

Making Arbitrary Relational Calculus Queries Safe-Range

Martin Raszyk Dmitriy Traytel

October 27, 2022

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	2
1.1	Iterated Function Update	2
1.2	Lists and Sets	3
1.3	Equivalence Closure and Classes	4
2	Relational Calculus	6
2.1	First-order Terms	6
2.2	Relational Calculus Syntax and Semantics	7
2.3	Constant Propagation	9
2.4	Big Disjunction	11
2.5	Substitution	13
2.6	Generated Variables	15
2.7	Variable Erasure	19
2.8	Generated Variables and Substitutions	20
2.9	Safe-Range Queries	20
2.10	Simplification	21
2.11	Covered Variables	22
2.12	More on Evaluation	28
3	Restricting Bound Variables	29

4	Refining the Non-Deterministic <i>simplification.rb</i> Function	31
5	Restricting Free Variables	34
6	Refining the Non-Deterministic <i>simplification.split</i> Function	40
7	Examples	41
7.1	Restricting Bounds in the "Suspicious Users" Query	42
7.2	Splitting a Disjunction of Predicates	42
7.3	Splitting a Conjunction with an Equality	42
7.4	Splitting the "Suspicious Users" Query	43
8	Collected Results from the ICDT'22 Paper	43

1 Preliminaries

1.1 Iterated Function Update

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

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

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

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

lemma *extend_nonempty*: $\text{extend } A \text{ } xs \text{ } ys \neq \{\}$
 $\langle \text{proof} \rangle$

lemma *length_extend*: $zs \in \text{extend } A \text{ } xs \text{ } ys \implies \text{length } zs = \text{length } xs$
 $\langle \text{proof} \rangle$

lemma *ex_lookup_extend*: $x \notin A \implies x \in \text{set } xs \implies \exists zs \in \text{extend } A \text{ } xs \text{ } ys. \text{lookup } xs \text{ } zs \text{ } x = d$
 $\langle \text{proof} \rangle$

lemma *restrict_extend*: $A \subseteq \text{set } xs \implies \text{length } ys = \text{card } A \implies zs \in \text{extend } A \text{ } xs \text{ } ys \implies \text{restrict } A \text{ } xs \text{ } zs = ys$
 $\langle \text{proof} \rangle$

lemma *fun_upds_notin[simp]*: $\text{length } xs = \text{length } ys \implies x \notin \text{set } xs \implies (\sigma[xs :=* ys]) x = \sigma x$
 $\langle \text{proof} \rangle$

lemma *fun_upds_twist*: $\text{length } xs = \text{length } ys \implies a \notin \text{set } xs \implies \sigma(a := x)[xs :=* ys] = (\sigma[xs :=* ys])(a := x)$
 $\langle \text{proof} \rangle$

lemma *fun_upds_twist_apply*: $\text{length } xs = \text{length } ys \implies a \notin \text{set } xs \implies a \neq b \implies (\sigma(a := x)[xs :=* ys]) b = (\sigma[xs :=* ys]) b$
 $\langle \text{proof} \rangle$

lemma *fun_upds_extend*:

$x \in A \implies A \subseteq \text{set } xs \implies \text{distinct } xs \implies \text{sorted } xs \implies \text{length } ys = \text{card } A \implies zs \in \text{extend } A \text{ } xs \text{ } ys \implies$
 $(\sigma[xs :=^* zs]) x = (\sigma[\text{sorted_list_of_set } A :=^* ys]) x$

<proof>

lemma *fun_upds_map_self*: $\sigma[xs :=^* \text{map } \sigma \text{ } xs] = \sigma$

<proof>

lemma *fun_upds_single*: $\text{distinct } xs \implies \sigma[xs :=^* \text{map } (\sigma(y := d)) \text{ } xs] = (\text{if } y \in \text{set } xs \text{ then } \sigma(y := d) \text{ else } \sigma)$

<proof>

1.2 Lists and Sets

lemma *find_index_less_size*: $\exists x \in \text{set } xs. P \ x \implies \text{find_index } P \ xs < \text{size } xs$

<proof>

lemma *index_less_size*: $x \in \text{set } xs \implies \text{index } xs \ x < \text{size } xs$

<proof>

lemma *fun_upds_in*: $\text{length } xs = \text{length } ys \implies \text{distinct } xs \implies x \in \text{set } xs \implies (\sigma[xs :=^* ys]) x = ys !$
 $\text{index } xs \ x$

<proof>

lemma *remove_nth_index*: $\text{remove_nth } (\text{index } ys \ y) \ ys = \text{remove1 } y \ ys$

<proof>

lemma *index_remove_nth*: $\text{distinct } xs \implies x \in \text{set } xs \implies \text{index } (\text{remove_nth } i \ xs) \ x = (\text{if } \text{index } xs \ x < i \text{ then } \text{index } xs \ x \text{ else if } i = \text{index } xs \ x \text{ then } \text{length } xs - 1 \text{ else } \text{index } xs \ x - 1)$

<proof>

lemma *insert_nth_nth_index*:

$y \neq z \implies y \in \text{set } ys \implies z \in \text{set } ys \implies \text{length } ys = \text{Suc } (\text{length } xs) \implies \text{distinct } ys \implies$
 $\text{insert_nth } (\text{index } ys \ y) \ x \ xs ! \text{index } ys \ z =$

$xs ! \text{index } (\text{remove1 } y \ ys) \ z$

<proof>

lemma *index_lt_index_remove*: $\text{index } xs \ x < \text{index } xs \ y \implies \text{index } xs \ x = \text{index } (\text{remove1 } y \ xs) \ x$

<proof>

lemma *index_gt_index_remove*: $\text{index } xs \ x > \text{index } xs \ y \implies \text{index } xs \ x = \text{Suc } (\text{index } (\text{remove1 } y \ xs) \ x)$

<proof>

lemma *lookup_map[simp]*: $x \in \text{set } xs \implies \text{lookup } xs \ (\text{map } f \ xs) \ x = f \ x$

<proof>

lemma *in_set_remove_cases*: $P \ z \implies (\forall x \in \text{set } (\text{remove1 } z \ xs). P \ x) \implies x \in \text{set } xs \implies P \ x$

<proof>

lemma *insert_remove_id*: $x \in X \implies X = \text{insert } x \ (X - \{x\})$

<proof>

lemma *infinite_surj*: $\text{infinite } A \implies A \subseteq f \ ^\ast B \implies \text{infinite } B$

<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  $\Rightarrow$  nat where to_nat_nat = id
instance <proof>
end

instantiation list :: (type) infinite begin
definition to_nat_list :: 'a list  $\Rightarrow$  nat where to_nat_list = length
instance <proof>
end

```

1.3 Equivalence Closure and Classes

```

definition symcl where
  symcl r =  $\{(x, y). (x, y) \in r \vee (y, x) \in r\}$ 

definition transymcl where
  transymcl r = trancl (symcl r)

lemma symclp_symcl_eq[pred_set_conv]: symclp  $(\lambda x y. (x, y) \in r)$  =  $(\lambda x y. (x, y) \in \text{symcl } r)$ 
  <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]:  $r \text{ `` } \{x\} = \{\} \longleftrightarrow x \notin \text{Domain } r$ 
  <proof>

lemma sym_symcl[simp]: sym (symcl r)
  <proof>

lemma in_symclI:
   $(a, b) \in r \implies (a, b) \in \text{symcl } r$ 
   $(a, b) \in r \implies (b, a) \in \text{symcl } r$ 
  <proof>

