```
POLYNOM_2
---------
*C polynom_2_begin            ************ POLYNOM_2 ************
*C oalist_com_1
               ==============================
               ORDERED A-LIST TYPE DEFINITION
               ==============================
*D before_df   before{<a:a:*>}(<u:u:*>;<ps:ps:*>)== before{}(<a>; <u>; <ps>)
               before(<u:u:*>;<ps:ps:*>)== before{}(<a>; <u>; <ps>)
*A before                     before(u;ps) ==  null(ps) ∨b(hd(ps) <b u)
*T before_wf  5.5 sec.
⊢ ∀a:DSet. ∀ps:|a| List. ∀u:|a|.  before(u;ps) ∈ 𝔹
|
BY (Unfold ‘before‘ 0 THEN RepD ...a)
|
1. a: DSet
2. ps: |a| List
3. u: |a|
⊢ (null(ps) ∨b(hd(ps) <b u)) ∈ 𝔹
|
BY (Unfold ‘bor‘ 0 THEN SplitOnConclITE ...)
|
4. ¬(ps = [])
⊢ ||ps|| ≥ 1
|
BY (BLemma ‘pos_length‘ ...)
*T comb_for_before_wf  1.1 sec.
⊢ (λa,ps,u,z.before(u;ps)) ∈ a:DSet ⟶ ps:|a| List ⟶ u:|a| ⟶ ↓True ⟶ 𝔹
|
BY ProveOpCombTyping ‘before_wf‘
*M before_eval let before_nilC =
                 MacroC ‘before_nilC‘
                 (RepeatC (UnfoldsC ‘‘before null‘‘) ANDTHENC ReduceC)
                 ⌜before(u;[])⌝
                 IdC ⌜tt⌝;;
               let before_consC =
                 MacroC ‘befire_consC‘
                 (RepeatC (UnfoldsC ‘‘before null hd‘‘) ANDTHENC ReduceC)
                 ⌜before(u;v::vs)⌝
                 IdC  ⌜v <b u⌝;;
               add_AbReduce_conv ‘before‘
                 (before_nilC ORELSEC before_consC);;
*T before_trans  5.6 sec.
⊢ ∀a:LOSet. ∀u,v:|a|. ∀ws:|a| List.  v <a u ⇒ ↑before(v;ws) ⇒ ↑before(u;ws)
|
BY (RepeatMFor 4 (D 0)
|   THENM OnVar ‘ws’ D ...a)
|  THEN AbReduce 0 THEN (RW bool_to_propC 0 ...)
|
1. a: LOSet
2. u: |a|
3. v: |a|
4. ws: |a| List
5. u1: |a|
6. v1: |a| List
7. v <a u
8. u1 <a v
```

```
⊢ u1 <a u
|
BY (FLemma ‘qoset_lt_trans‘ [7;8] ...)
*C sd_ordered_com
               sd_ordered = s(tricly) d(escending) ordered
               Design choice here between boolean and prop
               valued definition. Went with bool valued since it
               trival then to prove decidable.
               Decided to define this with an auxiliary predicate
               ‘before’ rather than stripping off two head elements
               and comparing them directly. ‘before’  makes
               one-cons unrolling of lists OK when reasoning with
               sd_ordered. It also hides the partial function ‘hd’.
*D sd_ordered_df
               sd_ordered{<s:s:*>}(<as:as:*>)== sd_ordered{}(<s>; <as>)
               sd_ordered(<as:as:*>)== sd_ordered{}(<s>; <as>)
*M sd_ordered_ml
               sd_ordered(as)
               ==r case as of [] => tt | a::bs => before(a;bs) ∧b sd_ordered(bs) esac
*T sd_ordered_wf  1.5 sec.
⊢ ∀s:DSet. ∀as:|s| List.  sd_ordered(as) ∈ 𝔹
|
BY (UnivCD ...a)
|  THEN OnVar ‘as’ ListInd
|\
| 1. s: DSet
| 2. as: |s| List
| ⊢ sd_ordered([]) ∈ 𝔹
| |
1 BY RecUnfold ‘sd_ordered‘ 0
|    THEN AbReduce 0 THEN Auto
 \
  1. s: DSet
  2. as: |s| List
  3. u: |s|
  4. v: |s| List
  5. sd_ordered(v) ∈ 𝔹
  ⊢ sd_ordered(u::v) ∈ 𝔹
  |
  BY RecUnfold ‘sd_ordered‘ 0
  |  THEN AbReduce 0
  |
  ⊢ (before(u;v) ∧b sd_ordered(v)) ∈ 𝔹
  |
  BY Auto
*T comb_for_sd_ordered_wf  0.7 sec.
⊢ (λs,as,z.sd_ordered(as)) ∈ s:DSet → as:|s| List → ↓True → 𝔹
|
BY ProveOpCombTyping ‘sd_ordered_wf‘
*M sd_ordered_eval
               let sd_ordered_nilC =
                 MacroC ‘sd_ordered_nilC‘
                  (RecUnfoldC ‘sd_ordered‘ ANDTHENC PrimReduceC) ⌜sd_ordered([])⌝
                  IdC ⌜tt⌝
               ;;
               let sd_ordered_consC =
                 MacroC ‘sd_ordered_consC‘
```

```
                  (RecUnfoldC ‘sd_ordered‘ ANDTHENC PrimReduceC) ⌜sd_ordered(a::as)⌝
                  IdC ⌜before(a;as) ∧b sd_ordered(as)⌝
               ;;
               add_AbReduce_conv ‘sd_ordered‘
                 (sd_ordered_nilC ORELSEC sd_ordered_consC) ;;
*C sd_ordered_char_com
               Alternate characterization of
               stricly-descending order predicate.
               Probably, would have been easiest to use this definition
               from the start.
               The proof of this theorem shows how annoying the
               bool/prop difference is.
*T sd_ordered_char  45.0 sec.
