\ x\ (Conj\ Q\ (y \approx x))\ ((\lambda Q.\ Q[y \to x])\ `\ G)$
| $x \neq y \Longrightarrow gen'\ x\ Q\ G \Longrightarrow\ gen'\ x\ (Exists\ y\ Q)\ (exists\ y\ `\ G)$

inductive *qp* **where**
ap: $ap\ Q \Longrightarrow qp\ Q$
| *exists*: $qp\ Q \Longrightarrow qp\ (exists\ x\ Q)$

lemma *qp_Exists*: $qp\ Q \Longrightarrow x \in fv\ Q \Longrightarrow qp\ (Exists\ x\ Q)$
⟨*proof*⟩

lemma *qp_ExistsE*: $qp\ (Exists\ x\ Q) \Longrightarrow (qp\ Q \Longrightarrow x \in fv\ Q \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

fun *qp_impl* **where**
qp_impl $(Eq\ x\ (Const\ c)) = True$
| *qp_impl* $(Pred\ x\ ts) = True$
| *qp_impl* $(Exists\ x\ Q) = (x \in fv\ Q \wedge qp\ Q)$
| *qp_impl* _ = *False*

lemma *qp_imp_qp_impl*: *qp Q* ⟹ *qp_impl Q*
⟨*proof*⟩

lemma *qp_impl_imp_qp*: *qp_impl Q* ⟹ *qp Q*
⟨*proof*⟩

lemma *qp_code*[*code*]: *qp Q* = *qp_impl Q*
⟨*proof*⟩

lemma *ap_cp*: $ap\ Q \Longrightarrow ap\ (cp\ Q)$
⟨*proof*⟩

lemma *qp_cp*: $qp\ Q \Longrightarrow qp\ (cp\ Q)$
⟨*proof*⟩

lemma *ap_substs*: $ap\ Q \Longrightarrow length\ xs = length\ ys \Longrightarrow ap\ (Q[xs \to^*\ ys])$
⟨*proof*⟩

lemma *ap_subst′*: $ap\ (Q[x \to y]) \Longrightarrow ap\ Q$
⟨*proof*⟩

lemma *qp_substs*: $qp\ Q \Longrightarrow length\ xs = length\ ys \Longrightarrow qp\ (Q[xs \to^*\ ys])$
⟨*proof*⟩

lemma *qp_subst*: $qp\ Q \Longrightarrow qp\ (Q[x \rightarrow y])$
⟨*proof*⟩

lemma *qp_Neg*[*dest*]: $qp\ (Neg\ Q) \Longrightarrow False$
⟨*proof*⟩

lemma *qp_Disj*[*dest*]: $qp\ (Disj\ Q1\ Q2) \Longrightarrow False$
⟨*proof*⟩

lemma *qp_Conj*[*dest*]: $qp\ (Conj\ Q1\ Q2) \Longrightarrow False$
⟨*proof*⟩

lemma *qp_eq*[*dest*]: $qp\ (x \approx y) \Longrightarrow False$
⟨*proof*⟩

lemma *qp_subst'*: $qp\ (Q[x \rightarrow y]) \Longrightarrow qp\ Q$
⟨*proof*⟩

lemma *qp_subst_eq*[*simp*]: $qp\ (Q[x \rightarrow y]) = qp\ Q$
⟨*proof*⟩

lemma *gen_qp*: $gen\ x\ Q\ G \Longrightarrow Qqp \in G \Longrightarrow qp\ Qqp$
⟨*proof*⟩

lemma *gen'_qp*: $gen'\ x\ Q\ G \Longrightarrow Qqp \in G \Longrightarrow qp\ Qqp$
⟨*proof*⟩

lemma *ap_cp_triv*: $ap\ Q \Longrightarrow cp\ Q = Q$
⟨*proof*⟩

lemma *qp_cp_triv*: $qp\ Q \Longrightarrow cp\ Q = Q$
⟨*proof*⟩

lemma *ap_cp_subst_triv*: $ap\ Q \Longrightarrow cp\ (Q[x \rightarrow y]) = Q[x \rightarrow y]$
⟨*proof*⟩

lemma *qp_cp_subst_triv*: $qp\ Q \Longrightarrow cp\ (Q[x \rightarrow y]) = Q[x \rightarrow y]$
⟨*proof*⟩

lemma *gen_nocp_intros*:
$gen\ y\ Q\ G \Longrightarrow gen\ x\ (Conj\ Q\ (x \approx y))\ ((\lambda Q.\ Q[y \rightarrow x])\ `\ G)$
$gen\ y\ Q\ G \Longrightarrow gen\ x\ (Conj\ Q\ (y \approx x))\ ((\lambda Q.\ Q[y \rightarrow x])\ `\ G)$
⟨*proof*⟩

lemma *gen'_cp_intros*:
$gen'\ y\ Q\ G \Longrightarrow gen'\ x\ (Conj\ Q\ (x \approx y))\ ((\lambda Q.\ cp\ (Q[y \rightarrow x]))\ `\ G)$
$gen'\ y\ Q\ G \Longrightarrow gen'\ x\ (Conj\ Q\ (y \approx x))\ ((\lambda Q.\ cp\ (Q[y \rightarrow x]))\ `\ G)$
⟨*proof*⟩

lemma *gen'_gen*: $gen'\ x\ Q\ G \Longrightarrow gen\ x\ Q\ G$
⟨*proof*⟩

lemma *gen_gen'*: $gen\ x\ Q\ G \Longrightarrow gen'\ x\ Q\ G$
⟨*proof*⟩

lemma *gen_eq_gen'*: $gen = gen'$
⟨*proof*⟩

lemmas *gen_induct*[*consumes 1*] = *gen′.induct*[*folded gen_eq_gen′*]

abbreviation *Gen* **where** $Gen\ x\ Q \equiv (\exists\, G.\ gen\ x\ Q\ G)$

lemma *qp_Gen*: $qp\ Q \Longrightarrow x \in fv\ Q \Longrightarrow Gen\ x\ Q$
⟨*proof*⟩

lemma *qp_gen*: $qp\ Q \Longrightarrow x \in fv\ Q \Longrightarrow gen\ x\ Q\ \{Q\}$
⟨*proof*⟩

lemma *gen_foldr_Disj*:
$list\_all2\ (gen\ x)\ Qs\ Gs \Longrightarrow gen\ x\ Q\ G \Longrightarrow GG = G \cup (\bigcup G \in set\ Gs.\ G) \Longrightarrow$
$gen\ x\ (foldr\ Disj\ Qs\ Q)\ GG$
⟨*proof*⟩

lemma *gen_foldr1_Disj*:
$list\_all2\ (gen\ x)\ Qs\ Gs \Longrightarrow gen\ x\ Q\ G \Longrightarrow GG = (if\ Qs = []\ then\ G\ else\ (\bigcup G \in set\ Gs.\ G)) \Longrightarrow$
$gen\ x\ (foldr1\ Disj\ Qs\ Q)\ GG$
⟨*proof*⟩

lemma *gen_Bool_True*[*simp*]: $gen\ x\ (Bool\ True)\ G = False$
⟨*proof*⟩

lemma *gen_Bool_False*[*simp*]: $gen\ x\ (Bool\ False)\ G = (G = \{\})$
⟨*proof*⟩

lemma *gen_Gen_cp*: $gen\ x\ Q\ G \Longrightarrow Gen\ x\ (cp\ Q)$
⟨*proof*⟩

lemma *Gen_cp*: $Gen\ x\ Q \Longrightarrow Gen\ x\ (cp\ Q)$
⟨*proof*⟩

lemma *Gen_DISJ*: $finite\ \mathcal{Q} \Longrightarrow \forall\, Q \in \mathcal{Q}.\ qp\ Q \wedge x \in fv\ Q \Longrightarrow Gen\ x\ (DISJ\ \mathcal{Q})$
⟨*proof*⟩

lemma *Gen_cp_DISJ*: $finite\ \mathcal{Q} \Longrightarrow \forall\, Q \in \mathcal{Q}.\ qp\ Q \wedge x \in fv\ Q \Longrightarrow Gen\ x\ (cp\ (DISJ\ \mathcal{Q}))$
⟨*proof*⟩

lemma *gen_Pred*[*simp*]:
$gen\ z\ (Pred\ p\ ts)\ G \longleftrightarrow z \in fv\_terms\_set\ ts \wedge G = \{Pred\ p\ ts\}$
⟨*proof*⟩

lemma *gen_Eq*[*simp*]:
$gen\ z\ (Eq\ a\ t)\ G \longleftrightarrow z = a \wedge (\exists\, c.\ t = Const\ c \wedge G = \{Eq\ a\ t\})$
⟨*proof*⟩

lemma *gen_empty_cp*: $gen\ z\ Q\ G \Longrightarrow G = \{\} \Longrightarrow cp\ Q = Bool\ False$
⟨*proof*⟩

inductive *genempty* **where**
$genempty\ (Bool\ False)$
$|\ genempty\ Q \Longrightarrow genempty\ (Neg\ (Neg\ Q))$
$|\ genempty\ (Conj\ (Neg\ Q1)\ (Neg\ Q2)) \Longrightarrow genempty\ (Neg\ (Disj\ Q1\ Q2))$
$|\ genempty\ (Disj\ (Neg\ Q1)\ (Neg\ Q2)) \Longrightarrow genempty\ (Neg\ (Conj\ Q1\ Q2))$
$|\ genempty\ Q1 \Longrightarrow genempty\ Q2 \Longrightarrow genempty\ (Disj\ Q1\ Q2)$
$|\ genempty\ Q1 \vee genempty\ Q2 \Longrightarrow genempty\ (Conj\ Q1\ Q2)$
$|\ genempty\ Q \Longrightarrow genempty\ (Conj\ Q\ (x \approx y))$
$|\ genempty\ Q \Longrightarrow genempty\ (Conj\ Q\ (y \approx x))$

| *genempty* $Q \Longrightarrow$ *genempty* (*Exists* y Q)

lemma *gen_genempty*: *gen* z Q $G \Longrightarrow G = \{\} \Longrightarrow$ *genempty* Q
⟨*proof*⟩

lemma *genempty_substs*: *genempty* $Q \Longrightarrow$ *length* *xs* = *length* *ys* $\Longrightarrow$ *genempty* (Q[*xs* $\rightarrow^*$ *ys*])
⟨*proof*⟩

lemma *genempty_substs_Exists*: *genempty* $Q \Longrightarrow$ *length* *xs* = *length* *ys* $\Longrightarrow$ *genempty* (*Exists* y Q[*xs* $\rightarrow^*$ *ys*])
⟨*proof*⟩

lemma *genempty_cp*: *genempty* $Q \Longrightarrow$ *cp* Q = *Bool False*
⟨*proof*⟩

lemma *gen_empty_cp_substs*:
gen x Q $\{\} \Longrightarrow$ *length* *xs* = *length* *ys* $\Longrightarrow$ *cp* (Q[*xs* $\rightarrow^*$ *ys*]) = *Bool False*
⟨*proof*⟩

lemma *gen_empty_cp_substs_Exists*:
gen x Q $\{\} \Longrightarrow$ *length* *xs* = *length* *ys* $\Longrightarrow$ *cp* (*Exists* y Q[*xs* $\rightarrow^*$ *ys*]) = *Bool False*
⟨*proof*⟩

lemma *gen_Gen_substs_Exists*:
length *xs* = *length* *ys* $\Longrightarrow x \neq y \Longrightarrow x \in$ *fv* $Q \Longrightarrow$
($\bigwedge$*xs ys*. *length* *xs* = *length* *ys* $\Longrightarrow$ *Gen* (*subst_var xs ys x*) (*cp* (Q[*xs* $\rightarrow^*$ *ys*]))) $\Longrightarrow$
Gen (*subst_var xs ys x*) (*cp* (*Exists* y Q[*xs* $\rightarrow^*$ *ys*]))
⟨*proof*⟩

lemma *gen_fv*:
gen x Q $G \Longrightarrow$ *Qqp* $\in G \Longrightarrow x \in$ *fv Qqp* $\wedge$ *fv Qqp* $\subseteq$ *fv* Q
⟨*proof*⟩

lemma *gen_sat*:
fixes x :: *nat*
shows *gen* x Q $G \Longrightarrow$ *sat* Q I $\sigma \Longrightarrow \exists$ *Qqp* $\in G$. *sat Qqp* I σ
⟨*proof*⟩

2.7 Variable Erasure

fun *erase* :: ('*a*, '*b*) *fmla* $\Rightarrow$ *nat* $\Rightarrow$ ('*a*, '*b*) *fmla* (**infix** ‹$\perp$› *65*) **where**
Bool $t \perp x$ = *Bool* t
| *Pred p ts* $\perp x$ = (*if* $x \in$ *fv_terms_set ts then Bool False else Pred p ts*)
| *Eq* z $t \perp x$ = (*if* t = *Var* z *then Bool True else*
if $x = z \vee x \in$ *fv_term_set* t *then Bool False else Eq* z t)
| *Neg* $Q \perp x$ = *Neg* ($Q \perp x$)
| *Conj Q1 Q2* $\perp x$ = *Conj* (*Q1* $\perp x$) (*Q2* $\perp x$)
| *Disj Q1 Q2* $\perp x$ = *Disj* (*Q1* $\perp x$) (*Q2* $\perp x$)
| *Exists* z $Q \perp x$ = (*if* $x = z$ *then Exists* x Q *else Exists* z ($Q \perp x$))

lemma *fv_erase*: *fv* ($Q \perp x$) $\subseteq$ *fv* $Q - \{x\}$
⟨*proof*⟩

lemma *ap_cp_erase*: *ap* $Q \Longrightarrow x \in$ *fv* $Q \Longrightarrow$ *cp* ($Q \perp x$) = *Bool False*
⟨*proof*⟩

lemma *qp_cp_erase*: *qp* $Q \Longrightarrow x \in$ *fv* $Q \Longrightarrow$ *cp* ($Q \perp x$) = *Bool False*
⟨*proof*⟩

lemma sat_erase: $sat\ (Q \perp x)\ I\ (\sigma(x := z)) = sat\ (Q \perp x)\ I\ \sigma$
⟨*proof*⟩

lemma $exists\_cp\_erase$: $exists\ x\ (cp\ (Q \perp x)) = cp\ (Q \perp x)$
⟨*proof*⟩

lemma gen_cp_erase:
fixes $x :: nat$
shows $gen\ x\ Q\ G \Longrightarrow Qqp \in G \Longrightarrow cp\ (Qqp \perp x) = Bool\ False$
⟨*proof*⟩