lemma sym_transymcl: sym (transymcl r)
  <proof>

lemma symcl_insert:

```

$\text{symcl } (\text{insert } (x, y) \text{ Qeq}) = \text{insert } (y, x) (\text{insert } (x, y) (\text{symcl } \text{Qeq}))$
(proof)

lemma *equiv_transymcl*: $\text{Equiv_Relations.equiv } (\text{Field } \text{Qeq}) (\text{transymcl } \text{Qeq})$
(proof)

lemma *equiv_quotient_no_empty_class*: $\text{Equiv_Relations.equiv } A \ r \implies \{\} \notin A // r$
(proof)

lemma *classes_cover*: $\bigcup (\text{classes } \text{Qeq}) = \text{Field } \text{Qeq}$
(proof)

lemma *classes_disjoint*: $X \in \text{classes } \text{Qeq} \implies Y \in \text{classes } \text{Qeq} \implies X = Y \vee X \cap Y = \{\}$
(proof)

lemma *classes_nonempty*: $\{\} \notin \text{classes } \text{Qeq}$
(proof)

definition *class x Qeq* = (if $\exists X \in \text{classes } \text{Qeq}. x \in X$ then *Some* (THE $X. X \in \text{classes } \text{Qeq} \wedge x \in X$) else *None*)

lemma *class_Some_eq*: $\text{class } x \ \text{Qeq} = \text{Some } X \longleftrightarrow X \in \text{classes } \text{Qeq} \wedge x \in X$
(proof)

lemma *class_None_eq*: $\text{class } x \ \text{Qeq} = \text{None} \longleftrightarrow x \notin \text{Field } \text{Qeq}$
(proof)

lemma *insert_Image_triv*: $x \notin r \implies \text{insert } (x, y) \ \text{Qeq} \text{ `` } r = \text{Qeq} \text{ `` } r$
(proof)

lemma *Un1_Image_triv*: $\text{Domain } B \cap r = \{\} \implies (A \cup B) \text{ `` } r = A \text{ `` } r$
(proof)

lemma *Un2_Image_triv*: $\text{Domain } A \cap r = \{\} \implies (A \cup B) \text{ `` } r = B \text{ `` } r$
(proof)

lemma *classes_empty*: $\text{classes } \{\} = \{\}$
(proof)

lemma *ex_class*: $x \in \text{Field } \text{Qeq} \implies \exists X. \text{class } x \ \text{Qeq} = \text{Some } X \wedge x \in X$
(proof)

lemma *equivD*:
 $\text{Equiv_Relations.equiv } A \ r \implies \text{refl_on } A \ r$
 $\text{Equiv_Relations.equiv } A \ r \implies \text{sym } r$
 $\text{Equiv_Relations.equiv } A \ r \implies \text{trans } r$
(proof)

lemma *transymcl_into*:
 $(x, y) \in r \implies (x, y) \in \text{transymcl } r$
 $(x, y) \in r \implies (y, x) \in \text{transymcl } r$
(proof)

lemma *transymcl_self*:
 $(x, y) \in r \implies (x, x) \in \text{transymcl } r$
 $(x, y) \in r \implies (y, y) \in \text{transymcl } r$
(proof)

lemma *transymcl_trans*: $(x, y) \in \text{transymcl } r \implies (y, z) \in \text{transymcl } r \implies (x, z) \in \text{transymcl } r$
 <proof>

lemma *transymcl_sym*: $(x, y) \in \text{transymcl } r \implies (y, x) \in \text{transymcl } r$
 <proof>

lemma *edge_same_class*: $X \in \text{classes } Qeq \implies (a, b) \in Qeq \implies a \in X \iff b \in X$
 <proof>

lemma *Field_transymcl_self*: $a \in \text{Field } Qeq \implies (a, a) \in \text{transymcl } Qeq$
 <proof>

lemma *transymcl_insert*: $\text{transymcl } (\text{insert } (a, b) \text{ } Qeq) = \text{transymcl } Qeq \cup \{(a,a),(b,b)\} \cup$
 $((\text{transymcl } Qeq \cup \{(a, a), (b, b)\}) \circ \{(a, b), (b, a)\} \circ (\text{transymcl } Qeq \cup \{(a, a), (b, b)\}) - \text{transymcl } Qeq)$
 <proof>

lemma *transymcl_insert_both_new*: $a \notin \text{Field } Qeq \implies b \notin \text{Field } Qeq \implies$
 $\text{transymcl } (\text{insert } (a, b) \text{ } Qeq) = \text{transymcl } Qeq \cup \{(a,a),(b,b),(a,b),(b,a)\}$
 <proof>

lemma *transymcl_insert_same_class*: $(x, y) \in \text{transymcl } Qeq \implies \text{transymcl } (\text{insert } (x, y) \text{ } Qeq) =$
 $\text{transymcl } Qeq$
 <proof>

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

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

2 Relational Calculus

2.1 First-order Terms

datatype 'a term = Const 'a | Var nat

type_synonym 'a val = nat \Rightarrow 'a

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

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

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

fun *eval_term* :: 'a val \Rightarrow 'a term \Rightarrow 'a (infix · 60) **where**

$eval_term\ \sigma\ (Const\ c) = c$
 $| eval_term\ \sigma\ (Var\ n) = \sigma\ n$

definition $eval_terms :: 'a\ val \Rightarrow ('a\ term)\ list \Rightarrow 'a\ list\ (infix\ \odot\ 60)$ **where**
 $eval_terms\ \sigma\ ts = map\ (eval_term\ \sigma)\ ts$

lemma $finite_set_term: finite\ (set_term\ t)$
 $\langle proof \rangle$

lemma $finite_fv_term_set: finite\ (fv_term_set\ t)$
 $\langle proof \rangle$

lemma $fv_term_setD: n \in fv_term_set\ t \Longrightarrow t = Var\ n$
 $\langle proof \rangle$

lemma $fv_term_set_cong: fv_term_set\ t = fv_term_set\ (map_term\ f\ t)$
 $\langle proof \rangle$

lemma $fv_terms_setI: Var\ m \in set\ ts \Longrightarrow m \in fv_terms_set\ ts$
 $\langle proof \rangle$

lemma $fv_terms_setD: m \in fv_terms_set\ ts \Longrightarrow Var\ m \in set\ ts$
 $\langle proof \rangle$

lemma $finite_fv_terms_set: finite\ (fv_terms_set\ ts)$
 $\langle proof \rangle$

lemma $fv_terms_set_cong: fv_terms_set\ ts = fv_terms_set\ (map\ (map_term\ f)\ ts)$
 $\langle proof \rangle$

lemma $eval_term_cong: (\bigwedge n. n \in fv_term_set\ t \Longrightarrow \sigma\ n = \sigma'\ n) \Longrightarrow$
 $eval_term\ \sigma\ t = eval_term\ \sigma'\ t$
 $\langle proof \rangle$

lemma $eval_terms_fv_terms_set: \sigma \odot ts = \sigma' \odot ts \Longrightarrow n \in fv_terms_set\ ts \Longrightarrow \sigma\ n = \sigma'\ n$
 $\langle proof \rangle$

lemma $eval_terms_cong: (\bigwedge n. n \in fv_terms_set\ ts \Longrightarrow \sigma\ n = \sigma'\ n) \Longrightarrow$
 $eval_terms\ \sigma\ ts = eval_terms\ \sigma'\ ts$
 $\langle proof \rangle$

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

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

definition exists where $\text{exists}\ x\ Q = (\text{if } x \in \text{fv}\ Q \text{ then } \text{Exists}\ x\ Q \text{ else } Q)$

abbreviation $\text{forall}\ x\ Q \equiv \text{Neg}\ (\text{Exists}\ x\ (\text{Neg}\ Q))$

abbreviation $\text{forall}\ x\ Q \equiv \text{Neg}\ (\text{exists}\ x\ (\text{Neg}\ Q))$

abbreviation $\text{Impl}\ Q1\ Q2 \equiv \text{Disj}\ (\text{Neg}\ Q1)\ Q2$

definition $\text{EXISTS}\ xs\ Q = \text{fold}\ \text{Exists}\ xs\ Q$

abbreviation close where

$\text{close}\ Q \equiv \text{EXISTS}\ (\text{sorted_list_of_set}\ (\text{fv}\ Q))\ Q$

lemma $\text{fv_exists}[simp]: \text{fv}\ (\text{exists}\ x\ Q) = \text{fv}\ Q - \{x\}$
 $\langle \text{proof} \rangle$

lemma $\text{fv_EXISTS}: \text{fv}\ (\text{EXISTS}\ xs\ Q) = \text{fv}\ Q - \text{set}\ xs$
 $\langle \text{proof} \rangle$

lemma $\text{exists_Exists}: x \in \text{fv}\ Q \implies \text{exists}\ x\ Q = \text{Exists}\ x\ Q$
 $\langle \text{proof} \rangle$

lemma $\text{is_Bool_exists}[simp]: \text{is_Bool}\ (\text{exists}\ x\ Q) = \text{is_Bool}\ Q$
 $\langle \text{proof} \rangle$

lemma $\text{finite_fv}[simp]: \text{finite}\ (\text{fv}\ \varphi)$
 $\langle \text{proof} \rangle$

lemma $\text{fv_close}[simp]: \text{fv}\ (\text{close}\ Q) = \{\}$
 $\langle \text{proof} \rangle$

type_synonym $'a\ \text{table} = ('a\ \text{list})\ \text{set}$

type_synonym $('a, 'b)\ \text{intp} = 'b \times \text{nat} \Rightarrow 'a\ \text{table}$

definition $\text{adom} :: ('a, 'b)\ \text{intp} \Rightarrow 'a\ \text{set}\ \text{where}$

$\text{adom}\ I = (\bigcup \text{rn}. \bigcup \text{xs} \in I\ \text{rn}. \text{set}\ \text{xs})$

fun $\text{sat} :: ('a, 'b)\ \text{fmla} \Rightarrow ('a, 'b)\ \text{intp} \Rightarrow 'a\ \text{val} \Rightarrow \text{bool}\ \text{where}$

$\text{sat}\ (\text{Pred}\ r\ ts)\ I\ \sigma \longleftrightarrow \sigma \odot ts \in I\ (r, \text{length}\ ts)$

$| \text{sat}\ (\text{Bool}\ b)\ I\ \sigma \longleftrightarrow b$

$| \text{sat}\ (Eq\ x\ t')\ I\ \sigma \longleftrightarrow \sigma\ x = \sigma \cdot t'$

$| \text{sat}\ (\text{Neg}\ \varphi)\ I\ \sigma \longleftrightarrow \neg \text{sat}\ \varphi\ I\ \sigma$

$| \text{sat}\ (\text{Conj}\ \varphi\ \psi)\ I\ \sigma \longleftrightarrow \text{sat}\ \varphi\ I\ \sigma \wedge \text{sat}\ \psi\ I\ \sigma$

$| \text{sat}\ (\text{Disj}\ \varphi\ \psi)\ I\ \sigma \longleftrightarrow \text{sat}\ \varphi\ I\ \sigma \vee \text{sat}\ \psi\ I\ \sigma$

$| \text{sat}\ (\text{Exists}\ z\ \varphi)\ I\ \sigma \longleftrightarrow (\exists x. \text{sat}\ \varphi\ I\ (\sigma(z := x)))$

lemma $\text{sat_fv_cong}: (\bigwedge n. n \in \text{fv}\ \varphi \implies \sigma\ n = \sigma'\ n) \implies$

$\text{sat}\ \varphi\ I\ \sigma \longleftrightarrow \text{sat}\ \varphi\ I\ \sigma'$

$\langle \text{proof} \rangle$

lemma $\text{sat_fun_upd}: n \notin \text{fv}\ Q \implies \text{sat}\ Q\ I\ (\sigma(n := z)) = \text{sat}\ Q\ I\ \sigma$

$\langle \text{proof} \rangle$

lemma $\text{sat_exists}[simp]: \text{sat}\ (\text{exists}\ n\ Q)\ I\ \sigma = (\exists x. \text{sat}\ Q\ I\ (\sigma(n := x)))$

$\langle \text{proof} \rangle$

abbreviation *eq* (**infix** \approx 80) **where**

$x \approx y \equiv Eq\ x\ (Var\ y)$

definition *equiv* (**infix** \triangleq 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*]: $Q1 \triangleq Q2 \implies Q2 \triangleq Q1$

<proof>

lemma *equiv_trans*[*trans*]: $Q1 \triangleq Q2 \implies Q2 \triangleq Q3 \implies Q1 \triangleq Q3$

<proof>

lemma *equiv_Neg_cong*[*simp*]: $Q \triangleq Q' \implies Neg\ Q \triangleq Neg\ Q'$

<proof>

lemma *equiv_Conj_cong*[*simp*]: $Q1 \triangleq Q1' \implies Q2 \triangleq Q2' \implies Conj\ Q1\ Q2 \triangleq Conj\ Q1'\ Q2'$

<proof>

lemma *equiv_Disj_cong*[*simp*]: $Q1 \triangleq Q1' \implies Q2 \triangleq Q2' \implies Disj\ Q1\ Q2 \triangleq Disj\ Q1'\ Q2'$

<proof>

lemma *equiv_Exists_cong*[*simp*]: $Q \triangleq Q' \implies Exists\ x\ Q \triangleq Exists\ x\ Q'$

<proof>

lemma *equiv_Exists_exists_cong*[*simp*]: $Q \triangleq Q' \implies Exists\ x\ Q \triangleq exists\ x\ Q'$

<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*]:

$list_all2\ (\triangleq)\ xs\ ys \implies b \triangleq c \implies foldr\ Disj\ xs\ b \triangleq foldr\ Disj\ ys\ c$

<proof>

lemma *Exists_nonfree_equiv*: $x \notin fv\ Q \implies Exists\ x\ Q \triangleq Q$

<proof>

2.3 Constant Propagation

fun *cp* **where**

$cp\ (Eq\ x\ t) = (case\ t\ of\ Var\ y \Rightarrow if\ x = y\ then\ Bool\ True\ else\ x \approx y\ | _ \Rightarrow Eq\ x\ t)$

| $cp\ (Neg\ Q) = (let\ Q' = cp\ Q\ in\ if\ is_Bool\ Q'\ then\ Bool\ (\neg\ un_Bool\ Q')\ else\ Neg\ Q')$

| $cp\ (Conj\ Q1\ Q2) =$

$(let\ Q1' = cp\ Q1;\ Q2' = cp\ Q2\ in$

$if\ is_Bool\ Q1'\ then\ if\ un_Bool\ Q1'\ then\ Q2'\ else\ Bool\ False$

$else\ if\ is_Bool\ Q2'\ then\ if\ un_Bool\ Q2'\ then\ Q1'\ else\ Bool\ False$

$else\ Conj\ Q1'\ Q2')$

| $cp\ (Disj\ Q1\ Q2) =$

$(let\ Q1' = cp\ Q1;\ Q2' = cp\ Q2\ in$

$if\ is_Bool\ Q1'\ then\ if\ un_Bool\ Q1'\ then\ Bool\ True\ else\ Q2'$

$else\ if\ is_Bool\ Q2'\ then\ if\ un_Bool\ Q2'\ then\ Bool\ True\ else\ Q1')$

$else\ Disj\ Q1'\ Q2')$
 $|\ cp\ (Exists\ x\ Q) = exists\ x\ (cp\ Q)$
 $| \ cp\ Q = Q$

lemma fv_cp : $fv\ (cp\ Q) \subseteq fv\ Q$
 $\langle proof \rangle$

lemma $cp_exists[simp]$: $cp\ (exists\ x\ Q) = exists\ x\ (cp\ Q)$
 $\langle proof \rangle$

fun $nocp$ **where**
 $nocp\ (Bool\ b) = False$
 $| \ nocp\ (Pred\ p\ ts) = True$
 $| \ nocp\ (Eq\ x\ t) = (t \neq Var\ x)$
 $| \ nocp\ (Neg\ Q) = nocp\ Q$
 $| \ nocp\ (Conj\ Q1\ Q2) = (nocp\ Q1 \wedge nocp\ Q2)$
 $| \ nocp\ (Disj\ Q1\ Q2) = (nocp\ Q1 \wedge nocp\ Q2)$
 $| \ nocp\ (Exists\ x\ Q) = (x \in fv\ Q \wedge nocp\ Q)$

lemma $nocp_exists[simp]$: $nocp\ (exists\ x\ Q) = nocp\ Q$
 $\langle proof \rangle$

lemma $nocp_cp_triv$: $nocp\ Q \implies cp\ Q = Q$
 $\langle proof \rangle$

lemma $is_Bool_cp_triv$: $is_Bool\ Q \implies cp\ Q = Q$
 $\langle proof \rangle$

lemma $nocp_cp_or_is_Bool$: $nocp\ (cp\ Q) \vee is_Bool\ (cp\ Q)$
 $\langle proof \rangle$

lemma $cp_idem[simp]$: $cp\ (cp\ Q) = cp\ Q$
 $\langle proof \rangle$

lemma $sat_cp[simp]$: $sat\ (cp\ Q)\ I\ \sigma = sat\ Q\ I\ \sigma$
 $\langle proof \rangle$

lemma $equiv_cp_cong[simp]$: $Q \triangleq Q' \implies cp\ Q \triangleq cp\ Q'$
 $\langle proof \rangle$

lemma $equiv_cp[simp]$: $cp\ Q \triangleq Q$
 $\langle proof \rangle$

definition $cpropagated$ **where** $cpropagated\ Q = (nocp\ Q \vee is_Bool\ Q)$

lemma $cpropagated_cp[simp]$: $cpropagated\ (cp\ Q)$
 $\langle proof \rangle$

lemma $nocp_cpropagated[simp]$: $nocp\ Q \implies cpropagated\ Q$
 $\langle proof \rangle$

lemma $cpropagated_cp_triv$: $cpropagated\ Q \implies cp\ Q = Q$
 $\langle proof \rangle$

lemma $cpropagated_nocp$: $cpropagated\ Q \implies x \in fv\ Q \implies nocp\ Q$
 $\langle proof \rangle$

lemma $cpropagated_simps[simp]$:

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

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]*: $sat\ (foldr\ Disj\ xs\ Q)\ I\ \sigma = (\exists Q \in set\ xs \cup \{Q\}. sat\ Q\ I\ \sigma)$
 $\langle proof \rangle$

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

lemma *sat_DISJ[simp]*: $finite\ G \implies sat\ (DISJ\ G)\ I\ \sigma = (\exists Q \in G. sat\ Q\ I\ \sigma)$
 $\langle proof \rangle$

lemma *foldr_Disj_equiv*: $insert\ Q\ (set\ Qs) = insert\ Q'\ (set\ Qs') \implies foldr\ Disj\ Qs\ Q \triangleq foldr\ Disj\ Qs'\ Q'$
 $\langle proof \rangle$

lemma *foldr1_Disj_equiv*: $set\ Qs = set\ Qs' \implies foldr1\ Disj\ Qs\ (Bool\ False) \triangleq foldr1\ Disj\ Qs'\ (Bool\ False)$
 $\langle proof \rangle$

lemma *foldr1_Disj_equiv_cong[simp]*:
 $list_all2\ (\triangleq)\ xs\ ys \implies b \triangleq c \implies foldr1\ Disj\ xs\ b \triangleq foldr1\ Disj\ ys\ c$
 $\langle proof \rangle$

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

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

lemma *Exists_DISJ*:
 $finite\ Q \implies Exists\ x\ (DISJ\ Q) \triangleq DISJ\ (exists\ x\ 'Q)$
 $\langle proof \rangle$

lemma *Exists_cp_DISJ*:
 $finite\ Q \implies Exists\ x\ (cp\ (DISJ\ Q)) \triangleq DISJ\ (exists\ x\ 'Q)$
 $\langle proof \rangle$

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

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

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

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

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

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

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

lemma *DISJ_insert_reorder'*: $finite\ \mathcal{Q} \implies finite\ \mathcal{Q}' \implies DISJ (insert\ (Disj\ (DISJ\ \mathcal{Q}')\ Q2)\ \mathcal{Q}) \triangleq DISJ (insert\ Q2\ (\mathcal{Q}' \cup \mathcal{Q}))$
 ⟨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*: $finite\ \mathcal{Q} \implies fv (DISJ\ \mathcal{Q}) \subseteq (\bigcup Q \in \mathcal{Q}.\ fv\ Q)$
 ⟨proof⟩

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

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

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

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

lemma *Exists_in_sub_cp_foldr_Disj*:

$Exists\ x\ Q' \in sub\ (cp\ (foldr\ Disj\ Qs\ Q)) \implies Exists\ x\ Q' \in sub\ (cp\ Q) \vee (\exists Q \in set\ Qs.\ Exists\ x\ Q' \in sub\ (cp\ Q))$
 ⟨proof⟩

lemma *Exists_in_sub_cp_foldr1_Disj*:

$Exists\ x\ Q' \in sub\ (cp\ (foldr1\ Disj\ Qs\ Q)) \implies Qs = [] \wedge Exists\ x\ Q' \in sub\ (cp\ Q) \vee (\exists Q \in set\ Qs.\ Exists\ x\ Q' \in sub\ (cp\ Q))$
 ⟨proof⟩

lemma *Exists_in_sub_cp_DISJ*: $Exists\ x\ Q' \in sub\ (cp\ (DISJ\ Q)) \implies finite\ Q \implies (\exists Q \in Q.\ Exists\ x\ Q' \in sub\ (cp\ Q))$
 ⟨proof⟩

lemma *Exists_in_sub_foldr_Disj*:

$Exists\ x\ Q' \in sub\ (foldr\ Disj\ Qs\ Q) \implies Exists\ x\ Q' \in sub\ Q \vee (\exists Q \in set\ Qs.\ Exists\ x\ Q' \in sub\ Q)$
 ⟨proof⟩

lemma *Exists_in_sub_foldr1_Disj*:

$Exists\ x\ Q' \in sub\ (foldr1\ Disj\ Qs\ Q) \implies Qs = [] \wedge Exists\ x\ Q' \in sub\ Q \vee (\exists Q \in set\ Qs.\ Exists\ x\ Q' \in sub\ Q)$
 ⟨proof⟩

lemma *Exists_in_sub_DISJ*: $Exists\ x\ Q' \in sub\ (DISJ\ Q) \implies finite\ Q \implies (\exists Q \in Q.\ Exists\ x\ Q' \in sub\ Q)$
 ⟨proof⟩

2.5 Substitution

fun *subst_term* ($_ _ \rightarrow t _$) [*90*, 0, 0] [*91*] **where**

$Var\ z[x \rightarrow t\ y] = Var\ (if\ x = z\ then\ y\ else\ z)$
 $| Const\ c[x \rightarrow t\ y] = Const\ c$

abbreviation *subst_term* ($_ _ \rightarrow t^* _$) [*90*, 0, 0] [*91*] **where**

$t[xs \rightarrow t^* ys] \equiv fold\ (\lambda(x, y)\ t.\ t[x \rightarrow t\ y])\ (zip\ xs\ ys)\ t$

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

lemma *fv_subst_term[simp]*: $fv_term_set\ (t[x \rightarrow t\ y]) =$

$(if\ x \in fv_term_set\ t\ then\ insert\ y\ (fv_term_set\ t - \{x\})\ else\ fv_term_set\ t)$
 ⟨proof⟩

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

function (*sequential*) *subst* :: ('a, 'b) fmla \Rightarrow nat \Rightarrow nat \Rightarrow ('a, 'b) fmla ($_ _ \rightarrow _$) [*90*, 0, 0] [*91*]
where

$Bool\ t[x \rightarrow y] = Bool\ t$
 $| Pred\ p\ ts[x \rightarrow y] = Pred\ p\ (map\ (\lambda t.\ t[x \rightarrow t\ y])\ ts)$
 $| Eq\ z\ t[x \rightarrow y] = Eq\ (if\ z = x\ then\ y\ else\ z)\ (t[x \rightarrow t\ y])$
 $| Neg\ Q[x \rightarrow y] = Neg\ (Q[x \rightarrow y])$
 $| Conj\ Q1\ Q2[x \rightarrow y] = Conj\ (Q1[x \rightarrow y])\ (Q2[x \rightarrow y])$
 $| Disj\ Q1\ Q2[x \rightarrow y] = Disj\ (Q1[x \rightarrow y])\ (Q2[x \rightarrow y])$
 $| Exists\ z\ Q[x \rightarrow y] = (if\ x = z\ then\ Exists\ x\ Q\ else$
 $if\ z = y\ then\ let\ z' = fresh2\ x\ y\ Q\ in\ Exists\ z'\ (Q[z \rightarrow z'][x \rightarrow y])\ else\ Exists\ z\ (Q[x \rightarrow y]))$

<proof>

abbreviation *substs* ($_ _ \rightarrow^* _$) [90, 0, 0] 91) **where**

$Q[xs \rightarrow^* ys] \equiv \text{fold } (\lambda(x, y) Q. Q[x \rightarrow y]) (\text{zip } xs \ ys) \ Q$

lemma *size_subst_p[simp]*: $\text{subst_dom } (Q, x, y) \implies \text{size } (Q[x \rightarrow y]) = \text{size } Q$

<proof>

termination *<proof>*

lemma *size_subst[simp]*: $\text{size } (Q[x \rightarrow y]) = \text{size } Q$

<proof>

lemma *fresh2_gt*:

$x < \text{fresh2 } x \ y \ Q$

$y < \text{fresh2 } x \ y \ Q$

$z \in \text{fv } Q \implies z < \text{fresh2 } x \ y \ Q$

<proof>

lemma *fresh2*:

$x \neq \text{fresh2 } x \ y \ Q$

$y \neq \text{fresh2 } x \ y \ Q$

$\text{fresh2 } x \ y \ Q \notin \text{fv } Q$

<proof>

lemma *fv_subst*:

$\text{fv } (Q[x \rightarrow y]) = (\text{if } x \in \text{fv } Q \text{ then insert } y \ (\text{fv } Q - \{x\}) \text{ else } \text{fv } Q)$

<proof>

lemma *subst_term_triv*: $x \notin \text{fv_term_set } t \implies t[x \rightarrow t \ y] = t$

<proof>

lemma *subst_exists*: $\text{exists } z \ Q[x \rightarrow y] = (\text{if } z \in \text{fv } Q \text{ then if } x = z \text{ then exists } x \ Q \text{ else}$

$\text{if } z = y \text{ then let } z' = \text{fresh2 } x \ y \ Q \text{ in exists } z' \ (Q[z \rightarrow z'] [x \rightarrow y]) \text{ else exists } z \ (Q[x \rightarrow y]) \text{ else } Q[x$

$\rightarrow y])$

<proof>

lemma *eval_subst[simp]*: $\sigma \cdot t[x \rightarrow t \ y] = \sigma(x := \sigma \ y) \cdot t$

<proof>

lemma *sat_subst[simp]*: $\text{sat } (Q[x \rightarrow y]) \ I \ \sigma = \text{sat } Q \ I \ (\sigma(x := \sigma \ y))$

<proof>

lemma *substs_Bool[simp]*: $\text{length } xs = \text{length } ys \implies \text{Bool } b[xs \rightarrow^* \ ys] = \text{Bool } b$

<proof>

lemma *substs_Neg[simp]*: $\text{length } xs = \text{length } ys \implies \text{Neg } Q[xs \rightarrow^* \ ys] = \text{Neg } (Q[xs \rightarrow^* \ ys])$

<proof>

lemma *substs_Conj[simp]*: $\text{length } xs = \text{length } ys \implies \text{Conj } Q1 \ Q2[xs \rightarrow^* \ ys] = \text{Conj } (Q1[xs \rightarrow^* \ ys])$

$(Q2[xs \rightarrow^* \ ys])$

<proof>

lemma *substs_Disj[simp]*: $\text{length } xs = \text{length } ys \implies \text{Disj } Q1 \ Q2[xs \rightarrow^* \ ys] = \text{Disj } (Q1[xs \rightarrow^* \ ys])$

$(Q2[xs \rightarrow^* \ ys])$

<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]: $length\ xs = length\ ys \implies length\ (substs_src\ z\ xs\ ys\ Q) = length\ (substs_dst\ z\ xs\ ys\ Q)$
 <proof>

lemma substs_Exists[simp]: $length\ xs = length\ ys \implies$

$Exists\ z\ Q[xs \rightarrow^* ys] = Exists\ (substs_bd\ z\ xs\ ys\ Q)\ (Q[substs_src\ z\ xs\ ys\ Q \rightarrow^* substs_dst\ z\ xs\ ys\ Q])$
 <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]: $length\ xs = length\ ys \implies (Eq\ x\ t)[xs \rightarrow^* ys] = Eq\ (subst_var\ xs\ ys\ x)\ (t[xs \rightarrow^* ys])$
 <proof>

lemma substs_term_Var[simp]: $length\ xs = length\ ys \implies (Var\ x)[xs \rightarrow^* ys] = Var\ (subst_var\ xs\ ys\ x)$
 <proof>

lemma substs_term_Const[simp]: $length\ xs = length\ ys \implies (Const\ c)[xs \rightarrow^* ys] = Const\ c$
 <proof>

lemma in_fv_substs:

$length\ xs = length\ ys \implies x \in fv\ Q \implies subst_var\ xs\ ys\ x \in fv\ (Q[xs \rightarrow^* ys])$
 <proof>

lemma exists_cp_subst: $x \neq y \implies exists\ x\ (cp\ (Q[x \rightarrow y])) = cp\ (Q[x \rightarrow y])$
 <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