⊢ ∀s:QOSet. ∀us:|s| List.  sd_ordered(us) = HTFor{<𝔹,∧b>} v::vs ∈ us. ∀bw(:|s|) ∈ vs. w <b v
|
BY (RepD THENM New [‘p’;‘ps’] (OnVar ‘us’ ListInd)
|   THENM AbReduce 0 ...a)
|\
| 1. s: QOSet
| 2. us: |s| List
| ⊢ tt = tt
| |
1 BY Auto
 \
  1. s: QOSet
  2. us: |s| List
  3. p: |s|
  4. ps: |s| List
  5. sd_ordered(ps) = HTFor{<𝔹,∧b>} v::vs ∈ ps. ∀bw(:|s|) ∈ vs. w <b v
  ⊢ before(p;ps) ∧b sd_ordered(ps)
  |  = (∀bw(:|s|) ∈ ps. w <b p) ∧b (HTFor{<𝔹,∧b>} v::vs ∈ ps. ∀bw(:|s|) ∈ vs. w <b v)
  |
  BY (RWH (RevHypC 5) 0
  |   THENM BLemma ‘iff_imp_equal_bool‘
  |   THENM RW bool_to_propC 0 ...)
  |\
  | 6. ↑before(p;ps)
  | 7. ↑sd_ordered(ps)
  | ⊢ ↑(∀bw(:|s|) ∈ ps. w <b p)
  | |
  1 BY (MoveDepHypsToConclFor D 4
  | |   THENM Reduce 0 ...a)
  | |\
  | | 5. sd_ordered([]) = HTFor{<𝔹,∧b>} v::vs ∈ []. ∀bw(:|s|) ∈ vs. w <b v
  | | 6. ↑before(p;[])
  | | 7. ↑sd_ordered([])
  | | ⊢ True
  | | |
  1 2 BY Trivial
  |  \
  |   5. u: |s|
  |   6. v: |s| List
  |   7. sd_ordered(u::v) = HTFor{<𝔹,∧b>} v::vs ∈ u::v. ∀bw(:|s|) ∈ vs. w <b v
  |   8. ↑before(p;u::v)
  |   9. ↑sd_ordered(u::v)
  |   ⊢ ↑(u <b p ∧b (∀bw(:|s|) ∈ v. w <b p))
  |   |
```

```
1   BY (Reduce 8 THEN OnClauses [0;8] (RW bool_to_propC) ...)
|   |
|   8. u <s p
|   ⊢ ↑(∀b w(:|s|) ∈ v. w <b p)
|   |
1   BY (RWH (HypC 7) 9
|   |   THENM Reduce 9
|   |   THENM RW bool_to_propC 9 ...)
|   |
|   9. ↑(∀b w(:|s|) ∈ v. w <b u)
|   10. ↑(HTFor{<𝔹,∧b>} v::vs ∈ v. ∀b w(:|s|) ∈ vs. w <b v)
|   |
1   BY (OnMCls [0;9] (RWH (LemmaC `ball_char`)) ...a)
|   |
|   9. ∀w:|s|. ↑(w ∈b v) ⇒ ↑(w <b u)
|   ⊢ ∀w:|s|. ↑(w ∈b v) ⇒ ↑(w <b p)
|   |
1   BY (RepD THENM InstHyp [⌈w⌉] 9 ...a)
|   |
|   11. w: |s|
|   12. ↑(w ∈b v)
|   13. ↑(w <b u)
|   ⊢ ↑(w <b p)
|   |
1   BY (OnMCls [0;-1] (RW bool_to_propC)
|       THENM RelRST ...)
 \
  6. ↑(∀b w(:|s|) ∈ ps. w <b p)
  7. ↑sd_ordered(ps)
  ⊢ ↑before(p;ps)
  |
  BY (Thin 5 THENM MoveDepHypsToConclFor D 4
  |   THENM Thin 4 ...a)
  |\
  | 4. ↑(∀b w(:|s|) ∈ []. w <b p)
  | 5. ↑sd_ordered([])
  | ⊢ ↑before(p;[])
  | |
  1 BY (Reduce 0 ...)
   \
    4. u: |s|
    5. v: |s| List
    6. ↑(∀b w(:|s|) ∈ u::v. w <b p)
    7. ↑sd_ordered(u::v)
    ⊢ ↑before(p;u::v)
    |
    BY (RW (LemmaC `ball_char`) 6 ...a)
    |  THEN Reduce 0
    |
    6. ∀w:|s|. ↑(w ∈b (u::v)) ⇒ ↑(w <b p)
    ⊢ ↑(u <b p)
    |
    BY (BHyp 6
    |   THENM Reduce 0 ...a)
    |
    ⊢ ↑((u =b u) ∨b(u ∈b v))
    |
```

```
      BY (RW bool_to_propC 0 THENM Sel 1 (D 0) ...)
*T before_all_imp_count_zero  16.6 sec.
⊢ ∀s:QOSet. ∀a:|s|. ∀cs:|s| List.  ↑(∀b c(:|s|) ∈ cs. c <b a) ⇒ a #∈ cs = 0
|
BY (CDToVarThen ‘cs’ ListIndA
|   THENM Reduce 0 THENM D 0  ...a)
|\
| 1. s: QOSet
| 2. a: |s|
| 3. True
| ⊢ 0 = 0
| |
1 BY Auto
 \
  1. s: QOSet
  2. a: |s|
  3. c: |s|
  4. cs: |s| List
  5. ↑(∀b c(:|s|) ∈ cs. c <b a) ⇒ a #∈ cs = 0
  6. ↑(c <b a ∧b (∀b c(:|s|) ∈ cs. c <b a))
  ⊢ b2i(c =b a) + (a #∈ cs) = 0
  |
  BY (SeqOnM
  |    [RW bool_to_propC (-1) ; D (-1)
  |    ;Unfold ‘b2i‘ 0 ; SplitOnConclITE
  |    ; Try (ArithSimp 0)] ...a)
  |\
  | 6. c <s a
  | 7. ↑(∀b c(:|s|) ∈ cs. c <b a)
  | 8. c = a
  | ⊢ 1 + (a #∈ cs) = 0
  | |
  1 BY (RelRST ...)
   \
    6. c <s a
    7. ↑(∀b c(:|s|) ∈ cs. c <b a)
    8. ¬(c = a)
    ⊢ a #∈ cs = 0
    |
    BY (HypBackchain ...)
*T sd_ordered_count  20.5 sec.
⊢ ∀s:QOSet. ∀as:|s| List.  ↑sd_ordered(as) ⇒ (∀c:|s|. c #∈ as ≤ 1)
|
BY (RepD THENM OnVar ‘as’ ListIndA
|   THENM D 0 ...a)
|\
| 1. s: QOSet
| 2. c: |s|
| 3. ↑sd_ordered([])
| ⊢ c #∈ [] ≤ 1
| |
1 BY (Reduce 0 ...)
 \
  1. s: QOSet
  2. c: |s|
  3. a: |s|
  4. as: |s| List
```

```
  5. ↑sd_ordered(as) ⇒ c #∈ as ≤ 1
  6. ↑sd_ordered(a::as)
  ⊢ c #∈ (a::as) ≤ 1
  |
  BY (RWH (LemmaC ‘sd_ordered_char‘) 6
  |   THENM Reduce (-1)
  |   THENM RW bool_to_propC (-1) THENM D (-1) ...a)
  |
  6. ↑(∀b w(:|s|) ∈ as. w <b a)
  7. ↑(HTFor{<𝔹,∧b>} v::vs ∈ as. ∀b w(:|s|) ∈ vs. w <b v)
  |
  BY (Reduce 0 THENM Unfold ‘b2i‘ 0
  |   THENM SplitOnConclITE ...a)
  |\
  | 8. a = c
  | ⊢ 1 + (c #∈ as) ≤ 1
  | |
  1 BY (RWH (RevHypC 8) 0
  |     THENM FLemma ‘before_all_imp_count_zero‘ [6] ...)