2.8 Generated Variables and Substitutions

lemma $gen\_Gen\_cp\_substs$: $gen\ z\ Q\ G \Longrightarrow length\ xs = length\ ys \Longrightarrow$
$Gen\ (subst\_var\ xs\ ys\ z)\ (cp\ (Q[xs \rightarrow^*\ ys]))$
⟨*proof*⟩

lemma Gen_cp_substs: $Gen\ z\ Q \Longrightarrow length\ xs = length\ ys \Longrightarrow Gen\ (subst\_var\ xs\ ys\ z)\ (cp\ (Q[xs \rightarrow^*\ ys]))$
⟨*proof*⟩

lemma Gen_cp_subst: $Gen\ z\ Q \Longrightarrow z \neq x \Longrightarrow Gen\ z\ (cp\ (Q[x \rightarrow y]))$
⟨*proof*⟩

lemma $substs\_bd\_fv$: $length\ xs = length\ ys \Longrightarrow substs\_bd\ z\ xs\ ys\ Q \in fv\ (Q[substs\_src\ z\ xs\ ys\ Q \rightarrow^*\ substs\_dst\ z\ xs\ ys\ Q]) \Longrightarrow z \in fv\ Q$
⟨*proof*⟩

lemma Gen_substs_bd: $length\ xs = length\ ys \Longrightarrow$
$(\bigwedge xs\ ys.\ length\ xs = length\ ys \Longrightarrow Gen\ (subst\_var\ xs\ ys\ z)\ (cp\ (Qz[xs \rightarrow^*\ ys]))) \Longrightarrow$
$Gen\ (substs\_bd\ z\ xs\ ys\ Qz)\ (cp\ (Qz[substs\_src\ z\ xs\ ys\ Qz \rightarrow^*\ substs\_dst\ z\ xs\ ys\ Qz]))$
⟨*proof*⟩

2.9 Safe-Range Queries

definition *nongens* **where**
$nongens\ Q = \{x \in fv\ Q.\ \neg\ Gen\ x\ Q\}$

abbreviation *rrf* **where**
$rrf\ Q \equiv nongens\ Q = \{\}$

definition *rrb* **where**
$rrb\ Q = (\forall\ y\ Qy.\ Exists\ y\ Qy \in sub\ Q \longrightarrow Gen\ y\ Qy)$

lemma $rrb\_simps[simp]$:
$rrb\ (Bool\ b) = True$
$rrb\ (Pred\ p\ ts) = True$
$rrb\ (Eq\ x\ t) = True$
$rrb\ (Neg\ Q) = rrb\ Q$
$rrb\ (Disj\ Q1\ Q2) = (rrb\ Q1 \wedge rrb\ Q2)$
$rrb\ (Conj\ Q1\ Q2) = (rrb\ Q1 \wedge rrb\ Q2)$
$rrb\ (Exists\ y\ Qy) = (Gen\ y\ Qy \wedge rrb\ Qy)$
$rrb\ (exists\ y\ Qy) = ((y \in fv\ Qy \longrightarrow Gen\ y\ Qy) \wedge rrb\ Qy)$
⟨*proof*⟩

lemma $ap\_rrb[simp]$: $ap\ Q \Longrightarrow rrb\ Q$
⟨*proof*⟩

lemma *qp_rrb*[*simp*]: $qp\ Q \Longrightarrow rrb\ Q$
⟨*proof*⟩

lemma *rrb_cp*: $rrb\ Q \Longrightarrow rrb\ (cp\ Q)$
⟨*proof*⟩

lemma *gen_Gen_erase*: $gen\ x\ Q\ G \Longrightarrow Gen\ x\ (Q \perp z)$
⟨*proof*⟩

lemma *Gen_erase*: $Gen\ x\ Q \Longrightarrow Gen\ x\ (Q \perp z)$
⟨*proof*⟩

lemma *rrb_erase*: $rrb\ Q \Longrightarrow rrb\ (Q \perp x)$
⟨*proof*⟩

lemma *rrb_DISJ*[*simp*]: $finite\ \mathcal{Q} \Longrightarrow (\forall\ Q \in \mathcal{Q}.\ rrb\ Q) \Longrightarrow rrb\ (DISJ\ \mathcal{Q})$
⟨*proof*⟩

lemma *rrb_cp_substs*: $rrb\ Q \Longrightarrow length\ xs = length\ ys \Longrightarrow rrb\ (cp\ (Q[xs \rightarrow^{*} ys]))$
⟨*proof*⟩

lemma *rrb_cp_subst*: $rrb\ Q \Longrightarrow rrb\ (cp\ (Q[x \rightarrow y]))$
⟨*proof*⟩

definition $sr\ Q = (rrf\ Q \wedge rrb\ Q)$

lemma *nongens_cp*: $nongens\ (cp\ Q) \subseteq nongens\ Q$
⟨*proof*⟩

lemma *sr_Disj*: $fv\ Q1 = fv\ Q2 \Longrightarrow sr\ (Disj\ Q1\ Q2) = (sr\ Q1 \wedge sr\ Q2)$
⟨*proof*⟩

lemma *sr_foldr_Disj*: $\forall\ Q' \in set\ Qs.\ fv\ Q' = fv\ Q \Longrightarrow sr\ (foldr\ Disj\ Qs\ Q) \longleftrightarrow (\forall\ Q \in set\ Qs.\ sr\ Q)$
$\wedge\ sr\ Q$
⟨*proof*⟩

lemma *sr_foldr1_Disj*: $\forall\ Q' \in set\ Qs.\ fv\ Q' = X \Longrightarrow sr\ (foldr1\ Disj\ Qs\ Q) \longleftrightarrow (if\ Qs = []\ then\ sr\ Q$
$else\ (\forall\ Q \in set\ Qs.\ sr\ Q))$
⟨*proof*⟩

lemma *sr_False*[*simp*]: $sr\ (Bool\ False)$
⟨*proof*⟩

lemma *sr_cp*: $sr\ Q \Longrightarrow sr\ (cp\ Q)$
⟨*proof*⟩

lemma *sr_DISJ*: $finite\ \mathcal{Q} \Longrightarrow \forall\ Q' \in \mathcal{Q}.\ fv\ Q' = X \Longrightarrow (\forall\ Q \in \mathcal{Q}.\ sr\ Q) \Longrightarrow sr\ (DISJ\ \mathcal{Q})$
⟨*proof*⟩

lemma *sr_Conj_eq*: $sr\ Q \Longrightarrow x \in fv\ Q \vee y \in fv\ Q \Longrightarrow sr\ (Conj\ Q\ (x \approx y))$
⟨*proof*⟩

2.10 Simplification

locale *simplification* =
fixes $simp :: ('a::\{infinite,\ linorder\},\ 'b :: linorder)\ fmla \Rightarrow ('a,\ 'b)\ fmla$
and $simplified :: ('a,\ 'b)\ fmla \Rightarrow bool$
assumes *sat_simp*: $sat\ (simp\ Q)\ I\ \sigma = sat\ Q\ I\ \sigma$

and *fv_simp*: $fv\ (simp\ Q) \subseteq fv\ Q$
and *rrb_simp*: $rrb\ Q \Longrightarrow rrb\ (simp\ Q)$
and *gen_Gen_simp*: $gen\ x\ Q\ G \Longrightarrow Gen\ x\ (simp\ Q)$
and *fv_simp_Disj_same*: $fv\ (simp\ Q1) = X \Longrightarrow fv\ (simp\ Q2) = X \Longrightarrow fv\ (simp\ (Disj\ Q1\ Q2)) = X$
and *simp_False*: $simp\ (Bool\ False) = Bool\ False$
and *simplified_sub*: $simplified\ Q \Longrightarrow Q' \in sub\ Q \Longrightarrow simplified\ Q'$
and *simplified_Conj_eq*: $simplified\ Q \Longrightarrow x \neq y \Longrightarrow x \in fv\ Q \lor y \in fv\ Q \Longrightarrow simplified\ (Conj\ Q\ (x \approx y))$
and *simplified_fv_simp*: $simplified\ Q \Longrightarrow fv\ (simp\ Q) = fv\ Q$
and *simplified_simp*: $simplified\ (simp\ Q)$
and *simplified_cp*: $simplified\ (cp\ Q)$
begin

lemma *Gen_simp*: $Gen\ x\ Q \Longrightarrow Gen\ x\ (simp\ Q)$
⟨*proof*⟩

lemma *nongens_simp*: $nongens\ (simp\ Q) \subseteq nongens\ Q$
⟨*proof*⟩

lemma *sr_simp*: $sr\ Q \Longrightarrow sr\ (simp\ Q)$
⟨*proof*⟩

lemma *equiv_simp_cong*: $Q \triangleq Q' \Longrightarrow simp\ Q \triangleq simp\ Q'$
⟨*proof*⟩

lemma *equiv_simp*: $simp\ Q \triangleq Q$
⟨*proof*⟩

lemma *fv_simp_foldr_Disj*: $\forall\ Q \in set\ Qs \cup \{Q\}.\ simplified\ Q \land fv\ Q = A \Longrightarrow$
$fv\ (simp\ (foldr\ Disj\ Qs\ Q)) = A$
⟨*proof*⟩

lemma *fv_simp_foldr1_Disj*: $simp\ (foldr1\ Disj\ Qs\ (Bool\ False)) \neq Bool\ False \Longrightarrow$
$\forall\ Q \in set\ Qs.\ simplified\ Q \land fv\ Q = A \Longrightarrow$
$fv\ (simp\ (foldr1\ Disj\ Qs\ (Bool\ False))) = A$
⟨*proof*⟩

lemma *fv_simp_DISJ_eq*:
$finite\ \mathcal{Q} \Longrightarrow simp\ (DISJ\ \mathcal{Q}) \neq Bool\ False \Longrightarrow \forall\ Q \in \mathcal{Q}.\ simplified\ Q \land fv\ Q = A \Longrightarrow fv\ (simp\ (DISJ\ \mathcal{Q})) = A$
⟨*proof*⟩

end

2.11 Covered Variables

inductive *cov* **where**
Eq_self: $cov\ x\ (x \approx x)\ \{\}$
| *nonfree*: $x \notin fv\ Q \Longrightarrow cov\ x\ Q\ \{\}$
| *EqL*: $x \neq y \Longrightarrow cov\ x\ (x \approx y)\ \{x \approx y\}$
| *EqR*: $x \neq y \Longrightarrow cov\ x\ (y \approx x)\ \{x \approx y\}$
| *ap*: $ap\ Q \Longrightarrow x \in fv\ Q \Longrightarrow cov\ x\ Q\ \{Q\}$
| *Neg*: $cov\ x\ Q\ G \Longrightarrow cov\ x\ (Neg\ Q)\ G$
| *Disj*: $cov\ x\ Q1\ G1 \Longrightarrow cov\ x\ Q2\ G2 \Longrightarrow cov\ x\ (Disj\ Q1\ Q2)\ (G1 \cup G2)$
| *DisjL*: $cov\ x\ Q1\ G \Longrightarrow cp\ (Q1 \perp x) = Bool\ True \Longrightarrow cov\ x\ (Disj\ Q1\ Q2)\ G$
| *DisjR*: $cov\ x\ Q2\ G \Longrightarrow cp\ (Q2 \perp x) = Bool\ True \Longrightarrow cov\ x\ (Disj\ Q1\ Q2)\ G$
| *Conj*: $cov\ x\ Q1\ G1 \Longrightarrow cov\ x\ Q2\ G2 \Longrightarrow cov\ x\ (Conj\ Q1\ Q2)\ (G1 \cup G2)$

| *ConjL*: *cov x Q1 G* ⟹ *cp* (*Q1* ⊥ *x*) = *Bool False* ⟹ *cov x* (*Conj Q1 Q2*) *G*
| *ConjR*: *cov x Q2 G* ⟹ *cp* (*Q2* ⊥ *x*) = *Bool False* ⟹ *cov x* (*Conj Q1 Q2*) *G*
| *Exists*: *x* ≠ *y* ⟹ *cov x Q G* ⟹ *x* ≈ *y* ∉ *G* ⟹ *cov x* (*Exists y Q*) (*exists y* ‘ *G*)
| *Exists_gen*: *x* ≠ *y* ⟹ *cov x Q G* ⟹ *gen y Q Gy* ⟹ *cov x* (*Exists y Q*) ((*exists y* ‘ (*G* − {*x* ≈ *y*})) ∪ ((λ*Q*. *cp* (*Q*[*y* → *x*])) ‘ *Gy*))

inductive *cov′* **where**
Eq_self: *cov′ x* (*x* ≈ *x*) {}
| *nonfree*: *x* ∉ *fv Q* ⟹ *cov′ x Q* {}
| *EqL*: *x* ≠ *y* ⟹ *cov′ x* (*x* ≈ *y*) {*x* ≈ *y*}
| *EqR*: *x* ≠ *y* ⟹ *cov′ x* (*y* ≈ *x*) {*x* ≈ *y*}
| *ap*: *ap Q* ⟹ *x* ∈ *fv Q* ⟹ *cov′ x Q* {*Q*}
| *Neg*: *cov′ x Q G* ⟹ *cov′ x* (*Neg Q*) *G*
| *Disj*: *cov′ x Q1 G1* ⟹ *cov′ x Q2 G2* ⟹ *cov′ x* (*Disj Q1 Q2*) (*G1* ∪ *G2*)
| *DisjL*: *cov′ x Q1 G* ⟹ *cp* (*Q1* ⊥ *x*) = *Bool True* ⟹ *cov′ x* (*Disj Q1 Q2*) *G*
| *DisjR*: *cov′ x Q2 G* ⟹ *cp* (*Q2* ⊥ *x*) = *Bool True* ⟹ *cov′ x* (*Disj Q1 Q2*) *G*
| *Conj*: *cov′ x Q1 G1* ⟹ *cov′ x Q2 G2* ⟹ *cov′ x* (*Conj Q1 Q2*) (*G1* ∪ *G2*)
| *ConjL*: *cov′ x Q1 G* ⟹ *cp* (*Q1* ⊥ *x*) = *Bool False* ⟹ *cov′ x* (*Conj Q1 Q2*) *G*
| *ConjR*: *cov′ x Q2 G* ⟹ *cp* (*Q2* ⊥ *x*) = *Bool False* ⟹ *cov′ x* (*Conj Q1 Q2*) *G*
| *Exists*: *x* ≠ *y* ⟹ *cov′ x Q G* ⟹ *x* ≈ *y* ∉ *G* ⟹ *cov′ x* (*Exists y Q*) (*exists y* ‘ *G*)
| *Exists_gen*: *x* ≠ *y* ⟹ *cov′ x Q G* ⟹ *gen y Q Gy* ⟹ *cov′ x* (*Exists y Q*) ((*exists y* ‘ (*G* − {*x* ≈ *y*})) ∪ ((λ*Q*. *Q*[*y* → *x*]) ‘ *Gy*))

lemma *cov_nocp_intros*:
x ≠ *y* ⟹ *cov x Q G* ⟹ *gen y Q Gy* ⟹ *cov x* (*Exists y Q*) ((*exists y* ‘ (*G* − {*x* ≈ *y*})) ∪ ((λ*Q*. *Q*[*y* → *x*]) ‘ *Gy*))
⟨*proof*⟩

lemma *cov′_cp_intros*:
x ≠ *y* ⟹ *cov′ x Q G* ⟹ *gen y Q Gy* ⟹ *cov′ x* (*Exists y Q*) ((*exists y* ‘ (*G* − {*x* ≈ *y*})) ∪ ((λ*Q*. *cp* (*Q*[*y* → *x*])) ‘ *Gy*))
⟨*proof*⟩

lemma *cov′_cov*: *cov′ x Q G* ⟹ *cov x Q G*
⟨*proof*⟩

lemma *cov_cov′*: *cov x Q G* ⟹ *cov′ x Q G*
⟨*proof*⟩

lemma *cov_eq_cov′*: *cov* = *cov′*
⟨*proof*⟩

lemmas *cov_induct*[*consumes 1*, *case_names Eq_self nonfree EqL EqR ap Neg Disj DisjL DisjR Conj ConjL ConjR Exists Exists_gen*] =
cov′.induct[*folded cov_eq_cov′*]