```

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

inductive gen' where

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

inductive qp where

$\text{ap}: \text{ap } Q \Longrightarrow \text{qp } Q$
 $| \text{exists}: \text{qp } Q \Longrightarrow \text{qp } (\text{exists } x \ Q)$

lemma $\text{qp_Exists}: \text{qp } Q \Longrightarrow x \in \text{fv } Q \Longrightarrow \text{qp } (\text{Exists } x \ Q)$
 $\langle \text{proof} \rangle$

lemma $\text{qp_ExistsE}: \text{qp } (\text{Exists } x \ Q) \Longrightarrow (\text{qp } Q \Longrightarrow x \in \text{fv } Q \Longrightarrow R) \Longrightarrow R$
 $\langle \text{proof} \rangle$

fun qp_impl where

$\text{qp_impl } (\text{Eq } x \ (\text{Const } c)) = \text{True}$
 $| \text{qp_impl } (\text{Pred } x \ ts) = \text{True}$
 $| \text{qp_impl } (\text{Exists } x \ Q) = (x \in \text{fv } Q \wedge \text{qp } Q)$
 $| \text{qp_impl } _ = \text{False}$

lemma $\text{qp_imp_qp_impl}: \text{qp } Q \Longrightarrow \text{qp_impl } Q$
 $\langle \text{proof} \rangle$

lemma $\text{qp_impl_imp_qp}: \text{qp_impl } Q \Longrightarrow \text{qp } Q$
 $\langle \text{proof} \rangle$

lemma $\text{qp_code}[code]: \text{qp } Q = \text{qp_impl } Q$
 $\langle \text{proof} \rangle$

lemma $\text{ap_cp}: \text{ap } Q \Longrightarrow \text{ap } (\text{cp } Q)$
 $\langle \text{proof} \rangle$

lemma $\text{qp_cp}: \text{qp } Q \Longrightarrow \text{qp } (\text{cp } Q)$
 $\langle \text{proof} \rangle$

lemma $\text{ap_subst}: \text{ap } Q \Longrightarrow \text{length } xs = \text{length } ys \Longrightarrow \text{ap } (Q[xs \rightarrow^* ys])$
 $\langle \text{proof} \rangle$

lemma $\text{ap_subst}': \text{ap } (Q[x \rightarrow y]) \Longrightarrow \text{ap } Q$
 $\langle \text{proof} \rangle$

lemma $\text{qp_subst}: \text{qp } Q \Longrightarrow \text{length } xs = \text{length } ys \Longrightarrow \text{qp } (Q[xs \rightarrow^* ys])$
 $\langle \text{proof} \rangle$

lemma *qp_subst*: $qp\ Q \implies qp\ (Q[x \rightarrow y])$
<proof>

lemma *qp_Neg[dest]*: $qp\ (Neg\ Q) \implies False$
<proof>

lemma *qp_Disj[dest]*: $qp\ (Disj\ Q1\ Q2) \implies False$
<proof>

lemma *qp_Conj[dest]*: $qp\ (Conj\ Q1\ Q2) \implies False$
<proof>

lemma *qp_eq[dest]*: $qp\ (x \approx y) \implies False$
<proof>

lemma *qp_subst'*: $qp\ (Q[x \rightarrow y]) \implies qp\ Q$
<proof>

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

lemma *gen_qp*: $gen\ x\ Q\ G \implies Qqp \in G \implies qp\ Qqp$
<proof>

lemma *gen'_qp*: $gen'\ x\ Q\ G \implies Qqp \in G \implies qp\ Qqp$
<proof>

lemma *ap_cp_triv*: $ap\ Q \implies cp\ Q = Q$
<proof>

lemma *qp_cp_triv*: $qp\ Q \implies cp\ Q = Q$
<proof>

lemma *ap_cp_subst_triv*: $ap\ Q \implies cp\ (Q[x \rightarrow y]) = Q[x \rightarrow y]$
<proof>

lemma *qp_cp_subst_triv*: $qp\ Q \implies cp\ (Q[x \rightarrow y]) = Q[x \rightarrow y]$
<proof>

lemma *gen_nocp_intros*:
 $gen\ y\ Q\ G \implies gen\ x\ (Conj\ Q\ (x \approx y))\ ((\lambda Q. Q[y \rightarrow x])\ 'G)$
 $gen\ y\ Q\ G \implies gen\ x\ (Conj\ Q\ (y \approx x))\ ((\lambda Q. Q[y \rightarrow x])\ 'G)$
<proof>

lemma *gen'_cp_intros*:
 $gen'\ y\ Q\ G \implies gen'\ x\ (Conj\ Q\ (x \approx y))\ ((\lambda Q. cp\ (Q[y \rightarrow x]))\ 'G)$
 $gen'\ y\ Q\ G \implies gen'\ x\ (Conj\ Q\ (y \approx x))\ ((\lambda Q. cp\ (Q[y \rightarrow x]))\ 'G)$
<proof>

lemma *gen'_gen*: $gen'\ x\ Q\ G \implies gen\ x\ Q\ G$
<proof>

lemma *gen_gen'*: $gen\ x\ Q\ G \implies gen'\ x\ Q\ G$
<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 $qp_Gen: qp\ Q \implies x \in fv\ Q \implies Gen\ x\ Q$
 ⟨proof⟩

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

lemma $gen_foldr_Disj:$
 $list_all2\ (gen\ x)\ Qs\ Gs \implies gen\ x\ Q\ G \implies GG = G \cup (\bigcup G \in set\ Gs. G) \implies$
 $gen\ x\ (foldr\ Disj\ Qs\ Q)\ GG$
 ⟨proof⟩

lemma $gen_foldr1_Disj:$
 $list_all2\ (gen\ x)\ Qs\ Gs \implies gen\ x\ Q\ G \implies GG = (if\ Qs = []\ then\ G\ else\ (\bigcup G \in set\ Gs. G)) \implies$
 $gen\ x\ (foldr1\ Disj\ Qs\ Q)\ GG$
 ⟨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 $gen_Gen_cp: gen\ x\ Q\ G \implies Gen\ x\ (cp\ Q)$
 ⟨proof⟩

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

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

lemma $Gen_cp_DISJ: finite\ \mathcal{Q} \implies \forall Q \in \mathcal{Q}. qp\ Q \wedge x \in fv\ Q \implies Gen\ x\ (cp\ (DISJ\ \mathcal{Q}))$
 ⟨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 $gen_empty_cp: gen\ z\ Q\ G \implies G = \{\} \implies cp\ Q = Bool\ False$
 ⟨proof⟩

inductive *genempty where*

$genempty\ (Bool\ False)$
 | $genempty\ Q \implies genempty\ (Neg\ (Neg\ Q))$
 | $genempty\ (Conj\ (Neg\ Q1)\ (Neg\ Q2)) \implies genempty\ (Neg\ (Disj\ Q1\ Q2))$
 | $genempty\ (Disj\ (Neg\ Q1)\ (Neg\ Q2)) \implies genempty\ (Neg\ (Conj\ Q1\ Q2))$
 | $genempty\ Q1 \implies genempty\ Q2 \implies genempty\ (Disj\ Q1\ Q2)$
 | $genempty\ Q1 \vee genempty\ Q2 \implies genempty\ (Conj\ Q1\ Q2)$
 | $genempty\ Q \implies genempty\ (Conj\ Q\ (x \approx y))$
 | $genempty\ Q \implies genempty\ (Conj\ Q\ (y \approx x))$

| *genempty* $Q \implies \text{genempty } (\text{Exists } y \ Q)$

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

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

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

lemma *genempty_cp*: $\text{genempty } Q \implies \text{cp } Q = \text{Bool False}$
 ⟨*proof*⟩