   \
    8. ¬(a = c)
    ⊢ 0 + (c #∈ as) ≤ 1
    |
    BY (ArithSimp 0
        THENM BHyp 5
        THENM RWH (LemmaC ‘sd_ordered_char‘) 0 ...)
*C oalist_com  (strictly) o(rdered) a(ssociation) lists
               This is an experimental use of ‘set’
               constructors. Proofs are unlikely to be
               that clean.
               Old Definition:
               oal(a;b) ==  {ps:(a × {z:b↓set| ¬(z = e)}) List| ↑sd_ordered(map(λx.x.1;ps))}
               This gave slower Inclusion proofs because dset a × {z:b↓set| ¬(z = e)}
               exposed when type checking with funs typed over concrete Lists
*D oalist_df                  oal(<a:a:*>;<b:b:*>)== oalist{}(<a>; <b>)
*A oalist      oal(a;b) ==
                 {ps:(a × b↓set) List| ↑sd_ordered(map(λx.x.1;ps)) ∧ ¬↑(e ∈b map(λx.x.2;ps))}
*T oalist_wf  10.6 sec.
⊢ ∀a:LOSet. ∀b:AbMon.  oal(a;b) ∈ DSet
|
BY (Unfold ‘oalist‘ 0 ...)
*M oalist_ml                  note_reduction_strength ‘oalist‘ ‘8‘;;
*T oalist_car_properties  0.0 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ws:|oal(a;b)|.  ↑sd_ordered(map(λx.x.1;ws)) ∧ ¬↑(e ∈b map(λx.x.2;ws))
|
BY Fiat
*T before_imp_before_all  18.7 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k:|a|. ∀ps:|oal(a;b)|.
|    ↑before(k;map(λz.z.1;ps)) ⇒ ↑(∀b x(:|a|) ∈ map(λz.z.1;ps). x <b k)
|
BY (RepD ...a)
|
1. a: LOSet
2. b: AbMon
3. k: |a|
4. ps: |oal(a;b)|
5. ↑before(k;map(λz.z.1;ps))
```

```
⊢ ↑(∀b x(:|a|) ∈ map(λz.z.1;ps). x <b k)
|
BY AddCarProperties 4 THEN RepD
|
5. ↑sd_ordered(map(λx.x.1;ps))
6. ¬↑(e ∈b map(λx.x.2;ps))
7. ↑before(k;map(λz.z.1;ps))
|
BY Assert ⌜↑sd_ordered(k::map(λz.z.1;ps))⌝
|  THENA (Reduce 0 THEN RW bool_to_propC 0 THEN Auto)
|
8. ↑sd_ordered(k::map(λz.z.1;ps))
|
BY (RWH (LemmaC `sd_ordered_char`) 8
|   THENM Reduce 8
|   THENM RW bool_to_propC 8
|   THENM D 8 ...a)
|
8. ↑(∀b w(:|a|) ∈ map(λz.z.1;ps). w <b k)
9. ↑(HTFor{<ℙ,∧b>} v::vs ∈ map(λz.z.1;ps). ∀b w(:|a|) ∈ vs. w <b v)
|
BY Trivial
*T before_all_imp_before  11.7 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k:|a|. ∀ps:(|a| × |b|) List.
|    ↑(∀b x(:|a|) ∈ map(λz.z.1;ps). x <b k) ⇒ ↑before(k;map(λz.z.1;ps))
|
BY (CDToVarThen `ps` D ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. k: |a|
| 4. ps: (|a| × |b|) List
| ⊢ ↑(∀b x(:|a|) ∈ map(λz.z.1;[]). x <b k) ⇒ ↑before(k;map(λz.z.1;[]))
| |
1 BY (Reduce 0 ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. k: |a|
  4. ps: (|a| × |b|) List
  5. u: |a| × |b|
  6. v: (|a| × |b|) List
  ⊢ ↑(∀b x(:|a|) ∈ map(λz.z.1;u::v). x <b k) ⇒ ↑before(k;map(λz.z.1;u::v))
  |
  BY Reduce 0 THEN (RW bool_to_propC 0 ...)
*T nil_in_oalist  7.7 sec.
⊢ ∀a:LOSet. ∀b:AbMon.  [] ∈ |oal(a;b)|
|
BY (RepD ...a)
|
1. a: LOSet
2. b: AbMon
⊢ [] ∈ |oal(a;b)|
|
BY AbReduce 0
|  THEN (MemTypeCD THEN AbReduce 0 ...)
|
```

```
⊢ ¬False
|
BY (D 0 ...)
*T cons_in_oalist  41.2 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ws:|oal(a;b)|. ∀x:|a|. ∀y:|b|.
|    ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ (<x, y>::ws) ∈ |oal(a;b)|
|
BY (RepD ...a)
|
1. a: LOSet
2. b: AbMon
3. ws: |oal(a;b)|
4. x: |a|
5. y: |b|
6. ↑before(x;map(λx.x.1;ws))
7. ¬(y = e)
⊢ (<x, y>::ws) ∈ |oal(a;b)|
|
BY (AddCarProperties 3
|   THENM AbReduce 0
|   THENM MemTypeCD
|   THEN AbReduce 0 ...)
|\
| 4. ↑sd_ordered(map(λx.x.1;ws))
| 5. ¬↑(e ∈b map(λx.x.2;ws))
| 6. x: |a|
| 7. y: |b|
| 8. ↑before(x;map(λx.x.1;ws))
| 9. ¬(y = e)
| ⊢ ↑(before(x;map(λx.x.1;ws)) ∧b sd_ordered(map(λx.x.1;ws)))
| |
1 BY (RW bool_to_propC 0 ...)
 \
  4. ↑sd_ordered(map(λx.x.1;ws))
  5. ¬↑(e ∈b map(λx.x.2;ws))
  6. x: |a|
  7. y: |b|
  8. ↑before(x;map(λx.x.1;ws))
  9. ¬(y = e)
  ⊢ ¬↑((y =b e) ∨b(e ∈b map(λx.x.2;ws)))
  |
  BY (D 0
      THENM RW bool_to_propC (-1)
      THENM D (-1)  ...)