lemma *ex_cov*: *rrb Q* ⟹ *x* ∈ *fv Q* ⟹ ∃ *G*. *cov x Q G*
⟨*proof*⟩

definition *qps* **where**
qps G = {*Q* ∈ *G*. *qp Q*}

lemma *qps_qp*: *Q* ∈ *qps G* ⟹ *qp Q*
⟨*proof*⟩

lemma *qps_in*: *Q* ∈ *qps G* ⟹ *Q* ∈ *G*
⟨*proof*⟩

lemma *qps_empty*[*simp*]: *qps* {} = {}
⟨*proof*⟩

lemma *qps_insert*: *qps* (*insert Q Qs*) = (*if qp Q then insert Q* (*qps Qs*) *else qps Qs*)
⟨*proof*⟩

lemma *qps_union*[*simp*]: *qps* (*X* ∪ *Y*) = *qps X* ∪ *qps Y*
⟨*proof*⟩

lemma *finite_qps*[*simp*]: *finite G* ⟹ *finite* (*qps G*)
⟨*proof*⟩

lemma *qps_exists*[*simp*]: *x* ≠ *y* ⟹ *qps* (*exists y* ‘ *G*) = *exists y* ‘ *qps G*
⟨*proof*⟩

lemma *qps_subst*[*simp*]: *qps* ((λ*Q*. *Q*[*x* → *y*]) ‘ *G*) = (λ*Q*. *Q*[*x* → *y*]) ‘ *qps G*
⟨*proof*⟩

lemma *qps_minus*[*simp*]: *qps* (*G* − {*x* ≈ *y*}) = *qps G*
⟨*proof*⟩

lemma *gen_qps*[*simp*]: *gen x Q G* ⟹ *qps G* = *G*
⟨*proof*⟩

lemma *qps_rrb*[*simp*]: *Q* ∈ *qps G* ⟹ *rrb Q*
⟨*proof*⟩

definition *eqs* **where**
eqs x G = {*y*. *x* ≠ *y* ∧ *x* ≈ *y* ∈ *G*}

lemma *eqs_in*: *y* ∈ *eqs x G* ⟹ *x* ≈ *y* ∈ *G*
⟨*proof*⟩

lemma *eqs_noteq*: *y* ∈ *eqs x Q* ⟹ *x* ≠ *y*
⟨*proof*⟩

lemma *eqs_empty*[*simp*]: *eqs x* {} = {}
⟨*proof*⟩

lemma *eqs_union*[*simp*]: *eqs x* (*X* ∪ *Y*) = *eqs x X* ∪ *eqs x Y*
⟨*proof*⟩

lemma *finite_eqs*[*simp*]: *finite G* ⟹ *finite* (*eqs x G*)
⟨*proof*⟩

lemma *eqs_exists*[*simp*]: *x* ≠ *y* ⟹ *eqs x* (*exists y* ‘ *G*) = *eqs x G* − {*y*}
⟨*proof*⟩

lemma *notin_eqs*[*simp*]: *x* ≈ *y* ∉ *G* ⟹ *y* ∉ *eqs x G*
⟨*proof*⟩

lemma *eqs_minus*[*simp*]: *eqs x* (*G* − {*x* ≈ *y*}) = *eqs x G* − {*y*}
⟨*proof*⟩

lemma *Var_eq_subst_iff*: *Var z* = *t*[*x* →*t y*] ⟷ (*if z* = *x then x* = *y* ∧ *t* = *Var x else*
if z = *y then t* = *Var x* ∨ *t* = *Var y else t* = *Var z*)
⟨*proof*⟩

lemma *Eq_eq_subst_iff*: $y \approx z = Q[x \to y] \longleftrightarrow$ (*if* $z = x$ *then* $x = y \wedge Q = x \approx x$ *else*
$Q = x \approx z \vee Q = y \approx z \vee (z = y \wedge Q \in \{x \approx x,\ y \approx y,\ y \approx x\})$)
⟨*proof*⟩

lemma *eqs_subst*[*simp*]: $x \neq y \Longrightarrow$ *eqs* y $((\lambda Q.\ Q[x \to y])$ ‘ $G) = ($*eqs* y $G - \{x\}) \cup ($*eqs* x $G - \{y\})$
⟨*proof*⟩

lemma *gen_eqs*[*simp*]: *gen* x Q $G \Longrightarrow$ *eqs* z $G = \{\}$
⟨*proof*⟩

lemma *eqs_insert*: *eqs* x (*insert* Q *Qs*) = (*case* Q *of* $z \approx y \Rightarrow$
if $z = x \wedge z \neq y$ *then insert* y (*eqs* x *Qs*) *else eqs* x *Qs* | _ $\Rightarrow$ *eqs* x *Qs*)
⟨*proof*⟩

lemma *eqs_insert'*: $y \neq x \Longrightarrow$ *eqs* x (*insert* $(x \approx y)$ *Qs*) = *insert* y (*eqs* x *Qs*)
⟨*proof*⟩

lemma *eqs_code*[*code*]: *eqs* x G = (λ*eq*. *case eq of* $y \approx z \Rightarrow z$) ‘ (*Set.filter* ($\lambda$*eq*. *case eq of* $y \approx z \Rightarrow x =$
$y \wedge x \neq z$ | _ => *False*) G)
⟨*proof*⟩

lemma *gen_finite*[*simp*]: *gen* x Q $G \Longrightarrow$ *finite* G
⟨*proof*⟩

lemma *cov_finite*[*simp*]: *cov* x Q $G \Longrightarrow$ *finite* G
⟨*proof*⟩

lemma *gen_sat_erase*: *gen* y Q *Gy* $\Longrightarrow$ *sat* $(Q \perp x)$ I $\sigma \Longrightarrow \exists\, Q{\in}Gy.$ *sat* Q I σ
⟨*proof*⟩

lemma *cov_sat_erase*: *cov* x Q $G \Longrightarrow$
sat (*Neg* (*Disj* (*DISJ* (*qps* G)) (*DISJ* $((\lambda y.\ x \approx y)$ ‘ *eqs* x G)))) I $\sigma \Longrightarrow$
sat Q I $\sigma \longleftrightarrow$ *sat* (*cp* $(Q \perp x)$) I σ
⟨*proof*⟩

lemma *cov_fv_aux*: *cov* x Q $G \Longrightarrow$ *Qqp* $\in G \Longrightarrow x \in$ *fv Qqp* $\wedge$ *fv Qqp* $- \{x\} \subseteq$ *fv* Q
⟨*proof*⟩

lemma *cov_fv*: *cov* x Q $G \Longrightarrow x \in$ *fv* $Q \Longrightarrow$ *Qqp* $\in G \Longrightarrow x \in$ *fv Qqp* $\wedge$ *fv Qqp* $\subseteq$ *fv* Q
⟨*proof*⟩

lemma *Gen_Conj*:
Gen x *Q1* $\Longrightarrow$ *Gen* x (*Conj Q1 Q2*)
Gen x *Q2* $\Longrightarrow$ *Gen* x (*Conj Q1 Q2*)
⟨*proof*⟩

lemma *cov_Gen_qps*: *cov* x Q $G \Longrightarrow x \in$ *fv* $Q \Longrightarrow$ *Gen* x (*Conj* Q (*DISJ* (*qps* G)))
⟨*proof*⟩

lemma *cov_equiv*:
assumes *cov* x Q G $\bigwedge Q\ I\ \sigma.$ *sat* (*simp* Q) I σ = *sat* Q I σ
shows $Q \triangleq$ *Disj* (*simp* (*Conj* Q (*DISJ* (*qps* G))))
(*Disj* (*DISJ* $((\lambda y.$ *Conj* (*cp* $(Q[x \to y])$) $(x \approx y))$ ‘ *eqs* x G))
(*Conj* $(Q \perp x)$ (*Neg* (*Disj* (*DISJ* (*qps* G)) (*DISJ* $((\lambda y.\ x \approx y)$ ‘ *eqs* x G))))))
(**is** _ $\triangleq$ *?rhs*)
⟨*proof*⟩

fun *csts_term* **where**

csts_term (*Var x*) = {}
| *csts_term* (*Const c*) = {*c*}

fun *csts* **where**
csts (*Bool b*) = {}
| *csts* (*Pred p ts*) = ($\bigcup$ *t* ∈ *set ts*. *csts_term t*)
| *csts* (*Eq x t*) = *csts_term t*
| *csts* (*Neg Q*) = *csts Q*
| *csts* (*Conj Q1 Q2*) = *csts Q1* ∪ *csts Q2*
| *csts* (*Disj Q1 Q2*) = *csts Q1* ∪ *csts Q2*
| *csts* (*Exists x Q*) = *csts Q*

lemma *finite_csts_term*[*simp*]: *finite* (*csts_term t*)
⟨*proof*⟩

lemma *finite_csts*[*simp*]: *finite* (*csts t*)
⟨*proof*⟩

lemma *ap_fresh_val*: *ap Q* ⟹ *σ x* ∉ *adom I* ⟹ *σ x* ∉ *csts Q* ⟹ *sat Q I σ* ⟹ *x* ∉ *fv Q*
⟨*proof*⟩

lemma *qp_fresh_val*: *qp Q* ⟹ *σ x* ∉ *adom I* ⟹ *σ x* ∉ *csts Q* ⟹ *sat Q I σ* ⟹ *x* ∉ *fv Q*
⟨*proof*⟩

lemma *ex_fresh_val*:
fixes *Q* :: (*'a* :: *infinite*, *'b*) *fmla*
assumes *finite* (*adom I*) *finite A*
shows ∃ *x*. *x* ∉ *adom I* ∧ *x* ∉ *csts Q* ∧ *x* ∉ *A*
⟨*proof*⟩

definition *fresh_val* :: (*'a* :: *infinite*, *'b*) *fmla* ⇒ (*'a*, *'b*) *intp* ⇒ *'a set* ⇒ *'a* **where**
fresh_val Q I A = (*SOME x*. *x* ∉ *adom I* ∧ *x* ∉ *csts Q* ∧ *x* ∉ *A*)

lemma *fresh_val*:
finite (*adom I*) ⟹ *finite A* ⟹ *fresh_val Q I A* ∉ *adom I*
finite (*adom I*) ⟹ *finite A* ⟹ *fresh_val Q I A* ∉ *csts Q*
finite (*adom I*) ⟹ *finite A* ⟹ *fresh_val Q I A* ∉ *A*
⟨*proof*⟩

lemma *csts_exists*[*simp*]: *csts* (*exists x Q*) = *csts Q*
⟨*proof*⟩

lemma *csts_term_subst_term*[*simp*]: *csts_term* (*t*[*x* →*t y*]) = *csts_term t*
⟨*proof*⟩

lemma *csts_subst*[*simp*]: *csts* (*Q*[*x* → *y*]) = *csts Q*
⟨*proof*⟩

lemma *gen_csts*: *gen x Q G* ⟹ *Qqp* ∈ *G* ⟹ *csts Qqp* ⊆ *csts Q*
⟨*proof*⟩

lemma *cov_csts*: *cov x Q G* ⟹ *Qqp* ∈ *G* ⟹ *csts Qqp* ⊆ *csts Q*
⟨*proof*⟩

lemma *not_self_eqs*[*simp*]: *x* ∉ *eqs x G*
⟨*proof*⟩

lemma (**in** *simplification*) *cov_Exists_equiv*:

fixes Q :: ($'a$ :: {*infinite*, *linorder*}, $'b$ :: *linorder*) *fmla*
assumes *cov x Q G x* $\in$ *fv Q*
shows *Exists x Q* $\triangleq$ *Disj* (*Exists x* (*simp* (*Conj Q* (*DISJ* (*qps G*)))))
(*Disj* (*DISJ* ((λy. *cp* ($Q[x \to y]$)) ' *eqs x G*)) (*cp* ($Q \perp x$)))
⟨*proof*⟩

definition *eval_on V Q I* =
(*let xs* = *sorted_list_of_set V*
in {*ds*. *length xs* = *length ds* $\wedge$ ($\exists\sigma$. *sat Q I* ($\sigma[xs :=^* ds]$))})

definition *eval Q I* = *eval_on* (*fv Q*) *Q I*

lemmas *eval_deep_def* = *eval_def*[*unfolded eval_on_def*]

lemma (**in** *simplification*) *cov_eval_fin*:
fixes Q :: ($'a$ :: {*infinite*, *linorder*}, $'b$ :: *linorder*) *fmla*
assumes *cov x Q G x* $\in$ *fv Q finite* (*adom I*) $\bigwedge\sigma$. $\neg$ *sat* ($Q \perp x$) *I* σ
shows *eval Q I* = *eval_on* (*fv Q*) (*Disj* (*simp* (*Conj Q* (*DISJ* (*qps G*))))
(*DISJ* ((λy. *Conj* (*cp* ($Q[x \to y]$)) ($x \approx y$)) ' *eqs x G*))) *I*
(**is** *eval Q I* = *eval_on* (*fv Q*) *?Q I*)
⟨*proof*⟩

lemma (**in** *simplification*) *cov_sat_fin*:
fixes Q :: ($'a$ :: {*infinite*, *linorder*}, $'b$ :: *linorder*) *fmla*
assumes *cov x Q G x* $\in$ *fv Q finite* (*adom I*) $\bigwedge\sigma$. $\neg$ *sat* ($Q \perp x$) *I* σ
shows *sat Q I* σ = *sat* (*Disj* (*simp* (*Conj Q* (*DISJ* (*qps G*))))
(*DISJ* ((λy. *Conj* (*cp* ($Q[x \to y]$)) ($x \approx y$)) ' *eqs x G*))) *I* σ
(**is** *sat Q I* σ = *sat ?Q I* σ)
⟨*proof*⟩

lemma *equiv_eval_eqI*: *finite* (*adom I*) $\Longrightarrow$ *fv Q* = *fv Q'* $\Longrightarrow$ $Q \triangleq Q'$ $\Longrightarrow$ *eval Q I* = *eval Q' I*
⟨*proof*⟩

lemma *equiv_eval_on_eqI*: *finite* (*adom I*) $\Longrightarrow$ $Q \triangleq Q'$ $\Longrightarrow$ *eval_on X Q I* = *eval_on X Q' I*
⟨*proof*⟩

lemma *equiv_eval_on_eval_eqI*: *finite* (*adom I*) $\Longrightarrow$ *fv Q* $\subseteq$ *fv Q'* $\Longrightarrow$ $Q \triangleq Q'$ $\Longrightarrow$ *eval_on* (*fv Q'*) *Q I* = *eval Q' I*
⟨*proof*⟩

lemma *finite_eval_on_Disj2D*:
assumes *finite X*
shows *finite* (*eval_on X* (*Disj Q1 Q2*) *I*) $\Longrightarrow$ *finite* (*eval_on X Q2 I*)
⟨*proof*⟩

lemma *finite_eval_Disj2D*: *finite* (*eval* (*Disj Q1 Q2*) *I*) $\Longrightarrow$ *finite* (*eval Q2 I*)
⟨*proof*⟩

lemma *infinite_eval_Disj2*:
fixes *Q1 Q2* :: ($'a$ :: {*infinite*, *linorder*}, $'b$ :: *linorder*) *fmla*
assumes *fv Q2* $\subset$ *fv* (*Disj Q1 Q2*) *sat Q2 I* σ
shows *infinite* (*eval* (*Disj Q1 Q2*) *I*)
⟨*proof*⟩

lemma *infinite_eval_on_Disj2*:
fixes *Q1 Q2* :: ($'a$ :: {*infinite*, *linorder*}, $'b$ :: *linorder*) *fmla*
assumes *fv Q2* $\subset$ *X fv Q1* $\subseteq$ *Xfinite X sat Q2 I* σ
shows *infinite* (*eval_on X* (*Disj Q1 Q2*) *I*)