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

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

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

lemma *gen_fv*:
 $\text{gen } x \ Q \ G \implies Qqp \in G \implies x \in \text{fv } Qqp \wedge \text{fv } Qqp \subseteq \text{fv } Q$
 ⟨*proof*⟩

lemma *gen_sat*:
fixes $x :: \text{nat}$
shows $\text{gen } x \ Q \ G \implies \text{sat } Q \ I \ \sigma \implies \exists Qqp \in G. \text{sat } Qqp \ I \ \sigma$
 ⟨*proof*⟩

2.7 Variable Erasure

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

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

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

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

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

lemma *exists_cp_erase*: $\text{exists } x (\text{cp } (Q \perp x)) = \text{cp } (Q \perp x)$
 ⟨proof⟩

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

2.8 Generated Variables and Substitutions

lemma *gen_Gen_cp_substs*: $\text{gen } z Q G \implies \text{length } xs = \text{length } ys \implies$
 $\text{Gen } (\text{subst_var } xs \text{ } ys \ z) (\text{cp } (Q[xs \rightarrow^* ys]))$
 ⟨proof⟩

lemma *Gen_cp_substs*: $\text{Gen } z Q \implies \text{length } xs = \text{length } ys \implies \text{Gen } (\text{subst_var } xs \text{ } ys \ z) (\text{cp } (Q[xs \rightarrow^* ys]))$
 ⟨proof⟩

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

lemma *substs_bd_fv*: $\text{length } xs = \text{length } ys \implies \text{substs_bd } z \text{ } xs \text{ } ys \ Q \in \text{fv } (Q[\text{substs_src } z \text{ } xs \text{ } ys \ Q \rightarrow^* \text{substs_dst } z \text{ } xs \text{ } ys \ Q]) \implies z \in \text{fv } Q$
 ⟨proof⟩

lemma *Gen_substs_bd*: $\text{length } xs = \text{length } ys \implies$
 $(\bigwedge xs \text{ } ys. \text{length } xs = \text{length } ys \implies \text{Gen } (\text{subst_var } xs \text{ } ys \ z) (\text{cp } (Qz[xs \rightarrow^* ys]))) \implies$
 $\text{Gen } (\text{substs_bd } z \text{ } xs \text{ } ys \ Qz) (\text{cp } (Qz[\text{substs_src } z \text{ } xs \text{ } ys \ Qz \rightarrow^* \text{substs_dst } z \text{ } xs \text{ } ys \ Qz]))$
 ⟨proof⟩

2.9 Safe-Range Queries

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

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

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

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

lemma *ap_rrb[simp]*: $\text{ap } Q \implies \text{rrb } Q$
 ⟨proof⟩

lemma *qp_rrb[simp]*: $qp\ Q \implies rrb\ Q$
<proof>

lemma *rrb_cp*: $rrb\ Q \implies rrb\ (cp\ Q)$
<proof>

lemma *gen_Gen_erase*: $gen\ x\ Q\ G \implies Gen\ x\ (Q \perp z)$
<proof>

lemma *Gen_erase*: $Gen\ x\ Q \implies Gen\ x\ (Q \perp z)$
<proof>

lemma *rrb_erase*: $rrb\ Q \implies rrb\ (Q \perp x)$
<proof>

lemma *rrb_DISJ[simp]*: $finite\ \mathcal{Q} \implies (\forall Q \in \mathcal{Q}. rrb\ Q) \implies rrb\ (DISJ\ \mathcal{Q})$
<proof>

lemma *rrb_cp_substs*: $rrb\ Q \implies length\ xs = length\ ys \implies rrb\ (cp\ (Q[xs \rightarrow^* ys]))$
<proof>

lemma *rrb_cp_subst*: $rrb\ Q \implies rrb\ (cp\ (Q[x \rightarrow y]))$
<proof>

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

lemma *nongens_cp*: $nongens\ (cp\ Q) \subseteq nongens\ Q$
<proof>

lemma *sr_Disj*: $fv\ Q1 = fv\ Q2 \implies sr\ (Disj\ Q1\ Q2) = (sr\ Q1 \wedge sr\ Q2)$
<proof>

lemma *sr_foldr_Disj*: $\forall Q' \in set\ Qs. fv\ Q' = fv\ Q \implies sr\ (foldr\ Disj\ Qs\ Q) \longleftrightarrow (\forall Q \in set\ Qs. sr\ Q) \wedge sr\ Q$
<proof>

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

lemma *sr_False[simp]*: $sr\ (Bool\ False)$
<proof>

lemma *sr_cp*: $sr\ Q \implies sr\ (cp\ Q)$
<proof>

lemma *sr_DISJ*: $finite\ \mathcal{Q} \implies \forall Q' \in \mathcal{Q}. fv\ Q' = X \implies (\forall Q \in \mathcal{Q}. sr\ Q) \implies sr\ (DISJ\ \mathcal{Q})$
<proof>

lemma *sr_Conj_eq*: $sr\ Q \implies x \in fv\ Q \vee y \in fv\ Q \implies sr\ (Conj\ Q\ (x \approx y))$
<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:  $rrb\ Q \implies rrb (simp\ Q)$ 
and gen_Gen_simp:  $gen\ x\ Q\ G \implies Gen\ x (simp\ Q)$ 
and fv_simp_Disj_same:  $fv (simp\ Q1) = X \implies fv (simp\ Q2) = X \implies fv (simp (Disj\ Q1\ Q2)) =$ 
X
and simp_False:  $simp (Bool\ False) = Bool\ False$ 
and simplified_sub:  $simplified\ Q \implies Q' \in sub\ Q \implies simplified\ Q'$ 
and simplified_Conj_eq:  $simplified\ Q \implies x \neq y \implies x \in fv\ Q \vee y \in fv\ Q \implies simplified (Conj\ Q$ 
( $x \approx y$ ))
and simplified_fv_simp:  $simplified\ Q \implies fv (simp\ Q) = fv\ Q$ 
and simplified_simp:  $simplified (simp\ Q)$ 
and simplified_cp:  $simplified (cp\ Q)$ 
begin

lemma Gen_simp:  $Gen\ x\ Q \implies Gen\ x (simp\ Q)$ 
  <proof>

lemma nongens_simp:  $nongens (simp\ Q) \subseteq nongens\ Q$ 
  <proof>

lemma sr_simp:  $sr\ Q \implies sr (simp\ Q)$ 
  <proof>

lemma equiv_simp_cong:  $Q \triangleq Q' \implies simp\ Q \triangleq simp\ Q'$ 
  <proof>

lemma equiv_simp:  $simp\ Q \triangleq Q$ 
  <proof>

lemma fv_simp_foldr_Disj:  $\forall Q \in set\ Qs \cup \{Q\}. simplified\ Q \wedge fv\ Q = A \implies$ 
 $fv (simp (foldr\ Disj\ Qs\ Q)) = A$ 
  <proof>

lemma fv_simp_foldr1_Disj:  $simp (foldr1\ Disj\ Qs (Bool\ False)) \neq Bool\ False \implies$ 
 $\forall Q \in set\ Qs. simplified\ Q \wedge fv\ Q = A \implies$ 
 $fv (simp (foldr1\ Disj\ Qs (Bool\ False))) = A$ 
  <proof>

lemma fv_simp_DISJ_eq:
   $finite\ Q \implies simp (DISJ\ Q) \neq Bool\ False \implies \forall Q \in Q. simplified\ Q \wedge fv\ Q = A \implies fv (simp (DISJ$ 
 $Q)) = A$ 
  <proof>