*T cons_pr_in_oalist  35.5 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ws:|oal(a;b)|. ∀x:|a|. ∀y:|b|.
|    ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ (<x, y>::ws) ∈ |oal(a;b)|
|
BY (RepD ...a)
|
1. a: LOSet
2. b: AbMon
3. ws: |oal(a;b)|
4. x: |a|
5. y: |b|
6. ↑before(x;map(λx.x.1;ws))
7. ¬(y = e)
```

```
⊢ (<x, y>::ws) ∈ |oal(a;b)|
|
BY (D 3 ...a) THENM AbReduce 0 THEN (MemTypeCD ...a)
|\
| 3. ws: |((a × b↓set) List)|
| 4. ↑sd_ordered(map(λx.x.1;ws)) ∧ ¬↑(e ∈b map(λx.x.2;ws))
| 5. x: |a|
| 6. y: |b|
| 7. ↑before(x;map(λx.x.1;ws))
| 8. ¬(y = e)
| ⊢ (<x, y>::ws) ∈ (|a| × |b|) List
| |
1 BY Auto
|\
| 3. ws: |((a × b↓set) List)|
| 4. ↑sd_ordered(map(λx.x.1;ws))
| 5. ¬↑(e ∈b map(λx.x.2;ws))
| 6. x: |a|
| 7. y: |b|
| 8. ↑before(x;map(λx.x.1;ws))
| 9. ¬(y = e)
| ⊢ ↑sd_ordered(map(λx.x.1;<x, y>::ws))
| |
1 BY RecUnfold `sd_ordered` 0
|    THEN AbReduce 0 THEN (RWH bool_to_propC 0 ...)
 \
  3. ws: |((a × b↓set) List)|
  4. ↑sd_ordered(map(λx.x.1;ws))
  5. ¬↑(e ∈b map(λx.x.2;ws))
  6. x: |a|
  7. y: |b|
  8. ↑before(x;map(λx.x.1;ws))
  9. ¬(y = e)
  ⊢ ¬↑(e ∈b map(λx.x.2;<x, y>::ws))
  |
  BY AbReduce 0 THEN (RW bool_to_propC 0
                      THENM ProveProp ...)
*C oal_nil_cons_com
               ====================================
               NIL AND CONS CONSTRUCTORS FOR OALISTS
               ====================================
*D oal_nil_df  Parens ::Prec(preop)::  00<a:set:*>,<b:mon:*>== oal_nil{}(<a>; <b>)
               00== oal_nil{}(<a>; <b>)
*A oal_nil                     00 ==  []
*T oal_nil_wf  7.9 sec.
⊢ ∀a:LOSet. ∀b:AbMon.  00 ∈ |oal(a;b)|
|
BY (Unfold `oal_nil` 0 ...)
|
1. a: LOSet
2. b: AbMon
⊢ [] ∈ |oal(a;b)|
|
BY AbReduce 0
|  THEN (MemTypeCD THEN AbReduce 0 ...)
|
⊢ ¬False
```

```
|
BY (D 0 ...)
*D oal_cons_pr_df
               oal_cons_pr(<x:x:*>;<y:y:*>;<ws:ws:*>)== oal_cons_pr{}(<x>; <y>; <ws>)
*A oal_cons_pr                oal_cons_pr(x;y;ws) ==  <x, y>::ws
*T oal_cons_pr_wf  0.0 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ws:|oal(a;b)|. ∀x:|a|. ∀y:|b|.
|    ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ oal_cons_pr(x;y;ws) ∈ |oal(a;b)|
|
BY Unfold `oal_cons_pr` 0
   THEN Lemma `cons_in_oalist`
*C oal_case_ind
               ====================================
               OALIST CASE SPLIT AND INDUCTION LEMMAS
               ====================================
*C oalist_cases_com
               NB: it helps typechecking here to make Q's domain type
               larger than |oal(a;b)| (otherwise get extra obligations
               to show that subtype predicates are satisfied).
               With hindsight, it might have been cleaner to use oal_nil
               and define an oal_cons_pr constructor for use in the cases
               and induction lemmas. Then, the nil_in_oalist and cons_in_oalist wf lemmas would
                never have to be manually invoked.
               Making such a change would require updating all the MacroC
               conversion involving oalists.
*T oalist_cases  77.1 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀Q:(|a| × |b|) List ⟶ ℙ.
|    Q[[]]
|    ⇒ (∀ws:|oal(a;b)|. ∀x:|a|. ∀y:|b|.
|          ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ Q[<x, y>::ws])
|    ⇒ {∀ws:|oal(a;b)|. Q[ws]}
|
BY (Unfold `guard` 0 THEN RepD ...a)
|
1. a: LOSet
2. b: AbMon
3. Q: (|a| × |b|) List ⟶ ℙ
4. Q[[]]
5. ∀ws:|oal(a;b)|. ∀x:|a|. ∀y:|b|.  ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ Q[<x, y>::ws]
6. ws: |oal(a;b)|
⊢ Q[ws]
|
BY D 6 THEN MoveToConcl (-1)
|  THEN D 6 THEN AbReduce 0 THEN (D 0 ...a)
|\
| 6. ws: |(a × b↓set)| List
| 7. True ∧ ¬False
| ⊢ Q[[]]
| |
1 BY Trivial
 \
  6. ws: |(a × b↓set)| List
  7. u: |(a × b↓set)|
  8. v: |(a × b↓set)| List
  9. ↑(before(u.1;map(λx.x.1;v)) ∧b sd_ordered(map(λx.x.1;v)))
     ∧ ¬↑((u.2 =b e) ∨b(e ∈b map(λx.x.2;v)))
  ⊢ Q[u::v]
```

```
  |
  BY D 7 THEN AbReduce (-1)
  |  THEN (RW bool_to_propC (-1)  ...a)
  |  THEN D (-1)
  |
  7. u1: |a|
  8. u2: |(b↓set)|
  9. v: |(a × b↓set)| List
  10. ↑before(u1;map(λx.x.1;v)) ∧ ↑sd_ordered(map(λx.x.1;v))
  11. ¬(u2 = e ∨ ↑(e ∈b map(λx.x.2;v)))
  ⊢ Q[<u1, u2>::v]
  |
  BY (BHyp 5 ...a)
  |\
  | 10. ↑before(u1;map(λx.x.1;v))
  | 11. ↑sd_ordered(map(λx.x.1;v))
  | 12. ¬(u2 = e ∨ ↑(e ∈b map(λx.x.2;v)))
  | ⊢ v ∈ |oal(a;b)|
  | |
  1 BY RWAddr [1] AbReduceTopC 0
  | |  THEN MemTypeCD THEN Auto
  | |
  | ⊢ ¬↑(e ∈b map(λx.x.2;v))
  | |
  1 BY (D 0 THENM D 12
  |     THENM RW bool_to_propC 0
  |     THENM Sel 2 (D 0) ...)
  |\
  | ⊢ ↑before(u1;map(λx.x.1;v))
  | |
  1 BY RecEval ``sd_ordered`` 10
  |    THEN (RW bool_to_propC 10 ...)
   \
    ⊢ ¬(u2 = e)
    |
    BY % trivial prop reasoning... %
       (SeqOnM [D 0;D (-2);Sel 1 (D 0)] ...)