⟨proof⟩

lemma *cov_eval_inf*:
  fixes Q :: ('a :: {infinite, linorder}, 'b :: linorder) fmla
  assumes *cov x Q G* $x \in$ *fv Q finite* (*adom I*) *sat* $(Q \perp x)$ I σ
  shows *infinite* (*eval Q I*)
⟨proof⟩

2.12 More on Evaluation

lemma *eval_Bool_False*[*simp*]: *eval* (*Bool False*) $I = \{\}$
  ⟨proof⟩

lemma *eval_on_False*[*simp*]: *eval_on X* (*Bool False*) $I = \{\}$
  ⟨proof⟩

lemma *eval_DISJ_prune_unsat*: *finite B* $\Longrightarrow A \subseteq B \Longrightarrow \forall Q \in B - A.\ \forall \sigma.\ \neg$ *sat Q I* $\sigma \Longrightarrow$ *eval_on X* (*DISJ A*) I = *eval_on X* (*DISJ B*) I
  ⟨proof⟩

lemma *eval_DISJ*: *finite* $\mathcal{Q} \Longrightarrow \forall Q \in \mathcal{Q}.$ *fv* $Q = A \Longrightarrow$ *eval_on A* (*DISJ* $\mathcal{Q}$) $I = (\bigcup Q \in \mathcal{Q}.$ *eval Q I*)
  ⟨proof⟩

lemma *eval_cp_DISJ_closed*: *finite* $\mathcal{Q} \Longrightarrow \forall Q \in \mathcal{Q}.$ *fv* $Q = \{\} \Longrightarrow$ *eval* (*cp* (*DISJ* $\mathcal{Q}$)) $I = (\bigcup Q \in \mathcal{Q}.$ *eval Q I*)
  ⟨proof⟩

lemma (**in** *simplification*) *eval_simp_DISJ_closed*: *finite* $\mathcal{Q} \Longrightarrow \forall Q \in \mathcal{Q}.$ *fv* $Q = \{\} \Longrightarrow$ *eval* (*simp* (*DISJ* $\mathcal{Q}$)) $I = (\bigcup Q \in \mathcal{Q}.$ *eval Q I*)
  ⟨proof⟩

lemma *eval_cong*: *fv* Q = *fv* $Q' \Longrightarrow (\bigwedge\sigma.$ *sat Q I* σ = *sat* Q' I $\sigma) \Longrightarrow$ *eval Q I* = *eval* Q' I
  ⟨proof⟩

lemma *eval_on_cong*: $(\bigwedge\sigma.$ *sat Q I* σ = *sat* Q' I $\sigma) \Longrightarrow$ *eval_on X Q I* = *eval_on X* Q' I
  ⟨proof⟩

lemma *eval_empty_alt*: *eval Q I* $= \{\} \longleftrightarrow (\forall \sigma.\ \neg$ *sat Q I* $\sigma)$
⟨proof⟩

lemma *sat_EXISTS*: *distinct xs* $\Longrightarrow$ *sat* (*EXISTS xs Q*) I $\sigma = (\exists$ *ds*. *length ds* = *length xs* $\wedge$ *sat Q I* $(\sigma[xs :=^* ds]))$
⟨proof⟩

lemma *eval_empty_close*: *eval* (*close Q*) $I = \{\} \longleftrightarrow (\forall \sigma.\ \neg$ *sat Q I* $\sigma)$
  ⟨proof⟩

lemma *infinite_eval_on_extra_variables*:
  assumes *finite X fv* (Q :: ('a :: infinite, 'b) fmla) $\subset X$ $\exists \sigma.$ *sat Q I* σ
  shows *infinite* (*eval_on X Q I*)
⟨proof⟩

lemma *eval_on_cp*: *eval_on X* (*cp Q*) = *eval_on X Q*
  ⟨proof⟩

lemma (**in** *simplification*) *eval_on_simp*: *eval_on X* (*simp Q*) = *eval_on X Q*
  ⟨proof⟩

lemma (**in** *simplification*) *eval_simp_False*: $eval\ (simp\ (Bool\ False))\ I = \{\}$
⟨*proof*⟩

abbreviation $idx\_of\_var\ x\ Q \equiv index\ (sorted\_list\_of\_set\ (fv\ Q))\ x$

lemma *evalE*: $ds \in eval\ Q\ I \Longrightarrow (\bigwedge\sigma.\ length\ ds = card\ (fv\ Q) \Longrightarrow sat\ Q\ I\ (\sigma[sorted\_list\_of\_set\ (fv\ Q) :=^*\ ds]) \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

lemma *infinite_eval_Conj*:
assumes $x \notin fv\ Q$ $infinite\ (eval\ Q\ I)$
shows $infinite\ (eval\ (Conj\ Q\ (x \approx y))\ I)$
(**is** $infinite\ (eval\ ?Qxy\ I)$)
⟨*proof*⟩

lemma *infinite_Implies_mono_on*: $infinite\ (eval\_on\ X\ Q\ I) \Longrightarrow finite\ X \Longrightarrow (\bigwedge\sigma.\ sat\ (Impl\ Q\ Q')\ I\ \sigma) \Longrightarrow infinite\ (eval\_on\ X\ Q'\ I)$
⟨*proof*⟩

3 Restricting Bound Variables

fun *flat_Disj* **where**
$flat\_Disj\ (Disj\ Q1\ Q2) = flat\_Disj\ Q1 \cup flat\_Disj\ Q2$
$|\ flat\_Disj\ Q = \{Q\}$

lemma *finite_flat_Disj*[*simp*]: $finite\ (flat\_Disj\ Q)$
⟨*proof*⟩

lemma *DISJ_flat_Disj*: $DISJ\ (flat\_Disj\ Q) \triangleq Q$
⟨*proof*⟩

lemma *fv_flat_Disj*: $(\bigcup Q' \in flat\_Disj\ Q.\ fv\ Q') = fv\ Q$
⟨*proof*⟩

lemma *fv_flat_DisjD*: $Q' \in flat\_Disj\ Q \Longrightarrow x \in fv\ Q' \Longrightarrow x \in fv\ Q$
⟨*proof*⟩

lemma *cpropagated_flat_DisjD*: $Q' \in flat\_Disj\ Q \Longrightarrow cpropagated\ Q \Longrightarrow cpropagated\ Q'$
⟨*proof*⟩

lemma *flat_Disj_sub*: $flat\_Disj\ Q \subseteq sub\ Q$
⟨*proof*⟩

lemma (**in** *simplification*) *simplified_flat_DisjD*: $Q' \in flat\_Disj\ Q \Longrightarrow simplified\ Q \Longrightarrow simplified\ Q'$
⟨*proof*⟩

definition *fixbound* **where**
$fixbound\ \mathcal{Q}\ x = \{Q \in \mathcal{Q}.\ x \in nongens\ Q\}$

definition (**in** *simplification*) *rb_spec* **where**
$rb\_spec\ Q = SPEC\ (\lambda Q'.\ rrb\ Q' \wedge simplified\ Q' \wedge Q \triangleq Q' \wedge fv\ Q' \subseteq fv\ Q)$

definition (**in** *simplification*) *rb_INV* **where**
$rb\_INV\ x\ Q\ \mathcal{Q} = (finite\ \mathcal{Q}\ \wedge$
$Exists\ x\ Q \triangleq DISJ\ (exists\ x\ `\ \mathcal{Q})\ \wedge$
$(\forall\ Q' \in \mathcal{Q}.\ rrb\ Q' \wedge fv\ Q' \subseteq fv\ Q \wedge simplified\ Q'))$

lemma (**in** *simplification*) *rb_INV_I*:

finite 𝒬 ⟹ Exists x Q ≜ DISJ (exists x ‘ 𝒬) ⟹ (⋀Q′. Q′ ∈ 𝒬 ⟹ rrb Q′) ⟹
(⋀Q′. Q′ ∈ 𝒬 ⟹ fv Q′ ⊆ fv Q) ⟹ (⋀Q′. Q′ ∈ 𝒬 ⟹ simplified Q′) ⟹ rb_INV x Q 𝒬
⟨proof⟩

fun (**in** simplification) rb :: ('a :: {infinite, linorder}, 'b :: linorder) fmla ⇒ ('a, 'b) fmla nres **where**
rb (Neg Q) = do { Q′ ← rb Q; RETURN (simp (Neg Q′))}
| rb (Disj Q1 Q2) = do { Q1′ ← rb Q1; Q2′ ← rb Q2; RETURN (simp (Disj Q1′ Q2′))}
| rb (Conj Q1 Q2) = do { Q1′ ← rb Q1; Q2′ ← rb Q2; RETURN (simp (Conj Q1′ Q2′))}
| rb (Exists x Q) = do {
Q′ ← rb Q;
𝒬 ← $WHILE_T^{rb\_INV\ x\ Q'}$
(λ𝒬. fixbound 𝒬 x ≠ {}) (λ𝒬. do {
Qfix ← RES (fixbound 𝒬 x);
G ← SPEC (cov x Qfix);
RETURN (𝒬 − {Qfix} ∪
{simp (Conj Qfix (DISJ (qps G)))} ∪
(⋃y ∈ eqs x G. {cp (Qfix[x → y])}) ∪
{cp (Qfix ⊥ x)}})
(flat_Disj Q′);
RETURN (simp (DISJ (exists x ‘ 𝒬)))}
| rb Q = do { RETURN (simp Q) }

lemma (**in** simplification) cov_fixbound: cov x Q G ⟹ x ∈ fv Q ⟹
fixbound (insert (cp (Q ⊥ x)) (insert (simp (Conj Q (DISJ (qps G))))
(𝒬 − {Q} ∪ ((λy. cp (Q[x → y])) ‘ eqs x G)))) x = fixbound 𝒬 x − {Q}
⟨proof⟩

lemma finite_fixbound[simp]: finite 𝒬 ⟹ finite (fixbound 𝒬 x)
⟨proof⟩

lemma fixboundE[elim_format]: Q ∈ fixbound 𝒬 x ⟹ x ∈ fv Q ∧ Q ∈ 𝒬 ∧ ¬ Gen x Q
⟨proof⟩

lemma fixbound_fv: Q ∈ fixbound 𝒬 x ⟹ x ∈ fv Q
⟨proof⟩

lemma fixbound_in: Q ∈ fixbound 𝒬 x ⟹ Q ∈ 𝒬
⟨proof⟩

lemma fixbound_empty_Gen: fixbound 𝒬 x = {} ⟹ x ∈ fv Q ⟹ Q ∈ 𝒬 ⟹ Gen x Q
⟨proof⟩

lemma fixbound_insert:
fixbound (insert Q 𝒬) x = (if Gen x Q ∨ x ∉ fv Q then fixbound 𝒬 x else insert Q (fixbound 𝒬 x))
⟨proof⟩

lemma fixbound_empty[simp]:
fixbound {} x = {}
⟨proof⟩

lemma flat_Disj_Exists_sub: Q′ ∈ flat_Disj Q ⟹ Exists y Qy ∈ sub Q′ ⟹ Exists y Qy ∈ sub Q
⟨proof⟩

lemma rrb_flat_Disj[simp]: Q ∈ flat_Disj Q′ ⟹ rrb Q′ ⟹ rrb Q
⟨proof⟩

lemma (**in** simplification) rb_INV_finite[simp]: rb_INV x Q 𝒬 ⟹ finite 𝒬
⟨proof⟩

lemma (**in** *simplification*) *rb_INV_fv*: *rb_INV x Q* $\mathcal{Q} \Longrightarrow Q' \in \mathcal{Q} \Longrightarrow z \in$ *fv* $Q' \Longrightarrow z \in$ *fv Q*
⟨*proof*⟩

lemma (**in** *simplification*) *rb_INV_rrb*: *rb_INV x Q* $\mathcal{Q} \Longrightarrow Q' \in \mathcal{Q} \Longrightarrow$ *rrb* Q'
⟨*proof*⟩

lemma (**in** *simplification*) *rb_INV_cpropagated*: *rb_INV x Q* $\mathcal{Q} \Longrightarrow Q' \in \mathcal{Q} \Longrightarrow$ *simplified* Q'
⟨*proof*⟩

lemma (**in** *simplification*) *rb_INV_equiv*: *rb_INV x Q* $\mathcal{Q} \Longrightarrow$ *Exists x Q* $\triangleq$ *DISJ* (*exists x* ' $\mathcal{Q}$)
⟨*proof*⟩

lemma (**in** *simplification*) *rb_INV_init*[*simp*]: *simplified Q* $\Longrightarrow$ *rrb Q* $\Longrightarrow$ *rb_INV x Q* (*flat_Disj Q*)
⟨*proof*⟩

lemma (**in** *simplification*) *rb_INV_step*[*simp*]:
fixes *Q* :: ('*a* :: {*infinite*, *linorder*}, '*b* :: *linorder*) *fmla*
assumes *rb_INV x Q* $\mathcal{Q}$ $Q' \in$ *fixbound* $\mathcal{Q}$ *x cov x* Q' *G*
shows *rb_INV x Q* (*insert* (*cp* ($Q' \perp x$)) (*insert* (*simp* (*Conj* Q' (*DISJ* (*qps G*)))) ($\mathcal{Q}$ − {Q'} ∪ (λ*y*. *cp* ($Q'[x \to y]$)) ' *eqs x G*)))
⟨*proof*⟩

lemma (**in** *simplification*) *rb_correct*:
fixes *Q* :: ('*a* :: {*linorder*, *infinite*}, '*b* :: *linorder*) *fmla*
shows *rb Q* ≤ *rb_spec Q*
⟨*proof*⟩