end

```

2.11 Covered Variables

inductive *cov* **where**

```

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

```

| *ConjL*: $\text{cov } x \ Q1 \ G \Longrightarrow \text{cp } (Q1 \perp x) = \text{Bool False} \Longrightarrow \text{cov } x \ (\text{Conj } Q1 \ Q2) \ G$
 | *ConjR*: $\text{cov } x \ Q2 \ G \Longrightarrow \text{cp } (Q2 \perp x) = \text{Bool False} \Longrightarrow \text{cov } x \ (\text{Conj } Q1 \ Q2) \ G$
 | *Exists*: $x \neq y \Longrightarrow \text{cov } x \ Q \ G \Longrightarrow x \approx y \notin G \Longrightarrow \text{cov } x \ (\text{Exists } y \ Q) \ (\text{exists } y \ ' \ G)$
 | *Exists_gen*: $x \neq y \Longrightarrow \text{cov } x \ Q \ G \Longrightarrow \text{gen } y \ Q \ Gy \Longrightarrow \text{cov } x \ (\text{Exists } y \ Q) \ ((\text{exists } y \ ' \ (G - \{x \approx y\})) \cup ((\lambda Q. \text{cp } (Q[y \rightarrow x])) \ ' \ Gy))$

inductive *cov'* where

| *Eq_self*: $\text{cov}' \ x \ (x \approx x) \ \{\}$
 | *nonfree*: $x \notin \text{fv } Q \Longrightarrow \text{cov}' \ x \ Q \ \{\}$
 | *EqL*: $x \neq y \Longrightarrow \text{cov}' \ x \ (x \approx y) \ \{x \approx y\}$
 | *EqR*: $x \neq y \Longrightarrow \text{cov}' \ x \ (y \approx x) \ \{x \approx y\}$
 | *ap*: $\text{ap } Q \Longrightarrow x \in \text{fv } Q \Longrightarrow \text{cov}' \ x \ Q \ \{Q\}$
 | *Neg*: $\text{cov}' \ x \ Q \ G \Longrightarrow \text{cov}' \ x \ (\text{Neg } Q) \ G$
 | *Disj*: $\text{cov}' \ x \ Q1 \ G1 \Longrightarrow \text{cov}' \ x \ Q2 \ G2 \Longrightarrow \text{cov}' \ x \ (\text{Disj } Q1 \ Q2) \ (G1 \cup G2)$
 | *DisjL*: $\text{cov}' \ x \ Q1 \ G \Longrightarrow \text{cp } (Q1 \perp x) = \text{Bool True} \Longrightarrow \text{cov}' \ x \ (\text{Disj } Q1 \ Q2) \ G$
 | *DisjR*: $\text{cov}' \ x \ Q2 \ G \Longrightarrow \text{cp } (Q2 \perp x) = \text{Bool True} \Longrightarrow \text{cov}' \ x \ (\text{Disj } Q1 \ Q2) \ G$
 | *Conj*: $\text{cov}' \ x \ Q1 \ G1 \Longrightarrow \text{cov}' \ x \ Q2 \ G2 \Longrightarrow \text{cov}' \ x \ (\text{Conj } Q1 \ Q2) \ (G1 \cup G2)$
 | *ConjL*: $\text{cov}' \ x \ Q1 \ G \Longrightarrow \text{cp } (Q1 \perp x) = \text{Bool False} \Longrightarrow \text{cov}' \ x \ (\text{Conj } Q1 \ Q2) \ G$
 | *ConjR*: $\text{cov}' \ x \ Q2 \ G \Longrightarrow \text{cp } (Q2 \perp x) = \text{Bool False} \Longrightarrow \text{cov}' \ x \ (\text{Conj } Q1 \ Q2) \ G$
 | *Exists*: $x \neq y \Longrightarrow \text{cov}' \ x \ Q \ G \Longrightarrow x \approx y \notin G \Longrightarrow \text{cov}' \ x \ (\text{Exists } y \ Q) \ (\text{exists } y \ ' \ G)$
 | *Exists_gen*: $x \neq y \Longrightarrow \text{cov}' \ x \ Q \ G \Longrightarrow \text{gen } y \ Q \ Gy \Longrightarrow \text{cov}' \ x \ (\text{Exists } y \ Q) \ ((\text{exists } y \ ' \ (G - \{x \approx y\})) \cup ((\lambda Q. \text{Q}[y \rightarrow x]) \ ' \ Gy))$

lemma *cov_nocp_intros*:

$x \neq y \Longrightarrow \text{cov } x \ Q \ G \Longrightarrow \text{gen } y \ Q \ Gy \Longrightarrow \text{cov } x \ (\text{Exists } y \ Q) \ ((\text{exists } y \ ' \ (G - \{x \approx y\})) \cup ((\lambda Q. \text{Q}[y \rightarrow x]) \ ' \ Gy))$
 <proof>

lemma *cov'_cp_intros*:

$x \neq y \Longrightarrow \text{cov}' \ x \ Q \ G \Longrightarrow \text{gen } y \ Q \ Gy \Longrightarrow \text{cov}' \ x \ (\text{Exists } y \ Q) \ ((\text{exists } y \ ' \ (G - \{x \approx y\})) \cup ((\lambda Q. \text{cp } (Q[y \rightarrow x])) \ ' \ Gy))$
 <proof>

lemma *cov'_cov*: $\text{cov}' \ x \ Q \ G \Longrightarrow \text{cov } x \ Q \ G$

<proof>

lemma *cov_cov'*: $\text{cov } x \ Q \ G \Longrightarrow \text{cov}' \ x \ Q \ G$

<proof>

lemma *cov_eq_cov'*: $\text{cov} = \text{cov}'$

<proof>

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

lemma *ex_cov*: $\text{rrb } Q \Longrightarrow x \in \text{fv } Q \Longrightarrow \exists G. \text{cov } x \ Q \ G$

<proof>

definition *qps* where

$\text{qps } G = \{Q \in G. \text{qp } Q\}$

lemma *qps_qp*: $Q \in \text{qps } G \Longrightarrow \text{qp } Q$

<proof>

lemma *qps_in*: $Q \in \text{qps } G \Longrightarrow Q \in G$

<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]*: $qps (X \cup Y) = qps X \cup qps Y$
<proof>

lemma *finite_qps[simp]*: $finite G \implies finite (qps G)$
<proof>

lemma *qps_exists[simp]*: $x \neq y \implies qps (exists y ' G) = exists y ' qps G$
<proof>

lemma *qps_subst[simp]*: $qps ((\lambda Q. Q[x \rightarrow y]) ' G) = (\lambda Q. Q[x \rightarrow y]) ' qps G$
<proof>

lemma *qps_minus[simp]*: $qps (G - \{x \approx y\}) = qps G$
<proof>

lemma *gen_qps[simp]*: $gen x Q G \implies qps G = G$
<proof>

lemma *qps_rrb[simp]*: $Q \in qps G \implies rrb Q$
<proof>

definition *eqs where*
 $eqs x G = \{y. x \neq y \wedge x \approx y \in G\}$

lemma *eqs_in*: $y \in eqs x G \implies x \approx y \in G$
<proof>

lemma *eqs_noteq*: $y \in eqs x Q \implies x \neq y$
<proof>

lemma *eqs_empty[simp]*: $eqs x \{\} = \{\}$
<proof>

lemma *eqs_union[simp]*: $eqs x (X \cup Y) = eqs x X \cup eqs x Y$
<proof>

lemma *finite_eqs[simp]*: $finite G \implies finite (eqs x G)$
<proof>

lemma *eqs_exists[simp]*: $x \neq y \implies eqs x (exists y ' G) = eqs x G - \{y\}$
<proof>

lemma *notin_eqs[simp]*: $x \approx y \notin G \implies y \notin eqs x G$
<proof>

lemma *eqs_minus[simp]*: $eqs x (G - \{x \approx y\}) = eqs x G - \{y\}$
<proof>

lemma *Var_eq_subst_iff*: $Var z = t[x \rightarrow t y] \iff (if z = x then x = y \wedge t = Var x else if z = y then t = Var x \vee t = Var y else t = Var z)$
<proof>

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

lemma *eqs_subst[simp]*: $x \neq y \implies \text{eqs } y ((\lambda Q. Q[x \rightarrow y]) ' G) = (\text{eqs } y G - \{x\}) \cup (\text{eqs } x G - \{y\})$
 ⟨proof⟩

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

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

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

lemma *eqs_code[code]*: $\text{eqs } x G = (\lambda eq. \text{case eq of } y \approx z \Rightarrow z) ' (\text{Set.filter } (\lambda eq. \text{case eq of } y \approx z \Rightarrow x = y \wedge x \neq z \mid _ \Rightarrow \text{False}) G)$
 ⟨proof⟩

lemma *gen_finite[simp]*: $\text{gen } x Q G \implies \text{finite } G$
 ⟨proof⟩

lemma *cov_finite[simp]*: $\text{cov } x Q G \implies \text{finite } G$
 ⟨proof⟩

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

lemma *cov_sat_erase*: $\text{cov } x Q G \implies \text{sat } (\text{Neg } (\text{Disj } (\text{DISJ } (\text{qps } G)) (\text{DISJ } ((\lambda y. x \approx y) ' \text{eqs } x G)))) I \sigma \implies \text{sat } Q I \sigma \leftrightarrow \text{sat } (\text{cp } (Q \perp x)) I \sigma$
 ⟨proof⟩

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

lemma *cov_fv*: $\text{cov } x Q G \implies x \in \text{fv } Q \implies Qqp \in G \implies x \in \text{fv } Qqp \wedge \text{fv } Qqp \subseteq \text{fv } Q$
 ⟨proof⟩

lemma *Gen_Conj*:
 $\text{Gen } x Q1 \implies \text{Gen } x (\text{Conj } Q1 Q2)$
 $\text{Gen } x Q2 \implies \text{Gen } x (\text{Conj } Q1 Q2)$
 ⟨proof⟩

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

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

fun *csts_term* **where**

$csts_term (Var\ x) = \{\}$
 $| csts_term (Const\ c) = \{c\}$

fun *csts* **where**

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

lemma *finite_csts_term[simp]*: $finite\ (csts_term\ t)$
 $\langle proof \rangle$

lemma *finite_csts[simp]*: $finite\ (csts\ t)$
 $\langle proof \rangle$

lemma *ap_fresh_val*: $ap\ Q \implies \sigma\ x \notin adom\ I \implies \sigma\ x \notin csts\ Q \implies sat\ Q\ I\ \sigma \implies x \notin fv\ Q$
 $\langle proof \rangle$

lemma *qp_fresh_val*: $qp\ Q \implies \sigma\ x \notin adom\ I \implies \sigma\ x \notin csts\ Q \implies sat\ Q\ I\ \sigma \implies x \notin fv\ Q$
 $\langle proof \rangle$

lemma *ex_fresh_val*:
fixes $Q :: ('a :: infinite, 'b)\ fmla$
assumes $finite\ (adom\ I)\ finite\ A$
shows $\exists x.\ x \notin adom\ I \wedge x \notin csts\ Q \wedge x \notin A$
 $\langle proof \rangle$

definition *fresh_val* :: $('a :: infinite, 'b)\ fmla \Rightarrow ('a, 'b)\ intp \Rightarrow 'a\ set \Rightarrow 'a$ **where**
 $fresh_val\ Q\ I\ A = (SOME\ x.\ x \notin adom\ I \wedge x \notin csts\ Q \wedge x \notin A)$

lemma *fresh_val*:
 $finite\ (adom\ I) \implies finite\ A \implies fresh_val\ Q\ I\ A \notin adom\ I$
 $finite\ (adom\ I) \implies finite\ A \implies fresh_val\ Q\ I\ A \notin csts\ Q$
 $finite\ (adom\ I) \implies finite\ A \implies fresh_val\ Q\ I\ A \notin A$
 $\langle proof \rangle$

lemma *csts_exists[simp]*: $csts\ (exists\ x\ Q) = csts\ Q$
 $\langle proof \rangle$

lemma *csts_term_subst_term[simp]*: $csts_term\ (t[x \rightarrow y]) = csts_term\ t$
 $\langle proof \rangle$

lemma *csts_subst[simp]*: $csts\ (Q[x \rightarrow y]) = csts\ Q$
 $\langle proof \rangle$

lemma *gen_csts*: $gen\ x\ Q\ G \implies Qqp \in G \implies csts\ Qqp \subseteq csts\ Q$
 $\langle proof \rangle$

lemma *cov_csts*: $cov\ x\ Q\ G \implies Qqp \in G \implies csts\ Qqp \subseteq csts\ Q$
 $\langle proof \rangle$

lemma *not_self_eqs[simp]*: $x \notin eqs\ x\ G$
 $\langle proof \rangle$

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

fixes $Q :: ('a :: \{\text{infinite}, \text{linorder}\}, 'b :: \text{linorder}) \text{ fmla}$
assumes $\text{cov } x \ Q \ G \ x \in \text{fv } Q$
shows $\text{Exists } x \ Q \triangleq \text{Disj } (\text{Exists } x \ (\text{simp } (\text{Conj } Q \ (\text{DISJ } (\text{qps } G))))$
 $(\text{Disj } (\text{DISJ } ((\lambda y. \text{cp } (Q[x \rightarrow y])) \text{ 'eqs } x \ G)) \ (\text{cp } (Q \perp x)))$
 $\langle \text{proof} \rangle$

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

definition $\text{eval } Q \ I = \text{eval_on } (\text{fv } Q) \ Q \ I$

lemmas $\text{eval_deep_def} = \text{eval_def}[\text{unfolded } \text{eval_on_def}]$

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

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

lemma equiv_eval_eqI : $\text{finite } (\text{adom } I) \implies \text{fv } Q = \text{fv } Q' \implies Q \triangleq Q' \implies \text{eval } Q \ I = \text{eval } Q' \ I$
 $\langle \text{proof} \rangle$

lemma equiv_eval_on_eqI : $\text{finite } (\text{adom } I) \implies Q \triangleq Q' \implies \text{eval_on } X \ Q \ I = \text{eval_on } X \ Q' \ I$
 $\langle \text{proof} \rangle$

lemma $\text{equiv_eval_on_eval_eqI}$: $\text{finite } (\text{adom } I) \implies \text{fv } Q \subseteq \text{fv } Q' \implies Q \triangleq Q' \implies \text{eval_on } (\text{fv } Q') \ Q$
 $I = \text{eval } Q' \ I$
 $\langle \text{proof} \rangle$

lemma $\text{finite_eval_on_Disj2D}$:
assumes $\text{finite } X$
shows $\text{finite } (\text{eval_on } X \ (\text{Disj } Q1 \ Q2) \ I) \implies \text{finite } (\text{eval_on } X \ Q2 \ I)$
 $\langle \text{proof} \rangle$

lemma $\text{finite_eval_Disj2D}$: $\text{finite } (\text{eval } (\text{Disj } Q1 \ Q2) \ I) \implies \text{finite } (\text{eval } Q2 \ I)$
 $\langle \text{proof} \rangle$

lemma $\text{infinite_eval_Disj2}$:
fixes $Q1 \ Q2 :: ('a :: \{\text{infinite}, \text{linorder}\}, 'b :: \text{linorder}) \text{ fmla}$
assumes $\text{fv } Q2 \subseteq \text{fv } (\text{Disj } Q1 \ Q2) \ \text{sat } Q2 \ I \ \sigma$
shows $\text{infinite } (\text{eval } (\text{Disj } Q1 \ Q2) \ I)$
 $\langle \text{proof} \rangle$

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

<proof>

lemma *cov_eval_inf*:

fixes $Q :: ('a :: \{infinite, linorder\}, 'b :: linorder) \text{ fmla}$
assumes $\text{cov } x \ Q \ G \ x \in \text{fv } Q \ \text{finite} \ (\text{adom } I) \ \text{sat} \ (Q \perp x) \ I \ \sigma$
shows $\text{infinite} \ (\text{eval } Q \ I)$

<proof>

2.12 More on Evaluation

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

<proof>

lemma *eval_on_Bool_False[simp]*: $\text{eval_on } X \ (\text{Bool } \text{False}) \ I = \{\}$

<proof>

lemma *eval_DISJ_prune_unsat*: $\text{finite } B \implies A \subseteq B \implies \forall Q \in B - A. \forall \sigma. \neg \text{sat } Q \ I \ \sigma \implies \text{eval_on } X \ (\text{DISJ } A) \ I = \text{eval_on } X \ (\text{DISJ } B) \ I$

<proof>

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

<proof>

lemma *eval_cp_DISJ_closed*: $\text{finite } \mathcal{Q} \implies \forall Q \in \mathcal{Q}. \text{fv } Q = \{\} \implies \text{eval} \ (\text{cp} \ (\text{DISJ } \mathcal{Q})) \ I = (\bigcup Q \in \mathcal{Q}. \text{eval } Q \ I)$

<proof>

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

<proof>

lemma *eval_cong*: $\text{fv } Q = \text{fv } Q' \implies (\bigwedge \sigma. \text{sat } Q \ I \ \sigma = \text{sat } Q' \ I \ \sigma) \implies \text{eval } Q \ I = \text{eval } Q' \ I$

<proof>

lemma *eval_on_cong*: $(\bigwedge \sigma. \text{sat } Q \ I \ \sigma = \text{sat } Q' \ I \ \sigma) \implies \text{eval_on } X \ Q \ I = \text{eval_on } X \ Q' \ I$

<proof>

lemma *eval_empty_alt*: $\text{eval } Q \ I = \{\} \iff (\forall \sigma. \neg \text{sat } Q \ I \ \sigma)$

<proof>

lemma *sat_EXISTS*: $\text{distinct } xs \implies \text{sat} \ (\text{EXISTS } xs \ Q) \ I \ \sigma = (\exists ds. \text{length } ds = \text{length } xs \wedge \text{sat } Q \ I \ (\sigma[xs :=^* ds]))$

<proof>

lemma *eval_empty_close*: $\text{eval} \ (\text{close } Q) \ I = \{\} \iff (\forall \sigma. \neg \text{sat } Q \ I \ \sigma)$