*T oalist_cases_a  83.5 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀Q:|oal(a;b)| ⟶ ℙ.
|    Q[[]]
|    ⇒ (∀ws:|oal(a;b)|. ∀x:|a|. ∀y:|b|.
|          ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ Q[<x, y>::ws])
|    ⇒ {∀ws:|oal(a;b)|. Q[ws]}
|
BY (RepD ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. Q: |oal(a;b)| ⟶ ℙ
| 4. Q[[]]
| 5. ∀ws:|oal(a;b)|. ∀x:|a|. ∀y:|b|.  ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ Q[<x, y>::ws]
| ⊢ ∀ws:|oal(a;b)|. Q[ws]
| |
1 BY (D 0 ...a)
| |
| 6. ws: |oal(a;b)|
| ⊢ Q[ws]
```

```
| |
1 BY D 6 THEN MoveToConcl (-1)
| |  THEN D 6 THEN AbReduce 0 THEN (D 0 ...a)
| |\
| | 6. ws: |(a × b↓set)| List
| | 7. True ∧ ¬False
| | ⊢ Q[[]]
| | |
1 2 BY Trivial
|  \
|   6. ws: |(a × b↓set)| List
|   7. u: |(a × b↓set)|
|   8. v: |(a × b↓set)| List
|   9. ↑(before(u.1;map(λx.x.1;v)) ∧_b sd_ordered(map(λx.x.1;v)))
|      ∧ ¬↑((u.2 =_b e) ∨_b(e ∈_b map(λx.x.2;v)))
|   ⊢ Q[u::v]
|   |
1   BY D 7 THEN AbReduce (-1)
|   |  THEN (RW bool_to_propC (-1)  ...a)
|   |  THEN D (-1)
|   |
|   7. u1: |a|
|   8. u2: |(b↓set)|
|   9. v: |(a × b↓set)| List
|   10. ↑before(u1;map(λx.x.1;v)) ∧ ↑sd_ordered(map(λx.x.1;v))
|   11. ¬(u2 = e ∨ ↑(e ∈_b map(λx.x.2;v)))
|   ⊢ Q[<u1, u2>::v]
|   |
1   BY (BHyp 5 ...a)
|   |\
|   | 10. ↑before(u1;map(λx.x.1;v))
|   | 11. ↑sd_ordered(map(λx.x.1;v))
|   | 12. ¬(u2 = e ∨ ↑(e ∈_b map(λx.x.2;v)))
|   | ⊢ v ∈ |oal(a;b)|
|   | |
1   2 BY RWAddr [1] AbReduceTopC 0
|   | |  THEN MemTypeCD THEN Auto
|   | |
|   | ⊢ ¬↑(e ∈_b map(λx.x.2;v))
|   | |
1   2 BY (D 0 THENM D 12
|   |     THENM RW bool_to_propC 0
|   |     THENM Sel 2 (D 0) ...)
|   |\
|   | ⊢ ↑before(u1;map(λx.x.1;v))
|   | |
1   2 BY RecEval ``sd_ordered`` 10
|   |    THEN (RW bool_to_propC 10 ...)
|    \
|     ⊢ ¬(u2 = e)
|     |
1     BY % trivial prop reasoning... %
|        (SeqOnM [D 0;D (-2);Sel 1 (D 0)] ...)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. Q: |oal(a;b)| ⟶ ℙ
```

```
| 4. Q[[]]
| 5. ws: |oal(a;b)|
| 6. x: |a|
| 7. y: |b|
| 8. ↑before(x;map(λx.x.1;ws))
| 9. ¬(y = e)
| ⊢ (<x, y>::ws) ∈ |oal(a;b)|
| |
1 BY (BLemma `cons_in_oalist` ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. Q: |oal(a;b)| ⟶ ℙ
  ⊢ [] ∈ |oal(a;b)|
  |
  BY (BLemma `nil_in_oalist` ...)
*T oalist_cases_c  0.0 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀Q:|oal(a;b)| ⟶ ℙ.
|    Q[00]
|    ⇒ (∀ws:|oal(a;b)|. ∀x:|a|. ∀y:|b|.
|          ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ Q[oal_cons_pr(x;y;ws)])
|    ⇒ {∀ws:|oal(a;b)|. Q[ws]}
|
BY Unfolds ``oal_nil oal_cons_pr`` 0 THEN Lemma `oalist_cases_a`
*T oalist_cases_b  26.5 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀Q:|oal(a;b)| ⟶ ℙ.
|    Q[[]]
|    ⇒ (∀ws:|oal(a;b)|. ∀k:|a|. ∀v:|b|.
|          ↑(∀b x(:|a|) ∈ map(λz.z.1;ws). x <b k) ⇒ ¬(v = e) ⇒ Q[<k, v>::ws])
|    ⇒ {∀ws:|oal(a;b)|. Q[ws]}
|
BY (RepD THENM BLemma `oalist_cases_a` ...)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. Q: |oal(a;b)| ⟶ ℙ
| 4. Q[[]]
| 5. ∀ws:|oal(a;b)|. ∀k:|a|. ∀v:|b|.
|      ↑(∀b x(:|a|) ∈ map(λz.z.1;ws). x <b k) ⇒ ¬(v = e) ⇒ Q[<k, v>::ws]
| 6. ws: |oal(a;b)|
| 7. x: |a|
| 8. y: |b|
| 9. ↑before(x;map(λx.x.1;ws))
| 10. ¬(y = e)
| ⊢ Q[<x, y>::ws]
| |
1 BY (BHyp 5 ...)
| |
| ⊢ ↑(∀b x1(:|a|) ∈ map(λz.z.1;ws). x1 <b x)
| |
1 BY (BLemma `before_imp_before_all` ...)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. Q: |oal(a;b)| ⟶ ℙ
| 4. Q[[]]
| 5. ws: |oal(a;b)|
```

```
| 6. k: |a|
| 7. v: |b|
| 8. ↑(∀b x(:|a|) ∈ map(λz.z.1;ws). x <b k)
| 9. ¬(v = e)
| ⊢ (<k, v>::ws) ∈ |oal(a;b)|
| |
1 BY (BLemma ‘cons_in_oalist‘ ...)
| |
| ⊢ ↑before(k;map(λx.x.1;ws))
| |
1 BY (BLemma ‘before_all_imp_before‘ ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. Q: |oal(a;b)| ⟶ ℙ
  ⊢ [] ∈ |oal(a;b)|
  |
  BY (BLemma ‘nil_in_oalist‘ ...)