4 Refining the Non-Deterministic *simplification.rb* Function

fun *gen_size* **where**
gen_size (*Bool b*) = *1*
| *gen_size* (*Eq x t*) = *1*
| *gen_size* (*Pred p ts*) = *1*
| *gen_size* (*Neg* (*Neg Q*)) = *Suc* (*gen_size Q*)
| *gen_size* (*Neg* (*Conj Q1 Q2*)) = *Suc* (*Suc* (*gen_size* (*Neg Q1*) + *gen_size* (*Neg Q2*)))
| *gen_size* (*Neg* (*Disj Q1 Q2*)) = *Suc* (*Suc* (*gen_size* (*Neg Q1*) + *gen_size* (*Neg Q2*)))
| *gen_size* (*Neg Q*) = *Suc* (*gen_size Q*)
| *gen_size* (*Conj Q1 Q2*) = *Suc* (*gen_size Q1* + *gen_size Q2*)
| *gen_size* (*Disj Q1 Q2*) = *Suc* (*gen_size Q1* + *gen_size Q2*)
| *gen_size* (*Exists x Q*) = *Suc* (*gen_size Q*)

function (*sequential*) *gen_impl* **where**
gen_impl x (*Bool False*) = [{}]
| *gen_impl x* (*Bool True*) = []
| *gen_impl x* (*Eq y* (*Const c*)) = (*if x* = *y then* [{*Eq y* (*Const c*)}] *else* [])
| *gen_impl x* (*Eq y* (*Var z*)) = []
| *gen_impl x* (*Pred p ts*) = (*if x* ∈ *fv_terms_set ts then* [{*Pred p ts*}] *else* [])
| *gen_impl x* (*Neg* (*Neg Q*)) = *gen_impl x Q*
| *gen_impl x* (*Neg* (*Conj Q1 Q2*)) = *gen_impl x* (*Disj* (*Neg Q1*) (*Neg Q2*))
| *gen_impl x* (*Neg* (*Disj Q1 Q2*)) = *gen_impl x* (*Conj* (*Neg Q1*) (*Neg Q2*))
| *gen_impl x* (*Neg _*) = []
| *gen_impl x* (*Disj Q1 Q2*) = [*G1* ∪ *G2*. *G1* ← *gen_impl x Q1*, *G2* ← *gen_impl x Q2*]
| *gen_impl x* (*Conj Q1* ($y \approx z$)) = (*if x* = *y then List.union* (*gen_impl x Q1*) (*map* (*image* (λ*Q*. *cp* ($Q[z \to x]$))) (*gen_impl z Q1*))
else if x = *z then List.union* (*gen_impl x Q1*) (*map* (*image* (λ*Q*. *cp* ($Q[y \to x]$))) (*gen_impl y Q1*))
else gen_impl x Q1)|

gen_impl x (*Conj Q1 Q2*) = *List.union* (*gen_impl x Q1*) (*gen_impl x Q2*)
| *gen_impl x* (*Exists y Q*) = (*if x* = *y then* [] *else map* (*image* (*exists y*)) (*gen_impl x Q*))
⟨*proof*⟩
termination ⟨*proof*⟩

lemma *gen_impl_gen*: $G \in$ *set* (*gen_impl x Q*) $\Longrightarrow$ *gen x Q G*
⟨*proof*⟩

lemma *gen_gen_impl*: *gen x Q G* $\Longrightarrow$ $G \in$ *set* (*gen_impl x Q*)
⟨*proof*⟩

lemma *set_gen_impl*: *set* (*gen_impl x Q*) = {*G*. *gen x Q G*}
⟨*proof*⟩

definition *flat xss* = *fold List.union xss* []

primrec *cov_impl* **where**
cov_impl x (*Bool b*) = [{}]
| *cov_impl x* (*Eq y t*) = (*case t of*
    Const c ⇒ [*if x* = *y then* {*Eq y* (*Const c*)} *else* {}]
  | *Var z* ⇒ [*if x* = *y* ∧ *x* ≠ *z then* {*x* ≈ *z*}
        else if x = *z* ∧ *x* ≠ *y then* {*x* ≈ *y*}
        else {}])
| *cov_impl x* (*Pred p ts*) = [*if x* ∈ *fv_terms_set ts then* {*Pred p ts*} *else* {}]
| *cov_impl x* (*Neg Q*) = *cov_impl x Q*
| *cov_impl x* (*Disj Q1 Q2*) = (*case* (*cp* (*Q1* ⊥ *x*), *cp* (*Q2* ⊥ *x*)) *of*
    (*Bool True*, *Bool True*) ⇒ *List.union* (*cov_impl x Q1*) (*cov_impl x Q2*)
  | (*Bool True*, _) ⇒ *cov_impl x Q1*
  | (_, *Bool True*) ⇒ *cov_impl x Q2*
  | (_, _) ⇒ [*G1* ∪ *G2*. *G1* ← *cov_impl x Q1*, *G2* ← *cov_impl x Q2*])
| *cov_impl x* (*Conj Q1 Q2*) = (*case* (*cp* (*Q1* ⊥ *x*), *cp* (*Q2* ⊥ *x*)) *of*
    (*Bool False*, *Bool False*) ⇒ *List.union* (*cov_impl x Q1*) (*cov_impl x Q2*)
  | (*Bool False*, _) ⇒ *cov_impl x Q1*
  | (_, *Bool False*) ⇒ *cov_impl x Q2*
  | (_, _) ⇒ [*G1* ∪ *G2*. *G1* ← *cov_impl x Q1*, *G2* ← *cov_impl x Q2*])
| *cov_impl x* (*Exists y Q*) = (*if x* = *y then* [{}] *else flat* (*map* (λ*G*.
    (*if x* ≈ *y* ∈ *G then* [*exists y* ‘ (*G* − {*x* ≈ *y*}) ∪ (λ*Q*. *cp* (*Q*[*y* → *x*])) ‘ *G′*. *G′* ← *gen_impl y Q*]
    else [*exists y* ‘ *G*])) (*cov_impl x Q*)))

lemma *union_empty_iff*: *List.union xs ys* = [] ⟷ *xs* = [] ∧ *ys* = []
⟨*proof*⟩

lemma *fold_union_empty_iff*: *fold List.union xss ys* = [] ⟷ (∀ *xs* ∈ *set xss*. *xs* = []) ∧ *ys* = []
⟨*proof*⟩

lemma *flat_empty_iff*: *flat xss* = [] ⟷ (∀ *xs* ∈ *set xss*. *xs* = [])
⟨*proof*⟩

lemma *set_fold_union*: *set* (*fold List.union xss ys*) = (⋃ (*set* ‘ *set xss*)) ∪ *set ys*
⟨*proof*⟩

lemma *set_flat*: *set* (*flat xss*) = ⋃ (*set* ‘ *set xss*)
⟨*proof*⟩

lemma *rrb_cov_impl*: *rrb Q* ⟹ *cov_impl x Q* ≠ []
⟨*proof*⟩

lemma *cov_Eq_self*: $cov\ x\ (y \approx y)\ \{\}$
⟨*proof*⟩

lemma *cov_impl_cov*: $G \in set\ (cov\_impl\ x\ Q) \Longrightarrow cov\ x\ Q\ G$
⟨*proof*⟩

definition *fixbound_impl* $\mathcal{Q}\ x = filter\ (\lambda Q.\ x \in fv\ Q \wedge gen\_impl\ x\ Q = [])\ (sorted\_list\_of\_set\ \mathcal{Q})$

lemma *set_fixbound_impl*: $finite\ \mathcal{Q} \Longrightarrow set\ (fixbound\_impl\ \mathcal{Q}\ x) = fixbound\ \mathcal{Q}\ x$
⟨*proof*⟩

lemma *fixbound_empty_iff*: $finite\ \mathcal{Q} \Longrightarrow fixbound\ \mathcal{Q}\ x \neq \{\} \longleftrightarrow fixbound\_impl\ \mathcal{Q}\ x \neq []$
⟨*proof*⟩

lemma *fixbound_impl_hd_in*: $finite\ \mathcal{Q} \Longrightarrow fixbound\_impl\ \mathcal{Q}\ x = y\ \#\ ys \Longrightarrow y \in \mathcal{Q}$
⟨*proof*⟩

fun (**in** *simplification*) *rb_impl* :: $('a :: \{infinite, linorder\}, 'b :: linorder)\ fmla \Rightarrow ('a, 'b)\ fmla\ nres$ **where**
$rb\_impl\ (Neg\ Q) = do\ \{\ Q' \leftarrow rb\_impl\ Q;\ RETURN\ (simp\ (Neg\ Q'))\}$
$|\ rb\_impl\ (Disj\ Q1\ Q2) = do\ \{\ Q1' \leftarrow rb\_impl\ Q1;\ Q2' \leftarrow rb\_impl\ Q2;\ RETURN\ (simp\ (Disj\ Q1'\ Q2'))\}$
$|\ rb\_impl\ (Conj\ Q1\ Q2) = do\ \{\ Q1' \leftarrow rb\_impl\ Q1;\ Q2' \leftarrow rb\_impl\ Q2;\ RETURN\ (simp\ (Conj\ Q1'\ Q2'))\}$
$|\ rb\_impl\ (Exists\ x\ Q) = do\ \{$
$Q' \leftarrow rb\_impl\ Q;$
$\mathcal{Q} \leftarrow WHILE$
$(\lambda\mathcal{Q}.\ fixbound\_impl\ \mathcal{Q}\ x \neq [])\ (\lambda\mathcal{Q}.\ do\ \{$
$Qfix \leftarrow RETURN\ (hd\ (fixbound\_impl\ \mathcal{Q}\ x));$
$G \leftarrow RETURN\ (hd\ (cov\_impl\ x\ Qfix));$
$RETURN\ (\mathcal{Q} - \{Qfix\} \cup$
$\{simp\ (Conj\ Qfix\ (DISJ\ (qps\ G)))\} \cup$
$(\bigcup y \in eqs\ x\ G.\ \{cp\ (Qfix[x \rightarrow y])\}) \cup$
$\{cp\ (Qfix \perp x)\}\})$
$(flat\_Disj\ Q');$
$RETURN\ (simp\ (DISJ\ (exists\ x\ `\ \mathcal{Q})))\}$
$|\ rb\_impl\ Q = do\ \{\ RETURN\ (simp\ Q)\ \}$

lemma (**in** *simplification*) *rb_impl_refines_rb*: $rb\_impl\ Q \leq rb\ Q$
⟨*proof*⟩

fun (**in** *simplification*) *rb_impl_det* :: $('a :: \{infinite, linorder\}, 'b :: linorder)\ fmla \Rightarrow ('a, 'b)\ fmla\ dres$ **where**
$rb\_impl\_det\ (Neg\ Q) = do\ \{\ Q' \leftarrow rb\_impl\_det\ Q;\ dRETURN\ (simp\ (Neg\ Q'))\}$
$|\ rb\_impl\_det\ (Disj\ Q1\ Q2) = do\ \{\ Q1' \leftarrow rb\_impl\_det\ Q1;\ Q2' \leftarrow rb\_impl\_det\ Q2;\ dRETURN\ (simp\ (Disj\ Q1'\ Q2'))\}$
$|\ rb\_impl\_det\ (Conj\ Q1\ Q2) = do\ \{\ Q1' \leftarrow rb\_impl\_det\ Q1;\ Q2' \leftarrow rb\_impl\_det\ Q2;\ dRETURN\ (simp\ (Conj\ Q1'\ Q2'))\}$
$|\ rb\_impl\_det\ (Exists\ x\ Q) = do\ \{$
$Q' \leftarrow rb\_impl\_det\ Q;$
$\mathcal{Q} \leftarrow dWHILE$
$(\lambda\mathcal{Q}.\ fixbound\_impl\ \mathcal{Q}\ x \neq [])\ (\lambda\mathcal{Q}.\ do\ \{$
$Qfix \leftarrow dRETURN\ (hd\ (fixbound\_impl\ \mathcal{Q}\ x));$
$G \leftarrow dRETURN\ (hd\ (cov\_impl\ x\ Qfix));$
$dRETURN\ (\mathcal{Q} - \{Qfix\} \cup$
$\{simp\ (Conj\ Qfix\ (DISJ\ (qps\ G)))\} \cup$
$(\bigcup y \in eqs\ x\ G.\ \{cp\ (Qfix[x \rightarrow y])\}) \cup$
$\{cp\ (Qfix \perp x)\}\})$
$(flat\_Disj\ Q');$

dRETURN (*simp* (*DISJ* (*exists x* ‘ *Q*)))}
| *rb_impl_det Q* = *do* { *dRETURN* (*simp Q*) }

lemma (**in** *simplification*) *rb_impl_det_refines_rb_impl*: *nres_of* (*rb_impl_det Q*) ≤ *rb_impl Q*
⟨*proof*⟩

lemmas (**in** *simplification*) *RB_correct* =
rb_impl_det_refines_rb_impl[*THEN order_trans*, *OF*
rb_impl_refines_rb[*THEN order_trans*, *OF*
rb_correct]]

5 Restricting Free Variables

definition *fixfree* :: ((′*a*, ′*b*) *fmla* × *nat rel*) *set* ⇒ ((′*a*, ′*b*) *fmla* × *nat rel*) *set* **where**
fixfree Qfin = {(*Qfix*, *Qeq*) ∈ *Qfin*. *nongens Qfix* ≠ {}}

definition *disjointvars Q Qeq* = (⋃ *V* ∈ *classes Qeq*. *if V* ∩ *fv Q* = {} *then V else* {})

fun *Conjs* **where**
Conjs Q [] = *Q*
| *Conjs Q* ((*x*, *y*) # *xys*) = *Conjs* (*Conj Q* (*x* ≈ *y*)) *xys*

function (*sequential*) *Conjs_disjoint* **where**
Conjs_disjoint Q xys = (*case find* (λ(*x*,*y*). {*x*, *y*} ∩ *fv Q* ≠ {}) *xys of*
None ⇒ *Conjs Q xys*
| *Some* (*x*, *y*) ⇒ *Conjs_disjoint* (*Conj Q* (*x* ≈ *y*)) (*remove1* (*x*, *y*) *xys*))
⟨*proof*⟩
termination
⟨*proof*⟩

declare *Conjs_disjoint.simps*[*simp del*]

definition *CONJ* **where**
CONJ = (λ(*Q*, *Qeq*). *Conjs Q* (*sorted_list_of_set Qeq*))

definition *CONJ_disjoint* **where**
CONJ_disjoint = (λ(*Q*, *Qeq*). *Conjs_disjoint Q* (*sorted_list_of_set Qeq*))

definition *inf* **where**
inf Qfin Q = {(*Q*′, *Qeq*) ∈ *Qfin*. *disjointvars Q*′ *Qeq* ≠ {} ∨ *fv Q*′ ∪ *Field Qeq* ≠ *fv Q*}

definition *FV* **where**
FV Q Qfin Qinf ≡ (*fv Qfin* = *fv Q* ∨ *Qfin* = *Bool False*) ∧ *fv Qinf* = {}

definition *EVAL* **where**
EVAL Q Qfin Qinf ≡ (∀ *I*. *finite* (*adom I*) ⟶ (*if eval Qinf I* = {} *then*
eval Qfin I = *eval Q I else infinite* (*eval Q I*)))

definition *EVAL*′ **where**
EVAL′ *Q Qfin Qinf* ≡ (∀ *I*. *finite* (*adom I*) ⟶ (*if eval Qinf I* = {} *then*
eval_on (*fv Q*) *Qfin I* = *eval Q I else infinite* (*eval Q I*)))

definition (**in** *simplification*) *split_spec* :: (′*a* :: {*infinite*, *linorder*}, ′*b* :: *linorder*) *fmla* ⇒ ((′*a*, ′*b*) *fmla* × (′*a*, ′*b*) *fmla*) *nres* **where**
split_spec Q = *SPEC* (λ(*Qfin*, *Qinf*). *sr Qfin* ∧ *sr Qinf* ∧ *FV Q Qfin Qinf* ∧ *EVAL Q Qfin Qinf* ∧
simplified Qfin ∧ *simplified Qinf*)

definition (**in** *simplification*) *assemble* = (λ(*Qfin*, *Qinf*). (*simp* (*DISJ* (*CONJ_disjoint* ‘ *Qfin*)), *simp* (*DISJ* (*close* ‘ *Qinf*))))

fun *leftfresh* **where**
leftfresh Q [] = *True*
| *leftfresh Q* ((*x*, *y*) # *xys*) = ($x \notin$ *fv Q* $\wedge$ *leftfresh* (*Conj Q* ($x \approx y$)) *xys*)