<proof>

lemma *infinite_eval_on_extra_variables*:

assumes $\text{finite } X \ \text{fv} \ (Q :: ('a :: infinite, 'b) \text{ fmla}) \subset X \ \exists \sigma. \text{sat } Q \ I \ \sigma$
shows $\text{infinite} \ (\text{eval_on } X \ Q \ I)$

<proof>

lemma *eval_on_cp*: $\text{eval_on } X \ (\text{cp } Q) = \text{eval_on } X \ Q$

<proof>

lemma (*in simplification*) *eval_on_simp*: $\text{eval_on } X \ (\text{simp } Q) = \text{eval_on } X \ Q$

<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*: $ds \in eval Q I \implies (\bigwedge \sigma. length ds = card (fv Q) \implies sat Q I (\sigma[sorted_list_of_set (fv Q) :=^* ds]) \implies R) \implies R$
 ⟨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*: $infinite (eval_on X Q I) \implies finite X \implies (\bigwedge \sigma. sat (Impl Q Q') I \sigma) \implies infinite (eval_on X Q' I)$
 ⟨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*: $Q' \in flat_Disj Q \implies x \in fv Q' \implies x \in fv Q$
 ⟨proof⟩

lemma *cpropagated_flat_DisjD*: $Q' \in flat_Disj Q \implies cpropagated Q \implies cpropagated Q'$
 ⟨proof⟩

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

lemma (in *simplification*) *simplified_flat_DisjD*: $Q' \in flat_Disj Q \implies simplified Q \implies simplified Q'$
 ⟨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*:

$finite \mathcal{Q} \implies \exists x Q \triangleq DISJ (\exists x ' \mathcal{Q}) \implies (\bigwedge Q'. Q' \in \mathcal{Q} \implies rrb Q') \implies$
 $(\bigwedge Q'. Q' \in \mathcal{Q} \implies fv Q' \subseteq fv Q) \implies (\bigwedge Q'. Q' \in \mathcal{Q} \implies simplified Q') \implies rb_INV x Q \mathcal{Q}$
 <proof>

fun (in simplification) rb :: ('a :: {infinite, linorder}, 'b :: linorder) fmla \Rightarrow ('a, 'b) fmla nres **where**
 rb (Neg Q) = do { Q' \leftarrow rb Q; RETURN (simp (Neg Q')) }
 | rb (Disj Q1 Q2) = do { Q1' \leftarrow rb Q1; Q2' \leftarrow rb Q2; RETURN (simp (Disj Q1' Q2')) }
 | rb (Conj Q1 Q2) = do { Q1' \leftarrow rb Q1; Q2' \leftarrow rb Q2; RETURN (simp (Conj Q1' Q2')) }
 | rb (Exists x Q) = do {
 Q' \leftarrow rb Q;
 Q \leftarrow WHILE_T rb_INV x Q'
 ($\lambda \mathcal{Q}. fixbound \mathcal{Q} x \neq \{\}$) ($\lambda \mathcal{Q}. do$ {
 Qfix \leftarrow RES (fixbound Q x);
 G \leftarrow SPEC (cov x Qfix);
 RETURN (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 ' Q))) }
 | rb Q = do { RETURN (simp Q) }

lemma (in simplification) cov_fixbound: cov x Q G $\implies x \in fv Q \implies$
 $fixbound (insert (cp (Q \perp x)) (insert (simp (Conj Q (DISJ (qps G))))$
 $(Q - \{Q\} \cup ((\lambda y. cp (Q[x \rightarrow y])) ' eqs x G)))) x = fixbound Q x - \{Q\}$
 <proof>

lemma finite_fixbound[simp]: finite Q $\implies finite (fixbound Q x)$
 <proof>

lemma fixboundE[elim_format]: Q $\in fixbound Q x \implies x \in fv Q \wedge Q \in \mathcal{Q} \wedge \neg Gen x Q$
 <proof>

lemma fixbound_fv: Q $\in fixbound Q x \implies x \in fv Q$
 <proof>

lemma fixbound_in: Q $\in fixbound Q x \implies Q \in \mathcal{Q}$
 <proof>

lemma fixbound_empty_Gen: fixbound Q x = {} $\implies x \in fv Q \implies Q \in \mathcal{Q} \implies Gen x Q$
 <proof>

lemma fixbound_insert:
 $fixbound (insert Q \mathcal{Q}) x = (if Gen x Q \vee x \notin fv Q then fixbound Q x else insert Q (fixbound Q x))$
 <proof>

lemma fixbound_empty[simp]:
 $fixbound \{\} x = \{\}$
 <proof>

lemma flat_Disj_Exists_sub: Q' $\in flat_Disj Q \implies \exists y Qy \in sub Q' \implies \exists y Qy \in sub Q$
 <proof>

lemma rrb_flat_Disj[simp]: Q $\in flat_Disj Q' \implies rrb Q' \implies rrb Q$
 <proof>

lemma (in simplification) rb_INV_finite[simp]: rb_INV x Q Q $\implies finite Q$
 <proof>

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

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

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

lemma (in *simplification*) *rb_INV_equiv*: $rb_INV\ x\ Q\ Q \implies \exists x\ Q \triangleq DISJ\ (\exists x\ 'Q)$
 ⟨proof⟩

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

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

lemma (in *simplification*) *rb_correct*:
 fixes $Q :: ('a :: \{linorder, infinite\}, 'b :: linorder)\ fmla$
 shows $rb\ Q \leq rb_spec\ Q$
 ⟨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 ≈ z)) = (if x = y then List.union (*gen_impl* x Q1) (map (image (λQ. cp (Q[z → x]))) (*gen_impl* z Q1))
 else if x = z then List.union (*gen_impl* x Q1) (map (image (λQ. cp (Q[y → 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 $gen_impl_gen: G \in set\ (gen_impl\ x\ Q) \implies gen\ x\ Q\ G$
 ⟨proof⟩

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

$cov_impl\ x\ (Bool\ b) = [\{\}]$
 $| cov_impl\ x\ (Eq\ y\ t) = (case\ t\ of$
 $Const\ c \Rightarrow [if\ x = y\ then\ \{Eq\ y\ (Const\ c)\}\ else\ \{\}]$
 $| Var\ z \Rightarrow [if\ x = y \wedge x \neq z\ then\ \{x \approx z\}$
 $\ else\ if\ x = z \wedge x \neq y\ then\ \{x \approx y\}$
 $\ else\ \{\}])$
 $| cov_impl\ x\ (Pred\ p\ ts) = [if\ x \in fv_terms_set\ ts\ then\ \{Pred\ p\ ts\}\ else\ \{\}]$
 $| cov_impl\ x\ (Neg\ Q) = cov_impl\ x\ Q$
 $| cov_impl\ x\ (Disj\ Q1\ Q2) = (case\ (cp\ (Q1 \perp x),\ cp\ (Q2 \perp x))\ of$
 $(Bool\ True,\ Bool\ True) \Rightarrow List.union\ (cov_impl\ x\ Q1)\ (cov_impl\ x\ Q2)$
 $| (Bool\ True,\ _) \Rightarrow cov_impl\ x\ Q1$
 $| (_,\ Bool\ True) \Rightarrow cov_impl\ x\ Q2$
 $| (_,\ _) \Rightarrow [G1 \cup G2.\ G1 \leftarrow cov_impl\ x\ Q1,\ G2 \leftarrow cov_impl\ x\ Q2])$
 $| cov_impl\ x\ (Conj\ Q1\ Q2) = (case\ (cp\ (Q1 \perp x),\ cp\ (Q2 \perp x))\ of$
 $(Bool\ False,\ Bool\ False) \Rightarrow List.union\ (cov_impl\ x\ Q1)\ (cov_impl\ x\ Q2)$
 $| (Bool\ False,\ _) \Rightarrow cov_impl\ x\ Q1$
 $| (_,\ Bool\ False) \Rightarrow cov_impl\ x\ Q2$
 $| (_,\ _) \Rightarrow [G1 \cup G2.\ G1 \leftarrow cov_impl\ x\ Q1,\ G2 \leftarrow cov_impl\ x\ Q2])$
 $| cov_impl\ x\ (Exists\ y\ Q) = (if\ x = y\ then\ [\{\}]\ else\ flat\ (map\ (\lambda G.$
 $(if\ x \approx y \in G\ then\ [exists\ y\ ' (G - \{x \approx y\}) \cup (\lambda Q.\ cp\ (Q[y \rightarrow x]))\ ' G'.\ G' \leftarrow gen_impl\ y\ Q]$
 $\ else\ [exists\ y\ ' G]))\ (cov_impl\ x\ Q)))$

lemma $union_empty_iff: List.union\ xs\ ys = [] \longleftrightarrow xs = [] \wedge ys = []$
 ⟨proof⟩

lemma $fold_union_empty_iff: fold\ List.union\ xss\ ys = [] \longleftrightarrow (\forall xs \in set\ xss.\ xs = []) \wedge ys = []$
 ⟨proof⟩

lemma $flat_empty_iff: flat\ xss = [] \longleftrightarrow (\forall xs \in set\ xss.\ xs = [])$
 ⟨proof⟩

lemma $set_fold_union: set\ (fold\ List.union\ xss\ ys) = (\bigcup (set\ ' set\ xss)) \cup set\ ys$
 ⟨proof⟩

lemma $set_flat: set\ (flat\ xss) = \bigcup (set\ ' set\ xss)$
 ⟨proof⟩

lemma $rrb_cov_impl: rrb\ Q \implies cov_impl\ x\ Q \neq []$
 ⟨proof⟩

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

lemma *cov_impl_cov*: $G \in \text{set } (\text{cov_impl } x Q) \implies \text{cov } x Q G$
 ⟨proof⟩

definition *fixbound_impl* $\mathcal{Q} x = \text{filter } (\lambda Q. x \in \text{fv } Q \wedge \text{gen_impl } x Q = []) (\text{sorted_list_of_set } \mathcal{Q})$

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

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

lemma *fixbound_impl_hd_in*: $\text{finite } \mathcal{Q} \implies \text{fixbound_impl } \mathcal{Q} x = y \# ys \implies y \in \mathcal{Q}$
 ⟨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;
 Q \leftarrow WHILE
 ($\lambda \mathcal{Q}. \text{fixbound_impl } \mathcal{Q} x \neq []$) ($\lambda \mathcal{Q}. \text{do } \{$
 Qfix \leftarrow RETURN (hd (fixbound_impl Q x));
 G \leftarrow RETURN (hd (cov_impl x Qfix));
 RETURN (Q - {Qfix} \cup
 {simp (Conj Qfix (DISJ (qps G)))}) \cup
 ($\bigcup y \in \text{eqs } x G. \{cp (Qfix[x \rightarrow y])\}$) \cup
 {cp (Qfix \perp x)}))
 (flat_Disj Q');
 RETURN (simp (DISJ (exists x ' Q))) }
 | *rb_impl* Q = do { RETURN (simp Q) }

lemma (in *simplification*) *rb_impl_refines_rb*: $\text{rb_impl } Q \leq \text{rb } Q$
 ⟨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;
 Q \leftarrow dWHILE
 ($\lambda \mathcal{Q}. \text{fixbound_impl } \mathcal{Q} x \neq []$) ($\lambda \mathcal{Q}. \text{do } \{$
 Qfix \leftarrow dRETURN (hd (fixbound_impl Q x));
 G \leftarrow dRETURN (hd (cov_impl x Qfix));
 dRETURN (Q - {Qfix} \cup
 {simp (Conj Qfix (DISJ (qps G)))}) \cup
 ($\bigcup y \in \text{eqs } x G. \{cp (Qfix[x \rightarrow y])\}$) \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*) $rb_impl_det_refines_rb_impl$: $nres_of (rb_impl_det Q) \leq rb_impl Q$
 $\langle proof \rangle$

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 $fixfree :: (('a, 'b) fmla \times nat\ rel) set \Rightarrow (('a, 'b) fmla \times nat\ rel) set$ **where**
 $fixfree\ Qfin = \{(Qfix, Qeq) \in Qfin. nongens\ Qfix \neq \{\}\}$

definition $disjointvars\ Q\ Qeq = (\bigcup V \in classes\ Qeq. if\ V \cap fv\ Q = \{\} then\ V\ else\ \{\})$

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

function (sequential) $Conjs_disjoint$ **where**
 $Conjs_disjoint\ Q\ xys = (case\ find\ (\lambda(x,y). \{x, y\} \cap fv\ Q \neq \{\})\ xys\ of$
 $None \Rightarrow Conjs\ Q\ xys$
 $| Some\ (x, y) \Rightarrow Conjs_disjoint (Conj\ Q\ (x \approx y)) (remove1\ (x, y)\ xys)$
 $\langle proof \rangle$

termination
 $\langle proof \rangle$

declare $Conjs_disjoint.simps[simp\ del]$

definition $CONJ$ **where**
 $CONJ = (\lambda(Q, Qeq). Conjs\ Q (sorted_list_of_set\ Qeq))$

definition $CONJ_disjoint$ **where**
 $CONJ_disjoint = (\lambda(Q, Qeq). Conjs_disjoint\ Q (sorted_list_of_set\ Qeq))$

definition inf **where**
 $inf\ Qfin\ Q = \{(Q', Qeq) \in Qfin. disjointvars\ Q'\ Qeq \neq \{\} \vee fv\ Q' \cup Field\ Qeq \neq fv\ Q\}$

definition FV **where**
 $FV\ Q\ Qfin\ Qinf \equiv (fv\ Qfin = fv\ Q \vee Qfin = Bool\ False) \wedge fv\ Qinf = \{\}$

definition $EVAL$ **where**
 $EVAL\ Q\ Qfin\ Qinf \equiv (\forall I. finite (adom\ I) \longrightarrow (if\ eval\ Qinf\ I = \{\} then$
 $eval\ Qfin\ I = eval\ Q\ I else\ infinite (eval\ Q\ I)))$

definition $EVAL'$ **where**
 $EVAL'\ Q\ Qfin\ Qinf \equiv (\forall I. finite (adom\ I) \longrightarrow (if\ eval\ Qinf\ I = \{\} then$
 $eval_on (fv\ Q)\ Qfin\ I = eval\ Q\ I else\ infinite (eval\ Q\ I)))$

definition (in *simplification*) $split_spec :: ('a :: \{infinite, linorder\}, 'b :: linorder) fmla \Rightarrow (('a, 'b) fmla$
 $\times ('a, 'b) fmla) nres$ **where**
 $split_spec\ Q = SPEC (\lambda(Qfin, Qinf). sr\ Qfin \wedge sr\ Qinf \wedge FV\ Q\ Qfin\ Qinf \wedge EVAL\ Q\ Qfin\ Qinf \wedge$
 $simplified\ Qfin \wedge simplified\ Qinf)$

definition (in *simplification*) $assemble = (\lambda(Qfin, Qinf). (simp (DISJ (CONJ_disjoint \text{' } Qfin)), simp (DISJ (close \text{' } Qinf))))$

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*) $wf_state\ Q\ P =$

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

definition (in *simplification*) $split_INV1\ Q = (\lambda Qpair. wf_state\ Q\ rrb\ Qpair \wedge (let\ (Qfin, Qinf) = assemble\ Qpair\ in\ EVAL'\ Q\ Qfin\ Qinf))$

definition (in *simplification*) $split_INV2\ Q = (\lambda Qpair. wf_state\ Q\ sr\ Qpair \wedge (let\ (Qfin, Qinf) = assemble\ Qpair\ in\ EVAL'\ Q\ Qfin\ Qinf))$

definition (in *simplification*) $split :: ('a :: \{infinite, linorder\}, 'b :: linorder)\ fmla \Rightarrow (('a, 'b)\ fmla \times ('a, 'b)\ fmla)\ nres\ \mathbf{where}$

$split\ Q = do\ \{$
 $Q' \leftarrow rb\ Q;$
 $Qpair \leftarrow WHILE_T\ split_INV1\ Q$
 $(\lambda(Qfin, _). fixfree\ Qfin \neq\ \{\}) (\lambda(Qfin, Qinf). do\ \{$
 $(Qfix, Qeq) \leftarrow RES\ (fixfree\ Qfin);$
 $x \leftarrow RES\ (nongens\ Qfix);$
 $G \leftarrow SPEC\ (cov\ x\ Qfix);$
 $let\ Qfin = Qfin - \{(Qfix, Qeq)\} \cup$
 $\{(simp\ (Conj\ Qfix\ (DISJ\ (qps\ G))), Qeq)\} \cup$
 $(\bigcup y \in\ eqs\ x\ G. \{(cp\ (Qfix[x \rightarrow y]), Qeq \cup \{(x,y)\}\});$
 $let\ Qinf = Qinf \cup \{cp\ (Qfix \perp\ x)\};$
 $RETURN\ (Qfin, Qinf)\}$
 $\{(Q', \{\}), \{\});$
 $Qpair \leftarrow WHILE_T\ split_INV2\ Q$
 $(\lambda(Qfin, _). inf\ Qfin\ Q \neq\ \{\}) (\lambda(Qfin, Qinf). do\ \{$
 $Qpair \leftarrow RES\ (inf\ Qfin\ Q);$
 $let\ Qfin = Qfin - \{Qpair\};$
 $let\ Qinf = Qinf \cup \{CONJ\ Qpair\};$
 $RETURN\ (Qfin, Qinf)\}$
 $Qpair;$
 $let\ (Qfin, Qinf) = assemble\ Qpair;$
 $Qinf \leftarrow rb\ Qinf;$
 $RETURN\ (Qfin, Qinf)\}$

lemma *finite_fixfree[simp]*: $finite\ Q \Rightarrow finite\ (fixfree\ Q)$
 $\langle proof \rangle$

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 $((nongens \circ fst) \text{' } \# mset_set\ (insert\ (simp\ (Conj\ Qfin\ (DISJ\ (qps\ G))), Qeq)\ (Qfin - \{(Qfin, Qeq)\} \cup (\lambda y. (cp\ (Qfin[x \rightarrow y]), insert\ (x, y)\ Qeq)) \text{' } eqs\ x\ G)),$
 $(nongens \circ fst) \text{' } \# mset_set\ Qfin) \in\ mult\ \{(X, Y). X \subset Y \wedge Y \subseteq F\}$
(is $(?f\ (insert\ ?Q\ (?A \cup\ ?B)), ?C) \in\ mult\ ?R)$
 $\langle proof \rangle$

lemma *EVAL_cong*:

$Qinf \triangleq Qinf' \Rightarrow fv\ Qinf = fv\ Qinf' \Rightarrow EVAL\ Q\ Qfin\ Qinf = EVAL\ Q\ Qfin\ Qinf'$
 $\langle proof \rangle$

lemma *EVAL'_cong*:

$$Qinf \triangleq Qinf' \implies fv\ Qinf = fv\ Qinf' \implies EVAL'\ Q\ Qfin\ Qinf = EVAL'\ Q\ Qfin\ Qinf'$$

<proof>

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

lemma *fv_Conjs_disjoint[simp]*: $distinct\ xys \implies fv\ (Conjs_disjoint\ Q\ xys) = fv\ Q \cup Field\ (set\ xys)$