*T oalist_ind  80.2 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀Q:(|a| × |b|) List ⟶ ℙ.
|    Q[[]]
|    ⇒ (∀ws:|oal(a;b)|
|          Q[ws] ⇒ (∀x:|a|. ∀y:|b|.  ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ Q[<x, y>::ws]))
|    ⇒ {∀ws:|oal(a;b)|. Q[ws]}
|
BY (Unfold ‘guard‘ 0 THEN RepD ...a)
|
1. a: LOSet
2. b: AbMon
3. Q: (|a| × |b|) List ⟶ ℙ
4. Q[[]]
5. ∀ws:|oal(a;b)|
     Q[ws] ⇒ (∀x:|a|. ∀y:|b|.  ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ Q[<x, y>::ws])
6. ws: |oal(a;b)|
⊢ Q[ws]
|
BY RankInd ⌈λx.||x||⌉ ⌈ℕ⌉ CompNatInd (-1) THENA Auto'
|
7. ∀w1:|oal(a;b)|. ||w1|| < ||ws|| ⇒ Q[w1]
|
BY (MoveToConclFor (BLemma ‘oalist_cases‘) 6 ...a)
|\
| 6. ∀w1:|oal(a;b)|. ||w1|| < ||[]|| ⇒ Q[w1]
| ⊢ Q[[]]
| |
1 BY Trivial
 \
  7. x: |a|
  8. y: |b|
  9. ↑before(x;map(λx.x.1;ws))
  10. ¬(y = e)
  11. ∀w1:|oal(a;b)|. ||w1|| < ||<x, y>::ws|| ⇒ Q[w1]
  ⊢ Q[<x, y>::ws]
  |
  BY (BHyp 5 ...)
  |
  ⊢ Q[ws]
```

```
  |
  BY (BHyp 11 ...')
*T oalist_ind_a  74.7 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀Q:|oal(a;b)| ⟶ ℙ.
|    Q[[]]
|    ⇒ (∀ws:|oal(a;b)|
|          Q[ws] ⇒ (∀x:|a|. ∀y:|b|.  ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ Q[<x, y>::ws]))
|    ⇒ {∀ws:|oal(a;b)|. Q[ws]}
|
BY (Unfold `guard` 0 THEN RepD ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. Q: |oal(a;b)| ⟶ ℙ
| 4. Q[[]]
| 5. ∀ws:|oal(a;b)|
|      Q[ws] ⇒ (∀x:|a|. ∀y:|b|.  ↑before(x;map(λx.x.1;ws)) ⇒ ¬(y = e) ⇒ Q[<x, y>::ws])
| 6. ws: |oal(a;b)|
| ⊢ Q[ws]
| |
1 BY RankInd ⌈λx.||x||⌉ ⌈ℕ⌉ CompNatInd (-1) THENA Auto'
| |
| 7. ∀w1:|oal(a;b)|. ||w1|| < ||ws|| ⇒ Q[w1]
| |
1 BY (MoveToConclFor (BLemma `oalist_cases_a`) 6 ...a)
| |\
| | 6. ∀w1:|oal(a;b)|. ||w1|| < ||[]|| ⇒ Q[w1]
| | ⊢ Q[[]]
| | |
1 2 BY Trivial
|  \
|   7. x: |a|
|   8. y: |b|
|   9. ↑before(x;map(λx.x.1;ws))
|   10. ¬(y = e)
|   11. ∀w1:|oal(a;b)|. ||w1|| < ||<x, y>::ws|| ⇒ Q[w1]
|   ⊢ Q[<x, y>::ws]
|   |
1   BY (BHyp 5 ...)
|   |
|   ⊢ Q[ws]
|   |
1   BY (BHyp 11 ...')
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. Q: |oal(a;b)| ⟶ ℙ
| 4. Q[[]]
| 5. ws: |oal(a;b)|
| 6. Q[ws]
| 7. x: |a|
| 8. y: |b|
| 9. ↑before(x;map(λx.x.1;ws))
| 10. ¬(y = e)
| ⊢ (<x, y>::ws) ∈ |oal(a;b)|
| |
1 BY (BLemma `cons_in_oalist` ...)
```

```
 \
  1. a: LOSet
  2. b: AbMon
  3. Q: |oal(a;b)| ⟶ ℙ
  ⊢ [] ∈ |oal(a;b)|
  |
  BY (BLemma ‘nil_in_oalist‘ ...)
*T list_pr_length_ind  20.0 sec.
⊢ ∀T:𝕌. ∀Q:T List ⟶ T List ⟶ ℙ.
|    (∀ps,qs:T List.
|       (∀us,vs:T List.  ||us|| + ||vs|| < ||ps|| + ||qs|| ⇒ Q[us;vs]) ⇒ Q[ps;qs])
|    ⇒ {∀ps,qs:T List.  Q[ps;qs]}
|
BY (UnivCD ...a)
|
1. T: 𝕌
2. Q: T List ⟶ T List ⟶ ℙ
3. ∀ps,qs:T List.  (∀us,vs:T List.  ||us|| + ||vs|| < ||ps|| + ||qs|| ⇒ Q[us;vs]) ⇒ Q[ps;qs]
⊢ ∀ps,qs:T List.  Q[ps;qs]
|
BY Assert ⌜∀rs:T List × T List. Q[rs.1;rs.2]⌝
|\
| ⊢ ∀rs:T List × T List. Q[rs.1;rs.2]
| |
1 BY D 0 THENM RankInd ⌜λrs.||rs.1|| + ||rs.2||⌝ ⌜ℕ⌝ CompNatInd (-1)
| |  THENA Auto’
| |
| 4. rs: T List × T List
| 5. ∀r1:T List × T List. ||r1.1|| + ||r1.2|| < ||rs.1|| + ||rs.2|| ⇒ Q[r1.1;r1.2]
| ⊢ Q[rs.1;rs.2]
| |
1 BY (BHyp 3 THENM RepD ...a)
| |
| 6. us: T List
| 7. vs: T List
| 8. ||us|| + ||vs|| < ||rs.1|| + ||rs.2||
| ⊢ Q[us;vs]
| |
1 BY New [‘ps’;‘qs’] (D 4)
| |
| 4. ps: T List
| 5. qs: T List
| 6. ∀r1:T List × T List. ||r1.1|| + ||r1.2|| < ||<ps, qs>.1|| + ||<ps, qs>.2|| ⇒ Q[r1.1;r1.2]
| 7. us: T List
| 8. vs: T List
| 9. ||us|| + ||vs|| < ||<ps, qs>.1|| + ||<ps, qs>.2||
| |
1 BY OnHyps [6;9] AbReduce
| |
| 6. ∀r1:T List × T List. ||r1.1|| + ||r1.2|| < ||ps|| + ||qs|| ⇒ Q[r1.1;r1.2]
| 9. ||us|| + ||vs|| < ||ps|| + ||qs||
| |
1 BY (With ⌜<us, vs>⌝ (D 6) ...a)
| |  THEN AbReduce (-1)
| |
| 6. us: T List
| 7. vs: T List
```

```
| 8. ||us|| + ||vs|| < ||ps|| + ||qs||
| 9. ||us|| + ||vs|| < ||ps|| + ||qs|| ⇒ Q[us;vs]
| |
1 BY (BHyp 9 ...)
 \
  4. ∀rs:T List × T List. Q[rs.1;rs.2]
  |
  BY (RepD THENM With ⌜<ps, qs>⌝ (D 4)
      THENM AbReduce (-1) ...)