definition (**in** *simplification*) *wf_state Q P* =
(λ(*Qfin*, *Qinf*). *finite Qfin* $\wedge$ *finite Qinf* $\wedge$
($\forall$ (*Qfix*, *Qeq*) $\in$ *Qfin*. *P Qfix* $\wedge$ *simplified Qfix* $\wedge$ ($\exists$ *xs*. *leftfresh Qfix xs* $\wedge$ *distinct xs* $\wedge$ *set xs* = *Qeq*) $\wedge$ *fv Qfix* $\cup$ *Field Qeq* $\subseteq$ *fv Q* $\wedge$ *irrefl Qeq*))

definition (**in** *simplification*) *split_INV1 Q* = (λ*Qpair*. *wf_state Q rrb Qpair* $\wedge$ (*let* (*Qfin*, *Qinf*) = *assemble Qpair in EVAL′ Q Qfin Qinf*))
definition (**in** *simplification*) *split_INV2 Q* = (λ*Qpair*. *wf_state Q sr Qpair* $\wedge$ (*let* (*Qfin*, *Qinf*) = *assemble Qpair in EVAL′ Q Qfin Qinf*))

definition (**in** *simplification*) *split* :: (′*a* :: {*infinite*, *linorder*}, ′*b* :: *linorder*) *fmla* $\Rightarrow$ ((′*a*, ′*b*) *fmla* $\times$ (′*a*, ′*b*) *fmla*) *nres* **where**
split Q = *do* {
Q′ $\leftarrow$ *rb Q*;
Qpair $\leftarrow$ $WHILE_T{}^{split\_INV1\ Q}$
(λ(*Qfin*, _). *fixfree Qfin* $\neq$ {}) (λ(*Qfin*, *Qinf*). *do* {
(*Qfix*, *Qeq*) $\leftarrow$ *RES* (*fixfree Qfin*);
x $\leftarrow$ *RES* (*nongens Qfix*);
G $\leftarrow$ *SPEC* (*cov x Qfix*);
let Qfin = *Qfin* − {(*Qfix*, *Qeq*)} $\cup$
{(*simp* (*Conj Qfix* (*DISJ* (*qps G*))), *Qeq*)} $\cup$
($\bigcup$ *y* $\in$ *eqs x G*. {(*cp* (*Qfix*[*x* $\rightarrow$ *y*]), *Qeq* $\cup$ {(*x*,*y*)})});
let Qinf = *Qinf* $\cup$ {*cp* (*Qfix* $\perp$ *x*)};
RETURN (*Qfin*, *Qinf*)})
({(*Q′*, {})}, {});
Qpair $\leftarrow$ $WHILE_T{}^{split\_INV2\ Q}$
(λ(*Qfin*, _). *inf Qfin Q* $\neq$ {}) (λ(*Qfin*, *Qinf*). *do* {
Qpair $\leftarrow$ *RES* (*inf Qfin Q*);
let Qfin = *Qfin* − {*Qpair*};
let Qinf = *Qinf* $\cup$ {*CONJ Qpair*};
RETURN (*Qfin*, *Qinf*)})
Qpair;
let (*Qfin*, *Qinf*) = *assemble Qpair*;
Qinf $\leftarrow$ *rb Qinf*;
RETURN (*Qfin*, *Qinf*)}

lemma *finite_fixfree*[*simp*]: *finite Q* $\Longrightarrow$ *finite* (*fixfree Q*)
⟨*proof*⟩

lemma (**in** *simplification*) *split_step_in_mult*:
assumes (*Qfin*, *Qeq*) $\in$ *Qfin finite Qfin x* $\in$ *nongens Qfin cov x Qfin G fv Qfin* $\subseteq$ *F*
shows ((*nongens* $\circ$ *fst*) ‘# *mset_set* (*insert* (*simp* (*Conj Qfin* (*DISJ* (*qps G*))), *Qeq*) (*Qfin* − {(*Qfin*, *Qeq*)} $\cup$ (λ*y*. (*cp* (*Qfin*[*x* $\rightarrow$ *y*]), *insert* (*x*, *y*) *Qeq*)) ‘ *eqs x G*)),
(*nongens* $\circ$ *fst*) ‘# *mset_set Qfin*) $\in$ *mult* {(*X*, *Y*). *X* $\subset$ *Y* $\wedge$ *Y* $\subseteq$ *F*}
(**is** (*?f* (*insert ?Q* (*?A* $\cup$ *?B*)), *?C*) $\in$ *mult ?R*)
⟨*proof*⟩

lemma *EVAL_cong*:
Qinf $\triangleq$ *Qinf′* $\Longrightarrow$ *fv Qinf* = *fv Qinf′* $\Longrightarrow$ *EVAL Q Qfin Qinf* = *EVAL Q Qfin Qinf′*
⟨*proof*⟩

lemma *EVAL′_cong*:
$Qinf \triangleq Qinf' \Longrightarrow fv\ Qinf = fv\ Qinf' \Longrightarrow EVAL'\ Q\ Qfin\ Qinf = EVAL'\ Q\ Qfin\ Qinf'$
⟨*proof*⟩

lemma *fv_Conjs*[*simp*]: $fv\ (Conjs\ Q\ xys) = fv\ Q \cup Field\ (set\ xys)$
⟨*proof*⟩

lemma *fv_Conjs_disjoint*[*simp*]: $distinct\ xys \Longrightarrow fv\ (Conjs\_disjoint\ Q\ xys) = fv\ Q \cup Field\ (set\ xys)$
⟨*proof*⟩

lemma *fv_CONJ*[*simp*]: $finite\ Qeq \Longrightarrow fv\ (CONJ\ (Q,\ Qeq)) = fv\ Q \cup Field\ Qeq$
⟨*proof*⟩

lemma *fv_CONJ_disjoint*[*simp*]: $finite\ Qeq \Longrightarrow fv\ (CONJ\_disjoint\ (Q,\ Qeq)) = fv\ Q \cup Field\ Qeq$
⟨*proof*⟩

lemma *rrb_Conjs*: $rrb\ Q \Longrightarrow rrb\ (Conjs\ Q\ xys)$
⟨*proof*⟩

lemma *CONJ_empty*[*simp*]: $CONJ\ (Q,\ \{\}) = Q$
⟨*proof*⟩

lemma *CONJ_disjoint_empty*[*simp*]: $CONJ\_disjoint\ (Q,\ \{\}) = Q$
⟨*proof*⟩

lemma *Conjs_eq_False_iff*[*simp*]: $irrefl\ (set\ xys) \Longrightarrow Conjs\ Q\ xys = Bool\ False \longleftrightarrow Q = Bool\ False \wedge xys = []$
⟨*proof*⟩

lemma *CONJ_eq_False_iff*[*simp*]: $finite\ Qeq \Longrightarrow irrefl\ Qeq \Longrightarrow CONJ\ (Q,\ Qeq) = Bool\ False \longleftrightarrow Q = Bool\ False \wedge Qeq = \{\}$
⟨*proof*⟩

lemma *Conjs_disjoint_eq_False_iff*[*simp*]: $irrefl\ (set\ xys) \Longrightarrow Conjs\_disjoint\ Q\ xys = Bool\ False \longleftrightarrow Q = Bool\ False \wedge xys = []$
⟨*proof*⟩

lemma *CONJ_disjoint_eq_False_iff*[*simp*]: $finite\ Qeq \Longrightarrow irrefl\ Qeq \Longrightarrow CONJ\_disjoint\ (Q,\ Qeq) = Bool\ False \longleftrightarrow Q = Bool\ False \wedge Qeq = \{\}$
⟨*proof*⟩

lemma *sr_Conjs_disjoint*:
$distinct\ xys \Longrightarrow (\forall\ V \in classes\ (set\ xys).\ V \cap fv\ Q \neq \{\}) \Longrightarrow sr\ Q \Longrightarrow sr\ (Conjs\_disjoint\ Q\ xys)$
⟨*proof*⟩

lemma *sr_CONJ_disjoint*:
$inf\ \mathcal{Q}fin\ Q = \{\} \Longrightarrow (Qfin,\ Qeq) \in \mathcal{Q}fin \Longrightarrow finite\ Qeq \Longrightarrow sr\ Qfin \Longrightarrow sr\ (CONJ\_disjoint\ (Qfin,\ Qeq))$
⟨*proof*⟩

lemma *equiv_Conjs_cong*: $Q \triangleq Q' \Longrightarrow Conjs\ Q\ xys \triangleq Conjs\ Q'\ xys$
⟨*proof*⟩

lemma *Conjs_pull_out*: $Conjs\ Q\ (xys\ @\ (x,\ y)\ \#\ xys') \triangleq Conjs\ (Conj\ Q\ (x \approx y))\ (xys\ @\ xys')$
⟨*proof*⟩

lemma *Conjs_reorder*: $distinct\ xys \Longrightarrow distinct\ xys' \Longrightarrow set\ xys = set\ xys' \Longrightarrow Conjs\ Q\ xys \triangleq Conjs\ Q\ xys'$

⟨*proof*⟩

lemma *ex_Conjs_disjoint_eq_Conjs*:
distinct xys $\Longrightarrow$ $\exists$ *xys'*. *distinct xys'* $\wedge$ *set xys* = *set xys'* $\wedge$ *Conjs_disjoint Q xys* = *Conjs Q xys'*
⟨*proof*⟩

lemma *Conjs_disjoint_equiv_Conjs*:
assumes *distinct xys*
shows *Conjs_disjoint Q xys* $\triangleq$ *Conjs Q xys*
⟨*proof*⟩

lemma *infinite_eval_Conjs*: *infinite* (*eval Q I*) $\Longrightarrow$ *leftfresh Q xys* $\Longrightarrow$ *infinite* (*eval* (*Conjs Q xys*) *I*)
⟨*proof*⟩

lemma *leftfresh_fv_subset*: *leftfresh Q xys* $\Longrightarrow$ *fv Q'* $\subseteq$ *fv Q* $\Longrightarrow$ *leftfresh Q' xys*
⟨*proof*⟩

lemma *fun_upds_map*: ($\forall$ *x*. *x* $\notin$ *set ys* $\longrightarrow$ σ *x* = τ *x*) $\Longrightarrow$ σ[*ys* :=* *map* τ *ys*] = τ
⟨*proof*⟩

lemma *map_fun_upds*: *length xs* = *length ys* $\Longrightarrow$ *distinct xs* $\Longrightarrow$ *map* (σ[*xs* :=* *ys*]) *xs* = *ys*
⟨*proof*⟩

lemma *zip_map*: *zip xs* (*map f xs*) = *map* (λ*x*. (*x*, *f x*)) *xs*
⟨*proof*⟩

lemma *filter_sorted_list_of_set*:
finite B $\Longrightarrow$ *A* $\subseteq$ *B* $\Longrightarrow$ *filter* (λ*x*. *x* $\in$ *A*) (*sorted_list_of_set B*) = *sorted_list_of_set A*
⟨*proof*⟩

lemma *infinite_eval_eval_on*[*rotated 2*]:
assumes *fv Q* $\subseteq$ *X finite X*
shows *infinite* (*eval Q I*) $\Longrightarrow$ *infinite* (*eval_on X Q I*)
⟨*proof*⟩

lemma *infinite_eval_CONJ_disjoint*:
assumes *infinite* (*eval Q I*) *finite* (*adom I*) *fv Q* $\subseteq$ *X Field Qeq* $\subseteq$ *X finite X* $\exists$ *xys*. *distinct xys* $\wedge$ *leftfresh Q xys* $\wedge$ *set xys* = *Qeq*
shows *infinite* (*eval_on X* (*CONJ_disjoint* (*Q*, *Qeq*)) *I*)
⟨*proof*⟩

lemma *sat_Conjs*: *sat* (*Conjs Q xys*) *I* σ $\longleftrightarrow$ *sat Q I* σ $\wedge$ ($\forall$ (*x*, *y*) $\in$ *set xys*. *sat* (*x* $\approx$ *y*) *I* σ)
⟨*proof*⟩

lemma *sat_Conjs_disjoint*: *sat* (*Conjs_disjoint Q xys*) *I* σ $\longleftrightarrow$ *sat Q I* σ $\wedge$ ($\forall$ (*x*, *y*) $\in$ *set xys*. *sat* (*x* $\approx$ *y*) *I* σ)
⟨*proof*⟩

lemma *sat_CONJ*: *finite Qeq* $\Longrightarrow$ *sat* (*CONJ* (*Q*, *Qeq*)) *I* σ $\longleftrightarrow$ *sat Q I* σ $\wedge$ ($\forall$ (*x*, *y*) $\in$ *Qeq*. *sat* (*x* $\approx$ *y*) *I* σ)
⟨*proof*⟩

lemma *sat_CONJ_disjoint*: *finite Qeq* $\Longrightarrow$ *sat* (*CONJ_disjoint* (*Q*, *Qeq*)) *I* σ $\longleftrightarrow$ *sat Q I* σ $\wedge$ ($\forall$ (*x*, *y*) $\in$ *Qeq*. *sat* (*x* $\approx$ *y*) *I* σ)
⟨*proof*⟩

lemma *Conjs_inject*: *Conjs Q xys* = *Conjs Q' xys* $\longleftrightarrow$ *Q* = *Q'*
⟨*proof*⟩

lemma *nonempty_disjointvars_infinite*:
assumes *disjointvars* (*Qfin* :: ('*a* :: *infinite*, '*b*) *fmla*) *Qeq* ≠ {}
finite Qeq fv Qfin ∪ *Field Qeq* ⊆ *X finite X sat Qfin I* σ ∀(*x*, *y*)∈*Qeq*. σ *x* = σ *y*
shows *infinite* (*eval_on X* (*CONJ_disjoint* (*Qfin*, *Qeq*)) *I*)
⟨*proof*⟩

lemma *EVAL'_EVAL*: *EVAL' Q Qfin Qinf* ⟹ *FV Q Qfin Qinf* ⟹ *EVAL Q Qfin Qinf*
⟨*proof*⟩

lemma *cpropagated_Conjs_disjoint*:
distinct xys ⟹ *irrefl* (*set xys*) ⟹ ∀ *V*∈*classes* (*set xys*). *V* ∩ *fv Q* ≠ {} ⟹ *cpropagated Q* ⟹ *cpropagated* (*Conjs_disjoint Q xys*)
⟨*proof*⟩

lemma (**in** *simplification*) *simplified_Conjs_disjoint*:
distinct xys ⟹ *irrefl* (*set xys*) ⟹ ∀ *V*∈*classes* (*set xys*). *V* ∩ *fv Q* ≠ {} ⟹ *simplified Q* ⟹ *simplified* (*Conjs_disjoint Q xys*)
⟨*proof*⟩

lemma *disjointvars_empty_iff*: *disjointvars Q Qeq* = {} ⟷ (∀ *V*∈*classes Qeq*. *V* ∩ *fv Q* ≠ {})
⟨*proof*⟩

lemma *cpropagated_CONJ_disjoint*:
finite Qeq ⟹ *irrefl Qeq* ⟹ *disjointvars Q Qeq* = {} ⟹ *cpropagated Q* ⟹ *cpropagated* (*CONJ_disjoint* (*Q*, *Qeq*))
⟨*proof*⟩