<proof>

lemma *fv_CONJ[simp]*: $finite\ Qeq \implies fv\ (CONJ\ (Q,\ Qeq)) = fv\ Q \cup Field\ Qeq$
<proof>

lemma *fv_CONJ_disjoint[simp]*: $finite\ Qeq \implies fv\ (CONJ_disjoint\ (Q,\ Qeq)) = fv\ Q \cup Field\ Qeq$
<proof>

lemma *rrb_Conjs*: $rrb\ Q \implies rrb\ (Conjs\ Q\ xys)$
<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]*: $irrefl\ (set\ xys) \implies Conjs\ Q\ xys = Bool\ False \longleftrightarrow Q = Bool\ False \wedge xys = []$
<proof>

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

lemma *Conjs_disjoint_eq_False_iff[simp]*: $irrefl\ (set\ xys) \implies Conjs_disjoint\ Q\ xys = Bool\ False \longleftrightarrow Q = Bool\ False \wedge xys = []$
<proof>

lemma *CONJ_disjoint_eq_False_iff[simp]*: $finite\ Qeq \implies irrefl\ Qeq \implies CONJ_disjoint\ (Q,\ Qeq) = Bool\ False \longleftrightarrow Q = Bool\ False \wedge Qeq = \{\}$
<proof>

lemma *sr_Conjs_disjoint*:

$$distinct\ xys \implies (\forall V \in classes\ (set\ xys). V \cap fv\ Q \neq \{\}) \implies sr\ Q \implies sr\ (Conjs_disjoint\ Q\ xys)$$

<proof>

lemma *sr_CONJ_disjoint*:

$$inf\ Qfin\ Q = \{\} \implies (Qfin,\ Qeq) \in Qfin \implies finite\ Qeq \implies sr\ Qfin \implies sr\ (CONJ_disjoint\ (Qfin,\ Qeq))$$

<proof>

lemma *equiv_Conjs_cong*: $Q \triangleq Q' \implies Conjs\ Q\ xys \triangleq Conjs\ Q'\ xys$
<proof>

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

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

<proof>

lemma *ex_Conjs_disjoint_eq_Conjs:*

distinct xys $\implies \exists xys'. \text{distinct } xys' \wedge \text{set } xys = \text{set } xys' \wedge \text{Conjs_disjoint } Q \text{ } xys = \text{Conjs } Q \text{ } xys'$

<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) \implies leftfresh Q xys \implies infinite (eval (Conjs Q xys) I)*

<proof>

lemma *leftfresh_fv_subset: leftfresh Q xys \implies fv Q' \subseteq fv Q \implies leftfresh Q' xys*

<proof>

lemma *fun_upds_map: ($\forall x. x \notin \text{set } ys \longrightarrow \sigma x = \tau x$) \implies $\sigma[\text{ys} :=^* \text{map } \tau \text{ } \text{ys}] = \tau$*

<proof>

lemma *map_fun_upds: length xs = length ys \implies distinct xs \implies map ($\sigma[\text{xs} :=^* \text{ys}]$) xs = ys*

<proof>

lemma *zip_map: zip xs (map f xs) = map ($\lambda x. (x, f x)$) xs*

<proof>

lemma *filter_sorted_list_of_set:*

finite B \implies A \subseteq B \implies filter ($\lambda x. x \in A$) (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) \implies 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 $\sigma \longleftrightarrow$ sat Q I $\sigma \wedge (\forall (x, y) \in \text{set } xys. \text{sat } (x \approx y) \text{ } I \text{ } \sigma)$*

<proof>

lemma *sat_Conjs_disjoint: sat (Conjs_disjoint Q xys) I $\sigma \longleftrightarrow$ sat Q I $\sigma \wedge (\forall (x, y) \in \text{set } xys. \text{sat } (x \approx y) \text{ } I \text{ } \sigma)$*

<proof>

lemma *sat_CONJ: finite Qeq \implies sat (CONJ (Q, Qeq)) I $\sigma \longleftrightarrow$ sat Q I $\sigma \wedge (\forall (x, y) \in \text{Qeq}. \text{sat } (x \approx y) \text{ } I \text{ } \sigma)$*

<proof>

lemma *sat_CONJ_disjoint: finite Qeq \implies sat (CONJ_disjoint (Q, Qeq)) I $\sigma \longleftrightarrow$ sat Q I $\sigma \wedge (\forall (x, y) \in \text{Qeq}. \text{sat } (x \approx y) \text{ } I \text{ } \sigma)$*

<proof>

lemma *Conjs_inject: Conjs Q xys = Conjs Q' xys \longleftrightarrow Q = Q'*

<proof>

lemma *nonempty_disjointvars_infinite*:

assumes *disjointvars* ($Q_{fin} :: ('a :: infinite, 'b) fmla$) $Q_{eq} \neq \{\}$
finite $Q_{eq} \text{ fv } Q_{fin} \cup \text{Field } Q_{eq} \subseteq X$ *finite* $X \text{ sat } Q_{fin} I \sigma \forall (x, y) \in Q_{eq}. \sigma x = \sigma y$
shows *infinite* (*eval_on* X (*CONJ_disjoint* (Q_{fin} , Q_{eq})) I)
 <proof>

lemma *EVAL'_EVAL*: $EVAL' Q Q_{fin} Q_{inf} \implies FV Q Q_{fin} Q_{inf} \implies EVAL Q Q_{fin} Q_{inf}$
 <proof>

lemma *cpropagated_Conjs_disjoint*:

distinct xys $\implies \text{irrefl } (set \ xys) \implies \forall V \in \text{classes } (set \ xys). V \cap \text{fv } Q \neq \{\} \implies \text{cpropagated } Q \implies$
cpropagated (*Conjs_disjoint* $Q \ xys$)
 <proof>

lemma (in *simplification*) *simplified_Conjs_disjoint*:

distinct xys $\implies \text{irrefl } (set \ xys) \implies \forall V \in \text{classes } (set \ xys). V \cap \text{fv } Q \neq \{\} \implies \text{simplified } Q \implies \text{simplified}$
 (*Conjs_disjoint* $Q \ xys$)
 <proof>

lemma *disjointvars_empty_iff*: $\text{disjointvars } Q \ Q_{eq} = \{\} \longleftrightarrow (\forall V \in \text{classes } Q_{eq}. V \cap \text{fv } Q \neq \{\})$
 <proof>

lemma *cpropagated_CONJ_disjoint*:

finite $Q_{eq} \implies \text{irrefl } Q_{eq} \implies \text{disjointvars } Q \ Q_{eq} = \{\} \implies \text{cpropagated } Q \implies \text{cpropagated } (CONJ_disjoint$
 (Q, Q_{eq}))
 <proof>

lemma (in *simplification*) *simplified_CONJ_disjoint*:

finite $Q_{eq} \implies \text{irrefl } Q_{eq} \implies \text{disjointvars } Q \ Q_{eq} = \{\} \implies \text{simplified } Q \implies \text{simplified } (CONJ_disjoint$
 (Q, Q_{eq}))
 <proof>

lemma (in *simplification*) *split_INV1_init*:

$\text{rrb } Q' \implies \text{simplified } Q' \implies Q \triangleq Q' \implies \text{fv } Q' \subseteq \text{fv } Q \implies \text{split_INV1 } Q (\{(Q', \{\})\}, \{\})$
 <proof>

lemma (in *simplification*) *split_INV1_I*:

wf_state $Q \ \text{rrb } (Q_{fin}, Q_{inf}) \implies EVAL' Q (\text{simp } (DISJ (CONJ_disjoint ' Q_{fin}))) (\text{simp } (DISJ (close$
 ' $Q_{inf}))) \implies$
split_INV1 $Q (Q_{fin}, Q_{inf})$
 <proof>

lemma *EVAL'_I*:

$(\bigwedge I. \text{finite } (adom \ I) \implies \text{eval } Q_{inf} \ I = \{\}) \implies \text{eval_on } (fv \ Q) \ Q_{fin} \ I = \text{eval } Q \ I \implies$
 $(\bigwedge I. \text{finite } (adom \ I) \implies \text{eval } Q_{inf} \ I \neq \{\}) \implies \text{infinite } (\text{eval } Q \ I) \implies EVAL' Q Q_{fin} Q_{inf}$
 <proof>

lemma (in *simplification*) *wf_state_Un*:

wf_state $Q \ P (Q_{fin}, Q_{inf}) \implies \text{wf_state } Q \ P (\text{insert } Q_{pair} \ Q_{new}, \{Q'\}) \implies$
wf_state $Q \ P (\text{insert } Q_{pair} (Q_{fin} \cup Q_{new}), \text{insert } Q' \ Q_{inf})$
 <proof>

lemma (in *simplification*) *wf_state_Diff*:

wf_state $Q \ P (Q_{fin}, Q_{inf}) \implies \text{wf_state } Q \ P (Q_{fin} - Q_{new}, Q_{inf})$
 <proof>

lemma (in *simplification*) *split_INV1_step*:

assumes *split_INV1* Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$) ($Qfin$, Qeq) \in *fixfree* $\mathcal{Q}fin$ $x \in$ *nongens* $\mathcal{Q}fin$ *cov* x $\mathcal{Q}fin$ G
shows *split_INV1* Q
(insert (simp (Conj Qfin (DISJ (qps G))), Qeq)
($\mathcal{Q}fin - \{(Qfin, Qeq)\} \cup (\lambda y. (cp (Qfin[x \rightarrow y]), insert (x, y) Qeq))$ ‘eqs x G),
insert (cp ($\mathcal{Q}fin \perp x$) $\mathcal{Q}inf$)
*(is *split_INV1* Q (? $Qfin$, ? $Qinf$))*
<proof>

lemma (in simplification) *split_INV1_decreases*:
assumes *split_INV1* Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$) ($Qfin$, Qeq) \in *fixfree* $\mathcal{Q}fin$ $x \in$ *nongens* $\mathcal{Q}fin$ *cov* x $\mathcal{Q}fin$ G
shows $((nongens \circ fst) \# mset_set (insert (simp (Conj Qfin (DISJ (qps G))), Qeq) (\mathcal{Q}fin - \{(Qfin, Qeq)\} \cup (\lambda y. (cp (Qfin[x \rightarrow y]), insert (x, y) Qeq))$ ‘eqs x G),
(nongens \circ fst) # mset_set $\mathcal{Q}fin$) \in mult $\{(X, Y). X \subset Y \wedge Y \subseteq fv Q\}$
<proof>

lemma (in simplification) *split_INV2_init*:
split_INV1 Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$) \implies *fixfree* $\mathcal{Q}fin = \{\}$ \implies *split_INV2* Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$)
<proof>

lemma (in simplification) *split_INV2_I*:
wf_state Q *sr* ($\mathcal{Q}fin$, $\mathcal{Q}inf$) \implies *EVAL'* Q (*simp* (*DISJ* (*CONJ_disjoint* ‘ $\mathcal{Q}fin$))) (*simp* (*DISJ* (*close* ‘ $\mathcal{Q}inf$))) \implies
split_INV2 Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$)
<proof>

lemma (in simplification) *split_INV2_step*:
assumes *split_INV2* Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$) ($Qfin$, Qeq) \in *inf* $\mathcal{Q}fin$ Q
shows *split_INV2* Q ($\mathcal{Q}fin - \{(Qfin, Qeq)\}$, *insert* (*CONJ* ($Qfin$, Qeq)) $\mathcal{Q}inf$)
<proof>

lemma (in simplification) *split_INV2_decreases*:
split_INV2 Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$) \implies ($Qfin$, Qeq) \in *Restrict_Frees.inf* $\mathcal{Q}fin$ $Q \implies$ *card* ($\mathcal{Q}fin - \{(Qfin, Qeq)\}$) $<$ *card* $\mathcal{Q}fin$
<proof>

lemma (in simplification) *split_INV2_stop_fin_sr*:
inf $\mathcal{Q}fin$ $Q = \{\}$ \implies *split_INV2* Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$) \implies *assemble* ($\mathcal{Q}fin$, $\mathcal{Q}inf$) = ($Qfin$, $Qinf$) \implies *sr* $\mathcal{Q}fin$
<proof>

lemma (in simplification) *split_INV2_stop_inf_sr*:
split_INV2 Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$) \implies *assemble* ($\mathcal{Q}fin$, $\mathcal{Q}inf$) = ($Qfin$, $Qinf$) \implies *fv* $Q' \subseteq$ *fv* $Qinf \implies$ *rrb* $Q' \implies$ *sr* Q'
<proof>

lemma (in simplification) *split_INV2_stop_FV*:
assumes *fv* $Q' \subseteq$ *fv* $Qinf$ *inf* $\mathcal{Q}fin$ $Q = \{\}$ *split_INV2* Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$) *assemble* ($\mathcal{Q}fin$, $\mathcal{Q}inf$) = ($Qfin$, $Qinf$)
shows *FV* Q $\mathcal{Q}fin$ Q'
<proof>

lemma (in simplification) *split_INV2_stop_EVAL*:
assumes *fv* $Q' \subseteq$ *fv* $Qinf$ *inf* $\mathcal{Q}fin$ $Q = \{\}$ *split_INV2* Q ($\mathcal{Q}fin$, $\mathcal{Q}inf$) *assemble* ($\mathcal{Q}fin$, $\mathcal{Q}inf$) = ($Qfin$, $Qinf$) $Qinf \triangleq$ Q'
shows *EVAL* Q $\mathcal{Q}fin$ Q'
<proof>

lemma (in simplification) *simplified_assemble*:
assemble ($\mathcal{Q}fin$, $\mathcal{Q}inf$) = ($Qfin$, $Qinf$) \implies *simplified* $\mathcal{Q}fin$

<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* $Q = \text{map } (\text{apsnd set}) (\text{filter } (\lambda(Q, _ :: (\text{nat} \times \text{nat}) \text{ list}). \exists x \in \text{fv } Q. \text{gen_impl } x \text{ } Q = []))$

$(\text{sorted_list_of_set } ((\text{apsnd sorted_list_of_set}) 'Q))$

definition *nongens_impl* $Q = \text{filter } (\lambda x. \text{gen_impl } x \text{ } Q = []) (\text{sorted_list_of_set } (\text{fv } Q))$

lemma *set_nongens_impl*: $\text{set } (\text{nongens_impl } Q) = \text{nongens } Q$

<proof>

lemma *set_fixfree_impl*: $\text{finite } Q \implies \forall (_, \text{Qeq}) \in Q. \text{finite } \text{Qeq} \implies \text{set } (\text{fixfree_impl } Q) = \text{fixfree } Q$

<proof>

lemma *fixfree_empty_iff*: $\text{finite } Q \implies \forall (_, \text{Qeq}) \in Q. \text{finite } \text{Qeq} \implies \text{fixfree } Q \neq \{\} \longleftrightarrow \text{fixfree_impl } Q \neq []$

<proof>

definition *inf_impl* $Q_{\text{fin}} \text{ } Q =$

$\text{map } (\text{apsnd set}) (\text{filter } (\lambda(Q_{\text{fix}}, \text{xy}). \text{disjointvars } Q_{\text{fix}} (\text{set } \text{xy}) \neq \{\} \vee \text{fv } Q_{\text{fix}} \cup \text{Field } (\text{set } \text{xy}) \neq \text{fv } Q))$

$(\text{sorted_list_of_set } ((\text{apsnd sorted_list_of_set}) 'Q_{\text{fin}}))$

lemma *set_inf_impl*: $\text{finite } Q_{\text{fin}} \implies \forall (_, \text{Qeq}) \in Q_{\text{fin}}. \text{finite } \text{Qeq} \implies \text{set } (\text{inf_impl } Q_{\text{fin}} \text{ } Q) = \text{inf } Q_{\text{fin}} \text{ } Q$

<proof>

lemma *inf_empty_iff*: $\text{finite } Q_{\text{fin}} \implies \forall (_, \text{Qeq}) \in Q_{\text{fin}}. \text{finite } \text{Qeq} \implies \text{inf } Q_{\text{fin}} \text{ } Q \neq \{\} \longleftrightarrow \text{inf_impl } Q_{\text{fin}} \text{ } Q \neq []$

<proof>

definition (in *simplification*) *split_impl* $:: ('a :: \{\text{infinite}, \text{linorder}\}, 'b :: \text{linorder}) \text{fmla} \Rightarrow (('a, 'b) \text{fmla} \times ('a, 'b) \text{fmla}) \text{nres}$ **where**

split_impl $Q = \text{do } \{$

$Q' \leftarrow \text{rb_impl } Q;$

$Q_{\text{pair}} \leftarrow \text{WHILE}$

$(\lambda(Q_{\text{fin}}, _). \text{fixfree_impl } Q_{\text{fin}} \neq []) (\lambda(Q_{\text{fin}}, Q_{\text{inf}}). \text{do } \{$

$(Q_{\text{fix}}, \text{Qeq}) \leftarrow \text{RETURN } (\text{hd } (\text{fixfree_impl } Q_{\text{fin}}));$

$x \leftarrow \text{RETURN } (\text{hd } (\text{nongens_impl } Q_{\text{fix}}));$

$G \leftarrow \text{RETURN } (\text{hd } (\text{cov_impl } x \text{ } Q_{\text{fix}}));$

$\text{let } Q_{\text{fin}} = Q_{\text{fin}} - \{(Q_{\text{fix}}, \text{Qeq})\} \cup$

$\{(\text{simp } (\text{Conj } Q_{\text{fix}} (\text{DISJ } (\text{qps } G))), \text{Qeq})\} \cup$

$(\bigcup y \in \text{eqs } x \text{ } G. \{(\text{cp } (Q_{\text{fix}}[x \rightarrow y]), \text{Qeq} \cup \{(x, y)\})\});$

$\text{let } Q_{\text{inf}} = Q_{\text{inf}} \cup \{\text{cp } (Q_{\text{fix}} \perp x)\};$

$\text{RETURN } (Q_{\text{fin}}, Q_{\text{inf}}))$

$\{(Q', \{\}), \{\}\};$

$Q_{\text{pair}} \leftarrow \text{WHILE}$

$(\lambda(Q_{\text{fin}}, _). \text{inf_impl } Q_{\text{fin}} \text{ } Q \neq []) (\lambda(Q_{\text{fin}}, Q_{\text{inf}}). \text{do } \{$

$Q_{\text{pair}} \leftarrow \text{RETURN } (\text{hd } (\text{inf_impl } Q_{\text{fin}} \text{ } Q));$