*T oalist_pr_length_ind  99.3 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀Q:(|a| × |b|) List ⟶ (|a| × |b|) List ⟶ ℙ.
|    (∀ps,qs:|oal(a;b)|.
|       (∀us,vs:|oal(a;b)|.  ||us|| + ||vs|| < ||ps|| + ||qs|| ⇒ Q[us;vs]) ⇒ Q[ps;qs])
|    ⇒ {∀ps,qs:|oal(a;b)|.  Q[ps;qs]}
|
BY (RepD ...a)
|
1. a: LOSet
2. b: AbMon
3. Q: (|a| × |b|) List ⟶ (|a| × |b|) List ⟶ ℙ
4. ∀ps,qs:|oal(a;b)|.
     (∀us,vs:|oal(a;b)|.  ||us|| + ||vs|| < ||ps|| + ||qs|| ⇒ Q[us;vs]) ⇒ Q[ps;qs]
⊢ ∀ps,qs:|oal(a;b)|.  Q[ps;qs]
|
BY Assert ⌜∀rs:|oal(a;b)| × |oal(a;b)|. Q[rs.1;rs.2]⌝
|  THENM (SeqOnM [RepD ; InstHyp [⌜<ps, qs>⌝] (-3)
|                ;Reduce (-1) ; BHyp (-1)] ...)
|  THEN (RepD ...a)
|
5. rs: |oal(a;b)| × |oal(a;b)|
⊢ Q[rs.1;rs.2]
|
BY OnVar 'rs' (RankInd ⌜λrs.||rs.1|| + ||rs.2||⌝ ⌜ℕ⌝ CompNatInd)
|  THENA Auto'
|
6. ∀r1:|oal(a;b)| × |oal(a;b)|. ||r1.1|| + ||r1.2|| < ||rs.1|| + ||rs.2|| ⇒ Q[r1.1;r1.2]
|
BY (BHyp 4 THENM RepD ...a)
|
7. us: |oal(a;b)|
8. vs: |oal(a;b)|
9. ||us|| + ||vs|| < ||rs.1|| + ||rs.2||
⊢ Q[us;vs]
|
BY (With ⌜<us, vs>⌝ (D 6) ...a)
   THEN Reduce (-1) THEN (BHyp (-1) ...)
*C oal_fun_fin_sup_char
               ==============================
               CHARACTERIZATION OF OALISTS AS
               FUNCTIONS OF FINITE SUPPORT
               ==============================
               It seems far more elegant to prove the
               algebraic properties of oalists and functions
               over them by using this characterization, rather
               than by plowing through inductive proofs.
*D oal_dom_df  dom{<a:oset:*>,<b:mon:*>}(<ps:oal:*>)== oal_dom{}(<a>; <b>; <ps>)
               dom(<ps:oal:*>)== oal_dom{}(<a>; <b>; <ps>)
```

```
*C oal_dom_com The type arguments are needed here to help with
               type checking (e.g. consider dom([])
               and provide arguments for rhs of eval rw rules.
*A oal_dom                    dom(ps) ==  mk_mset(map(λz.z.1;ps))
*T oal_dom_wf  3.4 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps:(|a| × |b|) List.  dom(ps) ∈ MSet{a}
|
BY (Unfold `oal_dom` 0 ...)
*T oal_dom_wf2  11.5 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps:|oal(a;b)|.  dom(ps) ∈ FSet{a}
|
BY RepD THENM MemTypeCD THENW Auto
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. ps: |oal(a;b)|
| ⊢ dom(ps) ∈ MSet{a}
| |
1 BY (BLemma `oal_dom_wf` ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. ps: |oal(a;b)|
  ⊢ ∀x:|a|. x #∈ dom(ps) ≤ 1
  |
  BY Unfold `oal_dom` 0
  |
  ⊢ ∀x:|a|. x #∈ mk_mset(map(λz.z.1;ps)) ≤ 1
  |
  BY (D 0 THENM Unfolds ``mset_count mk_mset`` 0 ...a)
  |
  4. x: |a|
  ⊢ x #∈ map(λz.z.1;ps) ≤ 1
  |
  BY (AddCarProperties 3
      THENM BLemma `sd_ordered_count` ...)
*M oal_dom_eval
               let oal_dom_nilC =
                 MacroC `oal_dom_nilC`
                   (EvalC ``oal_dom mk_mset``)
                   ⌈dom([])⌉
                   (UnfoldC `null_mset`)
                   ⌈0{a}⌉
               ;;
               let oal_dom_cons_prC =
                 MacroC `oal_dom_cons_prC`
                   (EvalC ``oal_dom mk_mset``)
                   ⌈dom(<k, v>::ps)⌉
                   (EvalC ``oal_dom mk_mset mset_inj mset_sum``)
                   ⌈mset_inj{a}(k) + dom(ps)⌉
               ;;
               add_AbReduce_conv `oal_dom`
                 (oal_dom_nilC ORELSEC oal_dom_cons_prC)
               ;;
*M oal_dom_eval2              DCL: oal_dom_nil:  dom(00) ∼ 0{s}
*C lookup_com  The lookup function defines the `function of
               finite support' that oalists represent. Most
```

```
               properties of functions on oalists are most easily
               proven with the help of this function.
*D lookup_df   <as:as:E>[<k:k:*>]{<s:s:*>,<z:z:*>}== lookup{}(<s>; <z>; <k>; <as>)
               Parens ::Prec(postop):: <as:as:E>[<k:k:*>]== lookup{}(<s>; <z>; <k>; <as>)
*M lookup_ml   as[k]
               ==r case as of
                     [] => z
                     b::bs => let <bk,bv> = b in if bk =b k then bv else bs[k] fi
                   esac
*T lookup_wf  2.5 sec.
⊢ ∀a:PosetSig. ∀B:𝕌. ∀z:B. ∀k:|a|. ∀xs:(|a| × B) List.  xs[k] ∈ B
|
BY (RepD ...a) THEN ListInd (-1)
|\
| 1. a: PosetSig
| 2. B: 𝕌
| 3. z: B
| 4. k: |a|
| 5. xs: (|a| × B) List
| ⊢ [][k] ∈ B
| |
1 BY (RecEval ``lookup`` 0 ...)
 \
  1. a: PosetSig
  2. B: 𝕌
  3. z: B
  4. k: |a|
  5. xs: (|a| × B) List
  6. u: |a| × B
  7. v: (|a| × B) List
  8. v[k] ∈ B
  ⊢ (u::v)[k] ∈ B
  |
  BY (D 6 THEN RecEval ``lookup`` 0 ...)
*T comb_for_lookup_wf  0.9 sec.