lemma (**in** *simplification*) *simplified_CONJ_disjoint*:
finite Qeq ⟹ *irrefl Qeq* ⟹ *disjointvars Q Qeq* = {} ⟹ *simplified Q* ⟹ *simplified* (*CONJ_disjoint* (*Q*, *Qeq*))
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV1_init*:
rrb Q' ⟹ *simplified Q'* ⟹ *Q* ≜ *Q'* ⟹ *fv Q'* ⊆ *fv Q* ⟹ *split_INV1 Q* ({(*Q'*, {})}, {})
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV1_I*:
wf_state Q rrb (*𝒬fin*, *𝒬inf*) ⟹ *EVAL' Q* (*simp* (*DISJ* (*CONJ_disjoint* ‘ *𝒬fin*))) (*simp* (*DISJ* (*close* ‘ *𝒬inf*))) ⟹
split_INV1 Q (*𝒬fin*, *𝒬inf*)
⟨*proof*⟩

lemma *EVAL'_I*:
(⋀*I*. *finite* (*adom I*) ⟹ *eval Qinf I* = {} ⟹ *eval_on* (*fv Q*) *Qfin I* = *eval Q I*) ⟹
(⋀*I*. *finite* (*adom I*) ⟹ *eval Qinf I* ≠ {} ⟹ *infinite* (*eval Q I*)) ⟹ *EVAL' Q Qfin Qinf*
⟨*proof*⟩

lemma (**in** *simplification*) *wf_state_Un*:
wf_state Q P (*𝒬fin*, *𝒬inf*) ⟹ *wf_state Q P* (*insert Qpair 𝒬new*, {*Q'*}) ⟹
wf_state Q P (*insert Qpair* (*𝒬fin* ∪ *𝒬new*), *insert Q' 𝒬inf*)
⟨*proof*⟩

lemma (**in** *simplification*) *wf_state_Diff*:
wf_state Q P (*𝒬fin*, *𝒬inf*) ⟹ *wf_state Q P* (*𝒬fin* − *𝒬new*, *𝒬inf*)
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV1_step*:

assumes *split_INV1 Q (Qfin, Qinf) (Qfin, Qeq) ∈ fixfree Qfin x ∈ nongens Qfin cov x Qfin G*
shows *split_INV1 Q*
(*insert* (*simp* (*Conj Qfin* (*DISJ* (*qps G*))), *Qeq*)
(*Qfin* − {(*Qfin*, *Qeq*)} ∪ (λ*y*. (*cp* (*Qfin*[*x* → *y*]), *insert* (*x*, *y*) *Qeq*)) ' *eqs x G*),
insert (*cp* (*Qfin* ⊥ *x*)) *Qinf*)
(**is** *split_INV1 Q* (*?Qfin*, *?Qinf*))
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV1_decreases*:
assumes *split_INV1 Q (Qfin, Qinf) (Qfin, Qeq) ∈ fixfree Qfin x ∈ nongens Qfin cov x Qfin G*
shows ((*nongens* ∘ *fst*) '# *mset_set* (*insert* (*simp* (*Conj Qfin* (*DISJ* (*qps G*))), *Qeq*) (*Qfin* − {(*Qfin*, *Qeq*)} ∪ (λ*y*. (*cp* (*Qfin*[*x* → *y*]), *insert* (*x*, *y*) *Qeq*)) ' *eqs x G*)),
(*nongens* ∘ *fst*) '# *mset_set Qfin*) ∈ *mult* {(*X*, *Y*). *X* ⊂ *Y* ∧ *Y* ⊆ *fv Q*}
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV2_init*:
split_INV1 Q (*Qfin*, *Qinf*) ⟹ *fixfree Qfin* = {} ⟹ *split_INV2 Q* (*Qfin*, *Qinf*)
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV2_I*:
wf_state Q sr (*Qfin*, *Qinf*) ⟹ *EVAL′ Q* (*simp* (*DISJ* (*CONJ_disjoint* ' *Qfin*))) (*simp* (*DISJ* (*close* ' *Qinf*))) ⟹
split_INV2 Q (*Qfin*, *Qinf*)
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV2_step*:
assumes *split_INV2 Q* (*Qfin*, *Qinf*) (*Qfin*, *Qeq*) ∈ *inf Qfin Q*
shows *split_INV2 Q* (*Qfin* − {(*Qfin*, *Qeq*)}, *insert* (*CONJ* (*Qfin*, *Qeq*)) *Qinf*)
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV2_decreases*:
split_INV2 Q (*Qfin*, *Qinf*) ⟹ (*Qfin*, *Qeq*) ∈ *Restrict_Frees.inf Qfin Q* ⟹ *card* (*Qfin* − {(*Qfin*, *Qeq*)}) < *card Qfin*
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV2_stop_fin_sr*:
inf Qfin Q = {} ⟹ *split_INV2 Q* (*Qfin*, *Qinf*) ⟹ *assemble* (*Qfin*, *Qinf*) = (*Qfin*, *Qinf*) ⟹ *sr Qfin*
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV2_stop_inf_sr*:
split_INV2 Q (*Qfin*, *Qinf*) ⟹ *assemble* (*Qfin*, *Qinf*) = (*Qfin*, *Qinf*) ⟹ *fv Q′* ⊆ *fv Qinf* ⟹ *rrb Q′* ⟹ *sr Q′*
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV2_stop_FV*:
assumes *fv Q′* ⊆ *fv Qinf inf Qfin Q* = {} *split_INV2 Q* (*Qfin*, *Qinf*) *assemble* (*Qfin*, *Qinf*) = (*Qfin*, *Qinf*)
shows *FV Q Qfin Q′*
⟨*proof*⟩

lemma (**in** *simplification*) *split_INV2_stop_EVAL*:
assumes *fv Q′* ⊆ *fv Qinf inf Qfin Q* = {} *split_INV2 Q* (*Qfin*, *Qinf*) *assemble* (*Qfin*, *Qinf*) = (*Qfin*, *Qinf*) *Qinf* ≜ *Q′*
shows *EVAL Q Qfin Q′*
⟨*proof*⟩

lemma (**in** *simplification*) *simplified_assemble*:
assemble (*Qfin*, *Qinf*) = (*Qfin*, *Qinf*) ⟹ *simplified Qfin*

⟨*proof*⟩

lemma (**in** *simplification*) *split_correct*:
  notes *cp.simps*[*simp del*]
  shows *split Q* ≤ *split_spec Q*
  ⟨*proof*⟩

6 Refining the Non-Deterministic *simplification.split* Function

definition *fixfree_impl* $\mathcal{Q}$ = *map* (*apsnd set*) (*filter* (λ(*Q*, _ :: (*nat* × *nat*) *list*). ∃ *x* ∈ *fv Q*. *gen_impl x Q* = [])
  (*sorted_list_of_set* ((*apsnd sorted_list_of_set*) ‘ $\mathcal{Q}$)))

definition *nongens_impl Q* = *filter* (λ*x*. *gen_impl x Q* = []) (*sorted_list_of_set* (*fv Q*))

lemma *set_nongens_impl*: *set* (*nongens_impl Q*) = *nongens Q*
  ⟨*proof*⟩

lemma *set_fixfree_impl*: *finite* $\mathcal{Q}$ ⟹ ∀ (_, *Qeq*) ∈ $\mathcal{Q}$. *finite Qeq* ⟹ *set* (*fixfree_impl* $\mathcal{Q}$) = *fixfree* $\mathcal{Q}$
  ⟨*proof*⟩

lemma *fixfree_empty_iff*: *finite* $\mathcal{Q}$ ⟹ ∀ (_, *Qeq*) ∈ $\mathcal{Q}$. *finite Qeq* ⟹ *fixfree* $\mathcal{Q}$ ≠ {} ⟷ *fixfree_impl* $\mathcal{Q}$ ≠ []
  ⟨*proof*⟩

definition *inf_impl* $\mathcal{Q}$*fin Q* =
  map (*apsnd set*) (*filter* (λ(*Qfix*, *xys*). *disjointvars Qfix* (*set xys*) ≠ {} ∨ *fv Qfix* ∪ *Field* (*set xys*) ≠ *fv Q*)
    (*sorted_list_of_set* ((*apsnd sorted_list_of_set*) ‘ $\mathcal{Q}$*fin*)))

lemma *set_inf_impl*: *finite* $\mathcal{Q}$*fin* ⟹ ∀ (_, *Qeq*) ∈ $\mathcal{Q}$*fin*. *finite Qeq* ⟹ *set* (*inf_impl* $\mathcal{Q}$*fin Q*) = *inf* $\mathcal{Q}$*fin Q*
  ⟨*proof*⟩

lemma *inf_empty_iff*: *finite* $\mathcal{Q}$*fin* ⟹ ∀ (_, *Qeq*) ∈ $\mathcal{Q}$*fin*. *finite Qeq* ⟹ *inf* $\mathcal{Q}$*fin Q* ≠ {} ⟷ *inf_impl* $\mathcal{Q}$*fin Q* ≠ []
  ⟨*proof*⟩

definition (**in** *simplification*) *split_impl* :: (′*a* :: {*infinite*, *linorder*}, ′*b* :: *linorder*) *fmla* ⇒ ((′*a*, ′*b*) *fmla* × (′*a*, ′*b*) *fmla*) *nres* **where**
  split_impl Q = *do* {
    Q′ ← *rb_impl Q*;
    $\mathcal{Q}$*pair* ← *WHILE*
      (λ($\mathcal{Q}$*fin*, _). *fixfree_impl* $\mathcal{Q}$*fin* ≠ []) (λ($\mathcal{Q}$*fin*, $\mathcal{Q}$*inf*). *do* {
        (*Qfix*, *Qeq*) ← *RETURN* (*hd* (*fixfree_impl* $\mathcal{Q}$*fin*));
        x ← *RETURN* (*hd* (*nongens_impl Qfix*));
        G ← *RETURN* (*hd* (*cov_impl x Qfix*));
        let $\mathcal{Q}$*fin* = $\mathcal{Q}$*fin* − {(*Qfix*, *Qeq*)} ∪
          {(*simp* (*Conj Qfix* (*DISJ* (*qps G*))), *Qeq*)} ∪
          (⋃ *y* ∈ *eqs x G*. {(*cp* (*Qfix*[*x* → *y*]), *Qeq* ∪ {(*x*,*y*)})});
        let $\mathcal{Q}$*inf* = $\mathcal{Q}$*inf* ∪ {*cp* (*Qfix* ⊥ *x*)};
        RETURN ($\mathcal{Q}$*fin*, $\mathcal{Q}$*inf*)})
      ({(*Q*′, {})}, {});
    $\mathcal{Q}$*pair* ← *WHILE*
      (λ($\mathcal{Q}$*fin*, _). *inf_impl* $\mathcal{Q}$*fin Q* ≠ []) (λ($\mathcal{Q}$*fin*, $\mathcal{Q}$*inf*). *do* {
        Qpair ← *RETURN* (*hd* (*inf_impl* $\mathcal{Q}$*fin Q*));

let $\mathcal{Q}fin = \mathcal{Q}fin - \{Qpair\}$;
let $\mathcal{Q}inf = \mathcal{Q}inf \cup \{CONJ\ Qpair\}$;
RETURN ($\mathcal{Q}fin$, $\mathcal{Q}inf$)})
$\mathcal{Q}pair$;
let (*Qfin*, *Qinf*) = *assemble* $\mathcal{Q}pair$;
Qinf ← *rb_impl Qinf*;
RETURN (*Qfin*, *Qinf*)}

lemma (**in** *simplification*) *split_INV2_imp_split_INV1*: *split_INV2 Q* $\mathcal{Q}pair$ ⟹ *split_INV1 Q* $\mathcal{Q}pair$
⟨*proof*⟩

lemma *hd_fixfree_impl_props*:
assumes *finite* $\mathcal{Q}$ ∀ (_, *Qeq*) ∈ $\mathcal{Q}$. *finite Qeq fixfree_impl* $\mathcal{Q}$ ≠ []
shows *hd* (*fixfree_impl* $\mathcal{Q}$) ∈ $\mathcal{Q}$ *nongens* (*fst* (*hd* (*fixfree_impl* $\mathcal{Q}$))) ≠ {}
⟨*proof*⟩

lemma (**in** *simplification*) *split_impl_refines_split*: *split_impl Q* ≤ *split Q*
⟨*proof*⟩

definition (**in** *simplification*) *split_impl_det* :: ('*a* :: {*infinite*, *linorder*}, '*b* :: *linorder*) *fmla* ⇒ (('*a*, '*b*) *fmla* × ('*a*, '*b*) *fmla*) *dres* **where**
split_impl_det Q = *do* {
Q' ← *rb_impl_det Q*;
$\mathcal{Q}pair$ ← *dWHILE*
(λ($\mathcal{Q}fin$, _). *fixfree_impl* $\mathcal{Q}fin$ ≠ []) (λ($\mathcal{Q}fin$, $\mathcal{Q}inf$). *do* {
(*Qfix*, *Qeq*) ← *dRETURN* (*hd* (*fixfree_impl* $\mathcal{Q}fin$));
x ← *dRETURN* (*hd* (*nongens_impl Qfix*));
G ← *dRETURN* (*hd* (*cov_impl x Qfix*));
let $\mathcal{Q}fin = \mathcal{Q}fin - \{(Qfix, Qeq)\} \cup$
{(*simp* (*Conj Qfix* (*DISJ* (*qps G*))), *Qeq*)} ∪
(⋃ *y* ∈ *eqs x G*. {(*cp* (*Qfix*[*x* → *y*]), *Qeq* ∪ {(*x*,*y*)})});
let $\mathcal{Q}inf = \mathcal{Q}inf \cup \{cp\ (Qfix \perp x)\}$;
dRETURN ($\mathcal{Q}fin$, $\mathcal{Q}inf$)})
({(*Q'*, {})}, {});
$\mathcal{Q}pair$ ← *dWHILE*
(λ($\mathcal{Q}fin$, _). *inf_impl* $\mathcal{Q}fin$ *Q* ≠ []) (λ($\mathcal{Q}fin$, $\mathcal{Q}inf$). *do* {
Qpair ← *dRETURN* (*hd* (*inf_impl* $\mathcal{Q}fin$ *Q*));
let $\mathcal{Q}fin = \mathcal{Q}fin - \{Qpair\}$;
let $\mathcal{Q}inf = \mathcal{Q}inf \cup \{CONJ\ Qpair\}$;
dRETURN ($\mathcal{Q}fin$, $\mathcal{Q}inf$)})
$\mathcal{Q}pair$;
let (*Qfin*, *Qinf*) = *assemble* $\mathcal{Q}pair$;
Qinf ← *rb_impl_det Qinf*;
dRETURN (*Qfin*, *Qinf*)}

lemma (**in** *simplification*) *split_impl_det_refines_split_impl*: *nres_of* (*split_impl_det Q*) ≤ *split_impl Q*
⟨*proof*⟩

lemmas (**in** *simplification*) *SPLIT_correct* =
split_impl_det_refines_split_impl[*THEN order_trans*, *OF*
split_impl_refines_split[*THEN order_trans*, *OF*
split_correct]]

7 Examples

global_interpretation *extra_cp*: *simplification cp cpropagated*
  defines *RB* = *simplification.rb_impl_det cp*
    and *assemble* = *simplification.assemble cp*
    and *SPLIT* = *simplification.split_impl_det cp*
  ⟨*proof*⟩