```

    let Qfin = Qfin - {Qpair};
    let Qinf = Qinf ∪ {CONJ Qpair};
    RETURN (Qfin, Qinf)}
  Qpair;
  let (Qfin, Qinf) = assemble Qpair;
  Qinf ← rb_impl Qinf;
  RETURN (Qfin, Qinf)}

```

lemma (in *simplification*) *split_INV2_imp_split_INV1*: *split_INV2 Q Qpair* \implies *split_INV1 Q Qpair*
 ⟨*proof*⟩

lemma *hd_fixfree_impl_props*:
assumes *finite Q* $\forall (_, Qeq) \in Q$. *finite Qeq* *fixfree_impl Q* $\neq []$
shows *hd (fixfree_impl Q) \in Q nongens (fst (hd (fixfree_impl Q)))* $\neq \{\}$
 ⟨*proof*⟩

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

definition (in *simplification*) *split_impl_det* :: ('a :: {infinite, linorder}, 'b :: linorder) *fmla* \Rightarrow (('a, 'b) *fmla* \times ('a, 'b) *fmla*) *dres* **where**

```

  split_impl_det Q = do {
    Q' ← rb_impl_det Q;
    Qpair ← dWHILE
      ( $\lambda(Qfin, \_)$ . fixfree_impl Qfin  $\neq []$ ) ( $\lambda(Qfin, Qinf)$ . do {
        (Qfix, Qeq) ← dRETURN (hd (fixfree_impl Qfin));
        x ← dRETURN (hd (nongens_impl Qfix));
        G ← dRETURN (hd (cov_impl x Qfix));
        let Qfin = Qfin - {(Qfix, Qeq)} ∪
          {(simp (Conj Qfix (DISJ (qps G))), Qeq)} ∪
          ( $\bigcup y \in eqs x G$ . {(cp (Qfix[x  $\rightarrow$  y)], Qeq  $\cup$  {(x,y)}})});
        let Qinf = Qinf  $\cup$  {cp (Qfix  $\perp$  x)};
        dRETURN (Qfin, Qinf)}
      )
    Qpair ← dWHILE
      ( $\lambda(Qfin, \_)$ . inf_impl Qfin Q  $\neq []$ ) ( $\lambda(Qfin, Qinf)$ . do {
        Qpair ← dRETURN (hd (inf_impl Qfin Q));
        let Qfin = Qfin - {Qpair};
        let Qinf = Qinf  $\cup$  {CONJ Qpair};
        dRETURN (Qfin, Qinf)}
      )
    Qpair;
    let (Qfin, Qinf) = assemble Qpair;
    Qinf ← rb_impl_det Qinf;
    dRETURN (Qfin, Qinf)}

```

lemma (in *simplification*) *split_impl_det_refines_split_impl*: *nres_of (split_impl_det Q)* \leq *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 ≡ λb. Pred "B" [Var b] :: (string, string) fmla
  and P ≡ λb p. Pred "P" [Var b, Var p] :: (string, string) fmla
  and S ≡ λp 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 ≡ λb. Pred "B" [Var b] :: (string, string) fmla
  and P ≡ λb 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  $B \equiv \lambda b. \text{Pred } "B" [\text{Var } b] :: (\text{string}, \text{string}) \text{ fmla}$ 
notes  $cp.simps[simp\ del]$ 
begin

definition  $Q\_eq$  where
 $Q\_eq = \text{Conj } (B\ x) (u \approx v)$ 
definition  $Q\_eq\_split\_fin :: (\text{string}, \text{string}) \text{ fmla}$  where
 $Q\_eq\_split\_fin = \text{Bool False}$ 
definition  $Q\_eq\_split\_inf :: (\text{string}, \text{string}) \text{ fmla}$  where
 $Q\_eq\_split\_inf = \text{Exists } x (B\ x)$ 

lemma  $ex\_split\_Q\_eq: \text{the\_res } (SPLIT\ Q\_eq) = (Q\_eq\_split\_fin, Q\_eq\_split\_inf)$ 
 $\langle \text{proof} \rangle$ 

end

```

7.4 Splitting the "Suspicious Users" Query

```

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

definition  $Q\_susp\_user\_split\_fin = \text{Conj } Q\_susp\_user\_rb (\text{Exists } s (\text{Exists } p (S\ p\ u\ s)))$ 
definition  $Q\_susp\_user\_split\_inf = \text{Exists } b (\text{Conj } (B\ b) (\text{Forall } p (\text{Neg } (P\ b\ p))))$ 

lemma  $ex\_split\_Q\_susp\_user: \text{the\_res } (SPLIT\ Q\_susp\_user) = (Q\_susp\_user\_split\_fin, Q\_susp\_user\_split\_inf)$ 
 $\langle \text{proof} \rangle$ 

end

```

8 Collected Results from the ICDT'22 Paper

```

global\_interpretation  $icdt22: \text{simplification } \lambda x. x\ \lambda x. \text{True}$ 
 $\langle \text{proof} \rangle$ 

lemma  $cov\_eval\_fin:$ 
assumes  $cov\ x (Q :: ('a :: \{\text{infinite}, \text{linorder}\}, 'b :: \text{linorder}) \text{ fmla})\ G\ x \in \text{fv } Q$ 
 $\text{finite } (\text{adom } I) \wedge \sigma. \neg \text{sat } (Q \perp x)\ I\ \sigma$ 
shows  $\text{eval } Q\ I = \text{eval } (\text{Disj } (\text{Conj } Q (\text{DISJ } (qps\ G))) (\text{DISJ } ((\lambda y. \text{Conj } (cp\ (Q[x \rightarrow y])) (x \approx y)) \text{ 'eqs } x\ G)))\ I$ 
 $\langle \text{proof} \rangle$ 

```

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

```

lemmas  $icdt22\_lemma\_1 = \text{gen\_fv } \text{gen\_sat } \text{gen\_cp\_erase}$ 
lemmas  $icdt22\_definition\_2 = \text{sub.simps } \text{nongens\_def } \text{rrb\_def } \text{sr\_def}$ 
lemmas  $icdt22\_lemma\_3 = \text{ex\_cov } \text{cov\_sat\_erase}$ 
lemmas  $icdt22\_lemma\_4 = \text{cov\_fv } \text{cov\_equiv}[OF\_ \text{refl}]$ 

```

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: $\llbracket \text{gen } x \ Q \ G; \ Q_{qp} \in G \rrbracket \implies x \in \text{fv } Q_{qp} \wedge \text{fv } Q_{qp} \subseteq \text{fv } Q$
 $\llbracket \text{gen } x \ Q \ G; \ \text{sat } Q \ I \ \sigma \rrbracket \implies \exists Q_{qp} \in G. \ \text{sat } Q_{qp} \ I \ \sigma$
 $\llbracket \text{gen } x \ Q \ G; \ Q_{qp} \in G \rrbracket \implies \text{cp } (Q_{qp} \perp x) = \text{Bool False}$

icdt22_definition_2: $\text{sub } (\text{Bool } t) = \{\text{Bool } t\}$
 $\text{sub } (\text{Pred } p \ ts) = \{\text{Pred } p \ ts\}$
 $\text{sub } (\text{fmla.Eq } x \ t) = \{\text{fmla.Eq } x \ t\}$
 $\text{sub } (\text{Neg } Q) = \text{insert } (\text{Neg } Q) (\text{sub } Q)$
 $\text{sub } (\text{Conj } Q1 \ Q2) = \text{insert } (\text{Conj } Q1 \ Q2) (\text{sub } Q1 \cup \text{sub } Q2)$
 $\text{sub } (\text{Disj } Q1 \ Q2) = \text{insert } (\text{Disj } Q1 \ Q2) (\text{sub } Q1 \cup \text{sub } Q2)$
 $\text{sub } (\text{Exists } z \ Q) = \text{insert } (\text{Exists } z \ Q) (\text{sub } Q)$
 $\text{nongens } Q = \{x \in \text{fv } Q. \ \neg \text{Gen } x \ Q\}$
 $\text{rrb } Q = (\forall y \ Qy. \ \text{Exists } y \ Qy \in \text{sub } Q \longrightarrow \text{Gen } y \ Qy)$
 $\text{sr } Q = (\text{rrf } Q \wedge \text{rrb } Q)$

icdt22_lemma_3: $\llbracket \text{rrb } Q; \ x \in \text{fv } Q \rrbracket \implies \exists G. \ \text{cov } x \ Q \ G$
 $\llbracket \text{cov } x \ Q \ G; \ \text{sat } (\text{Neg } (\text{Disj } (\text{DISJ } (\text{qps } G)) (\text{DISJ } ((\approx) \ x \ ' \ \text{eqs } x \ G)))) \ I \ \sigma \rrbracket \implies \text{sat } Q \ I \ \sigma$
 $= \text{sat } (\text{cp } (Q \perp x)) \ I \ \sigma$

icdt22_lemma_4: $\llbracket \text{cov } x \ Q \ G; \ x \in \text{fv } Q; \ Q_{qp} \in G \rrbracket \implies x \in \text{fv } Q_{qp} \wedge \text{fv } Q_{qp} \subseteq \text{fv } Q$
 $\text{cov } x \ Q \ G \implies Q \triangleq \text{Disj } (\text{Conj } Q (\text{DISJ } (\text{qps } G))) (\text{Disj } (\text{DISJ } ((\lambda y. \ \text{Conj } (\text{cp } (Q[x \rightarrow y]))) (x \approx y)) \ ' \ \text{eqs } x \ G)) (\text{Conj } (Q \perp x) (\text{Neg } (\text{Disj } (\text{DISJ } (\text{qps } G)) (\text{DISJ } ((\approx) \ x \ ' \ \text{eqs } x \ G))))))$

icdt22_lemma_5: $\llbracket \text{cov } x \ Q \ G; \ x \in \text{fv } Q \rrbracket \implies \text{Exists } x \ Q \triangleq \text{Disj } (\text{Exists } x (\text{Conj } Q (\text{DISJ } (\text{qps } G)))) (\text{Disj } (\text{DISJ } ((\lambda y. \ \text{cp } (Q[x \rightarrow y]))) \ ' \ \text{eqs } x \ G)) (\text{cp } (Q \perp x))$

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: $\llbracket cov\ x\ Q\ G; x \in fv\ Q; finite\ (adom\ I); \bigwedge \sigma. \neg\ sat\ (Q \perp x)\ I\ \sigma \rrbracket \implies eval\ Q\ I = eval\ (Disj\ (Conj\ Q\ (DISJ\ (qps\ G)))\ (DISJ\ ((\lambda y. Conj\ (cp\ (Q[x \rightarrow y]))\ (x \approx y))\ 'eqs\ x\ G)))\ I$

$\llbracket cov\ x\ Q\ G; x \in fv\ Q; finite\ (adom\ I); sat\ (Q \perp x)\ I\ \sigma \rrbracket \implies infinite\ (eval\ Q\ I)$

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

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

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

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

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.