7.1 Restricting Bounds in the "Suspicious Users" Query

context
  fixes *b s p u* :: *nat* **and** *B P S*
  defines *b* ≡ *0*
    and *s* ≡ *Suc 0*
    and *p* ≡ *Suc* (*Suc 0*)
    and *u* ≡ *Suc* (*Suc* (*Suc 0*))
    and *B* ≡ λ*b*. *Pred ''B''* [*Var b*] :: (*string*, *string*) *fmla*
    and *P* ≡ λ*b p*. *Pred ''P''* [*Var b*, *Var p*] :: (*string*, *string*) *fmla*
    and *S* ≡ λ*p u s*. *Pred ''S''* [*Var p*, *Var u*, *Var s*] :: (*string*, *string*) *fmla*
  notes *cp.simps*[*simp del*]
begin

definition *Q_susp_user* **where**
  Q_susp_user = *Conj* (*B b*) (*Exists s* (*Forall p* (*Impl* (*P b p*) (*S p u s*))))
definition *Q_susp_user_rb* :: (*string*, *string*) *fmla* **where**
  Q_susp_user_rb = *Conj* (*B b*) (*Disj* (*Exists s* (*Conj* (*Forall p* (*Impl* (*P b p*) (*S p u s*))) (*Exists p* (*S p u s*)))) (*Forall p* (*Neg* (*P b p*))))

lemma *ex_rb_Q_susp_user*: *the_res* (*RB Q_susp_user*) = *Q_susp_user_rb*
  ⟨*proof*⟩

end

7.2 Splitting a Disjunction of Predicates

context
  fixes *x y* :: *nat* **and** *B P*
  defines *x* ≡ *0*
    and *y* ≡ *1*
    and *B* ≡ λ*b*. *Pred ''B''* [*Var b*] :: (*string*, *string*) *fmla*
    and *P* ≡ λ*b p*. *Pred ''P''* [*Var b*, *Var p*] :: (*string*, *string*) *fmla*
  notes *cp.simps*[*simp del*]
begin

definition *Q_disj* **where**
  Q_disj = *Disj* (*B x*) (*P x y*)
definition *Q_disj_split_fin* :: (*string*, *string*) *fmla* **where**
  Q_disj_split_fin = *Conj* (*Disj* (*B x*) (*P x y*)) (*P x y*)
definition *Q_disj_split_inf* :: (*string*, *string*) *fmla* **where**
  Q_disj_split_inf = *Exists x* (*B x*)

lemma *ex_split_Q_disj*: *the_res* (*SPLIT Q_disj*) = (*Q_disj_split_fin*, *Q_disj_split_inf*)
  ⟨*proof*⟩

end

7.3 Splitting a Conjunction with an Equality

context
  fixes *x u v* :: *nat* **and** *B*
  defines *x* ≡ *0*

and $u \equiv 1$
and $v \equiv 2$
and $B \equiv \lambda b.\ Pred\ ''B''\ [Var\ b] :: (string, string)\ fmla$
notes *cp.simps*[*simp del*]
begin

definition *Q_eq* **where**
$Q\_eq = Conj\ (B\ x)\ (u \approx v)$
definition *Q_eq_split_fin* :: (*string*, *string*) *fmla* **where**
Q_eq_split_fin = *Bool False*
definition *Q_eq_split_inf* :: (*string*, *string*) *fmla* **where**
Q_eq_split_inf = *Exists x* (*B x*)

lemma *ex_split_Q_eq*: *the_res* (*SPLIT Q_eq*) = (*Q_eq_split_fin*, *Q_eq_split_inf*)
⟨*proof*⟩

end

7.4 Splitting the "Suspicious Users" Query

context
fixes *b s p u* :: *nat* **and** *B P S*
defines $b \equiv 0$
and $s \equiv Suc\ 0$
and $p \equiv Suc\ (Suc\ 0)$
and $u \equiv Suc\ (Suc\ (Suc\ 0))$
and $B \equiv \lambda b.\ Pred\ ''B''\ [Var\ b] :: (string, string)\ fmla$
and $P \equiv \lambda b\ p.\ Pred\ ''P''\ [Var\ b,\ Var\ p] :: (string, string)\ fmla$
and $S \equiv \lambda p\ u\ s.\ Pred\ ''S''\ [Var\ p,\ Var\ u,\ Var\ s] :: (string, string)\ fmla$
notes *cp.simps*[*simp del*]
begin

definition *Q_susp_user_split_fin* = *Conj Q_susp_user_rb* (*Exists s* (*Exists p* (*S p u s*)))
definition *Q_susp_user_split_inf* = *Exists b* (*Conj* (*B b*) (*Forall p* (*Neg* (*P b p*))))

lemma *ex_split_Q_susp_user*: *the_res* (*SPLIT Q_susp_user*) = (*Q_susp_user_split_fin*, *Q_susp_user_split_inf*)
⟨*proof*⟩

end

8 Collected Results from the ICDT'22 Paper

global_interpretation *icdt22*: *simplification* $\lambda x.\ x\ \lambda x.\ True$
⟨*proof*⟩

lemma *cov_eval_fin*:
assumes $cov\ x\ (Q :: ('a :: \{infinite, linorder\}, 'b :: linorder)\ fmla)\ G\ x \in fv\ Q$
$finite\ (adom\ I)\ \bigwedge\sigma.\ \neg\ sat\ (Q \perp x)\ I\ \sigma$
shows $eval\ Q\ I = eval\ (Disj\ (Conj\ Q\ (DISJ\ (qps\ G)))\ (DISJ\ ((\lambda y.\ Conj\ (cp\ (Q[x \rightarrow y]))\ (x \approx y))\ `\ eqs\ x\ G)))\ I$
⟨*proof*⟩

Remapping the formalization statements to the lemma's from the paper:

lemmas *icdt22_lemma_1* = *gen_fv gen_sat gen_cp_erase*
lemmas *icdt22_definition_2* = *sub.simps nongens_def rrb_def sr_def*
lemmas *icdt22_lemma_3* = *ex_cov cov_sat_erase*
lemmas *icdt22_lemma_4* = *cov_fv cov_equiv*[*OF _ refl*]

lemmas *icdt22_lemma_5 = icdt22.cov_Exists_equiv*
lemmas *icdt22_example_6 = ex_rb_Q_susp_user[unfolded Q_susp_user_def Q_susp_user_rb_def]*
lemmas *icdt22_lemma_7 = cov_eval_fin cov_eval_inf*
lemmas *icdt22_lemma_8 = inres_SPEC[OF _ icdt22.rb_correct[unfolded icdt22.rb_spec_def, simplified], of Q Q′* **for** *Q Q′]*
lemmas *icdt22_lemma_9 = inres_SPEC[OF _ icdt22.split_correct[unfolded icdt22.split_spec_def FV_def EVAL_def, simplified], of Q (Qfin, Qinf)* **for** *Q Qfin Qinf, simplified]*
lemmas *icdt22_example_10 = ex_split_Q_disj[unfolded Q_disj_def Q_disj_split_fin_def Q_disj_split_inf_def]*
lemmas *icdt22_example_11 = ex_split_Q_eq[unfolded Q_eq_def Q_eq_split_fin_def Q_eq_split_inf_def]*
lemmas *icdt22_example_12 = ex_split_Q_susp_user[unfolded Q_susp_user_def Q_susp_user_split_fin_def Q_susp_user_split_inf_def]*

Additionally, here are the correctness statements for the algorithm variants with intermediate constant propagation (which are used in the examples):

lemmas *icdt22_lemma_8′ = inres_SPEC[OF _ extra_cp.RB_correct[unfolded extra_cp.rb_spec_def], simplified, of Q Q′* **for** *Q Q′]*
lemmas *icdt22_lemma_9′ = inres_SPEC[OF _ extra_cp.SPLIT_correct[unfolded extra_cp.split_spec_def FV_def EVAL_def, simplified], of Q (Qfin, Qinf)* **for** *Q Qfin Qinf, simplified]*

Now, we summarize the formally verified results from our ICDT'22 paper [2]:

icdt22_lemma_1: ⟦*gen x Q G*; *Qqp* ∈ *G*⟧ ⟹ *x* ∈ *fv Qqp* ∧ *fv Qqp* ⊆ *fv Q*

⟦*gen x Q G*; *sat Q I σ*⟧ ⟹ ∃ *Qqp*∈*G*. *sat Qqp I σ*

⟦*gen x Q G*; *Qqp* ∈ *G*⟧ ⟹ *cp* (*Qqp* ⊥ *x*) = *Bool False*

icdt22_definition_2: *sub* (*Bool t*) = {*Bool t*}

sub (*Pred p ts*) = {*Pred p ts*}

sub (*fmla.Eq x t*) = {*fmla.Eq x t*}

sub (*Neg Q*) = *insert* (*Neg Q*) (*sub Q*)

sub (*Conj Q1 Q2*) = *insert* (*Conj Q1 Q2*) (*sub Q1* ∪ *sub Q2*)

sub (*Disj Q1 Q2*) = *insert* (*Disj Q1 Q2*) (*sub Q1* ∪ *sub Q2*)

sub (*Exists z Q*) = *insert* (*Exists z Q*) (*sub Q*)

nongens Q = {*x* ∈ *fv Q*. ¬ *Gen x Q*}

rrb Q = (∀ *y Qy*. *Exists y Qy* ∈ *sub Q* ⟶ *Gen y Qy*)

sr Q = (*rrf Q* ∧ *rrb Q*)

icdt22_lemma_3: ⟦*rrb Q*; *x* ∈ *fv Q*⟧ ⟹ ∃ *G*. *cov x Q G*

⟦*cov x Q G*; *sat* (*Neg* (*Disj* (*DISJ* (*qps G*)) (*DISJ* ((≈) *x* ‘ *eqs x G*)))) *I σ*⟧ ⟹ *sat Q I σ* = *sat* (*cp* (*Q* ⊥ *x*)) *I σ*

icdt22_lemma_4: ⟦*cov x Q G*; *x* ∈ *fv Q*; *Qqp* ∈ *G*⟧ ⟹ *x* ∈ *fv Qqp* ∧ *fv Qqp* ⊆ *fv Q*

cov x Q G ⟹ *Q* ≜ *Disj* (*Conj Q* (*DISJ* (*qps G*))) (*Disj* (*DISJ* ((λ*y*. *Conj* (*cp* (*Q*[*x* → *y*])) (*x* ≈ *y*)) ‘ *eqs x G*)) (*Conj* (*Q* ⊥ *x*) (*Neg* (*Disj* (*DISJ* (*qps G*)) (*DISJ* ((≈) *x* ‘ *eqs x G*))))))

icdt22_lemma_5: ⟦*cov x Q G*; *x* ∈ *fv Q*⟧ ⟹ *Exists x Q* ≜ *Disj* (*Exists x* (*Conj Q* (*DISJ* (*qps G*)))) (*Disj* (*DISJ* ((λ*y*. *cp* (*Q*[*x* → *y*])) ‘ *eqs x G*)) (*cp* (*Q* ⊥ *x*))) icdt22_example_6: the_res (RB (Conj (Pred ''B'' [Var 0]) (Exists (Suc 0) (Forall (Suc (Suc 0)) (Impl (Pred ''P'' [Var 0, Var (Suc (Suc 0))]) (Pred ''S'' [Var (Suc (Suc 0)), Var (Suc (Suc (Suc 0))), Var (Suc 0)]))))))) = Conj (Pred ''B'' [Var 0]) (Disj (Exists (Suc 0) (Conj (Forall (Suc (Suc 0)) (Impl (Pred ''P'' [Var 0, Var (Suc (Suc 0))]) (Pred ''S'' [Var (Suc (Suc 0)), Var (Suc (Suc (Suc 0))), Var (Suc 0)]))) (Exists (Suc (Suc 0)) (Pred ''S'' [Var (Suc (Suc 0)), Var (Suc (Suc (Suc 0))), Var (Suc 0)])))) (Forall (Suc (Suc 0)) (Neg (Pred ''P'' [Var 0, Var (Suc (Suc 0))]))))

icdt22_lemma_7: $[\![$cov x Q G; x $\in$ fv Q; finite (adom I); $\bigwedge\sigma.\ \neg$ sat (Q $\perp$ x) I $\sigma]\!]$ $\Longrightarrow$ eval Q I = eval (Disj (Conj Q (DISJ (qps G))) (DISJ ((λy. Conj (cp (Q[x $\to$ y])) (x $\approx$ y)) ‘ eqs x G))) I

$[\![$cov x Q G; x $\in$ fv Q; finite (adom I); sat (Q $\perp$ x) I $\sigma]\!]$ $\Longrightarrow$ infinite (eval Q I)

icdt22_lemma_8: inres (icdt22.rb Q) Q' $\Longrightarrow$ rrb Q' $\wedge$ Q $\triangleq$ Q' $\wedge$ fv Q' $\subseteq$ fv Q

icdt22_lemma_9: inres (icdt22.split Q) (Qfin, Qinf) $\Longrightarrow$ sr Qfin $\wedge$ sr Qinf $\wedge$ (fv Qfin = fv Q $\vee$ Qfin = Bool False) $\wedge$ fv Qinf = {} $\wedge$ ($\forall$I. finite (adom I) $\longrightarrow$ (if eval Qinf I = {} then eval Qfin I = eval Q I else infinite (eval Q I)))

icdt22_lemma_8': inres (nres_of (RB Q)) Q' $\Longrightarrow$ rrb Q' $\wedge$ cpropagated Q' $\wedge$ Q $\triangleq$ Q' $\wedge$ fv Q' $\subseteq$ fv Q

icdt22_lemma_9': inres (nres_of (SPLIT Q)) (Qfin, Qinf) $\Longrightarrow$ sr Qfin $\wedge$ sr Qinf $\wedge$ (fv Qfin = fv Q $\vee$ Qfin = Bool False) $\wedge$ fv Qinf = {} $\wedge$ ($\forall$I. finite (adom I) $\longrightarrow$ (if eval Qinf I = {} then eval Qfin I = eval Q I else infinite (eval Q I))) $\wedge$ cpropagated Qfin $\wedge$ cpropagated Qinf

icdt22_example_10: the_res (SPLIT (Disj (Pred ''B'' [Var 0]) (Pred ''P'' [Var 0, Var 1]))) = (Conj (Disj (Pred ''B'' [Var 0]) (Pred ''P'' [Var 0, Var 1])) (Pred ''P'' [Var 0, Var 1]), Exists 0 (Pred ''B'' [Var 0]))

icdt22_example_11: the_res (SPLIT (Conj (Pred ''B'' [Var 0]) (1 $\approx$ 2))) = (Bool False, Exists 0 (Pred ''B'' [Var 0]))

icdt22_example_12: the_res (SPLIT (Conj (Pred ''B'' [Var 0]) (Exists (Suc 0) (Forall (Suc (Suc 0)) (Impl (Pred ''P'' [Var 0, Var (Suc (Suc 0))]) (Pred ''S'' [Var (Suc (Suc 0)), Var (Suc (Suc (Suc 0))), Var (Suc 0)])))))) = (Conj Q_susp_user_rb (Exists (Suc 0) (Exists (Suc (Suc 0)) (Pred ''S'' [Var (Suc (Suc 0)), Var (Suc (Suc (Suc 0))), Var (Suc 0)]))), Exists 0 (Conj (Pred ''B'' [Var 0]) (Forall (Suc (Suc 0)) (Neg (Pred ''P'' [Var 0, Var (Suc (Suc 0))])))))

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] M. Raszyk, D. A. Basin, S. Krstic, and D. Traytel. Practical relational calculus query evaluation. In D. Olteanu and N. Vortmeier, editors, *ICDT 2022*, volume 220 of *LIPIcs*, pages 11:1–11:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.