⊢ (λa,B,z,k,xs,z1.xs[k]) ∈ a:PosetSig → B:𝕌 → z:B → k:|a| → xs:(|a| × B) List → ↓True → B
|
BY ProveOpCombTyping `lookup_wf`
*M lookup_eval let lookup_nilC =
                 MacroC `lookup_nilC`
                 (RecUnfoldTopC `lookup` ANDTHENC ReduceC)
                 ⌜[][k]⌝
                 IdC ⌜z⌝
               ;;
               let lookup_cons_prC =
                 MacroC `lookup_cons_prC`
                 (RecUnfoldTopC `lookup` ANDTHENC ReduceC)
                  ⌜(<a, b>::cs)[k]⌝
                  IdC ⌜if a =b k then b else cs[k] fi ⌝
               ;;
               add_AbReduce_conv `lookup`
                 (lookup_cons_prC ORELSEC lookup_nilC) ;;
               % would be interesting to see if ifthenelse makes this work
                 any faster than if matching used directly.
               %
               let lookup_evalC e t =
                 if is_term `lookup` t then
```

```
                   (lookup_nilC
                    ORELSEC
                    (ITECondC lookup_cons_prC (RelRST THEN Auto))
                   ) e t
                  else
                    failwith `lookup_evalC`
                ;;
*M lookup_oal_eval            DCL: lookup_oal_nil:  00[k] ~ z
*T lookup_fails  16.5 sec.
⊢ ∀a:DSet. ∀B:𝕌. ∀z:B. ∀k:|a|. ∀ps:(|a| × B) List.  ¬↑(k ∈b map(λx.x.1;ps)) ⇒ ps[k] = z
|
BY (CDToVarThen `ps` ListIndA
|   THENM Reduce 0 THENM RepD ...)
|
1. a: DSet
2. B: 𝕌
3. z: B
4. k: |a|
5. p: |a| × B
6. ps: (|a| × B) List
7. ¬↑(k ∈b map(λx.x.1;ps)) ⇒ ps[k] = z
8. ¬↑((p.1 =b k) ∨b(k ∈b map(λx.x.1;ps)))
⊢ (p::ps)[k] = z
|
BY (RW bool_to_propC 8
|   THENM RWH (LemmaC `not_over_or`) 8
|   THENM D 8 ...a)
|  THEN New [`kp`;`vp`] (D 5)
|  THEN Reduce 0
|
5. kp: |a|
6. vp: B
7. ps: (|a| × B) List
8. ¬↑(k ∈b map(λx.x.1;ps)) ⇒ ps[k] = z
9. ¬(<kp, vp>.1 = k)
10. ¬↑(k ∈b map(λx.x.1;ps))
⊢ if kp =b k then vp else ps[k] fi  = z
|
BY (Reduce 9 THENM SplitOnConclITE
|    ...)
|
9. ¬(kp = k)
11. ¬(kp = k)
⊢ ps[k] = z
|
BY (BHyp 8 ...)
*T lookup_non_zero  47.0 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k:|a|. ∀ps:|oal(a;b)|.  ¬(ps[k] = e) ⟺ ↑(k ∈b dom(ps))
|
BY (GenRepD ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. k: |a|
| 4. ps: |oal(a;b)|
| 5. ¬(ps[k] = e)
| ⊢ ↑(k ∈b dom(ps))
```

```
| |
1 BY (NegateConcl THENM D 5
| |   THENM BLemma `lookup_fails` ...a)
| |
| 5. ¬↑(k ∈b dom(ps))
| ⊢ ¬↑(k ∈b map(λx.x.1;ps))
| |
1 BY (RepUnfolds ``mset_mem oal_dom mk_mset`` 5 ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. k: |a|
  4. ps: |oal(a;b)|
  5. ↑(k ∈b dom(ps))
  ⊢ ¬(ps[k] = e)
  |
  BY D 4 THENM D 5
  |
  4. ps: |((a × b↓set) List)|
  5. ↑sd_ordered(map(λx.x.1;ps))
  6. ¬↑(e ∈b map(λx.x.2;ps))
  7. ↑(k ∈b dom(ps))
  |
  BY Reduce 4 THEN RepUnfolds ``mset_mem oal_dom mk_mset`` 7
  |  THENM Thin 5
  |
  4. ps: (|a| × |b|) List
  5. ¬↑(e ∈b map(λx.x.2;ps))
  6. ↑(k ∈b map(λz.z.1;ps))
  |
  BY (ListIndA 4 THENM Reduce 0 ...a)
  |\
  | ⊢ ¬False ⇒ False ⇒ ¬(e = e)
  | |
  1 BY Auto
   \
    4. p: |a| × |b|
    5. ps: (|a| × |b|) List
    6. ¬↑(e ∈b map(λx.x.2;ps)) ⇒ ↑(k ∈b map(λz.z.1;ps)) ⇒ ¬(ps[k] = e)
    ⊢ ¬↑((p.2 =b e) ∨b(e ∈b map(λx.x.2;ps)))
    |  ⇒ ↑((p.1 =b k) ∨b(k ∈b map(λz.z.1;ps)))
    |  ⇒ ¬((p::ps)[k] = e)
    |
    BY (RWH bool_to_propC 0
    |   THENM New [`kp`;`vp`] (D 4)
    |   THENM Reduce 0 ...a)
    |
    4. kp: |a|
    5. vp: |b|
    6. ps: (|a| × |b|) List
    7. ¬↑(e ∈b map(λx.x.2;ps)) ⇒ ↑(k ∈b map(λz.z.1;ps)) ⇒ ¬(ps[k] = e)
    ⊢ ¬(vp = e ∨ ↑(e ∈b map(λx.x.2;ps)))
    |  ⇒ kp = k ∨ ↑(k ∈b map(λz.z.1;ps))
    |  ⇒ ¬(if kp =b k then vp else ps[k] fi  = e)
    |
    BY (RepD THENM SplitOnConclITE  ...)
    |\
```

```
    | 8. ¬(vp = e ∨ ↑(e ∈b map(λx.x.2;ps)))
    | 9. kp = k ∨ ↑(k ∈b map(λz.z.1;ps))
    | 10. kp = k
    | ⊢ ¬(vp = e)
    | |
    1 BY (RWH (LemmaC `not_over_or`) 8 ...)
     \
      8. ¬(vp = e ∨ ↑(e ∈b map(λx.x.2;ps)))
      9. kp = k ∨ ↑(k ∈b map(λz.z.1;ps))
      10. ¬(kp = k)
      ⊢ ¬(ps[k] = e)
      |
      BY (D 9 THEN BHyp 7 ...)
      |
      9. ↑(k ∈b map(λz.z.1;ps))
      ⊢ ¬↑(e ∈b map(λx.x.2;ps))
      |
      BY (RWH (LemmaC `not_over_or`) 8
          THENM D 8 ...)
*T lookup_oal_eq_id  6.4 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k:|a|. ∀ps:|oal(a;b)|.  ¬↑(k ∈b dom(ps)) ⇒ ps[k] = e
|
BY (RepD THENM BLemma `lookup_fails` ...a)
|
1. a: LOSet
2. b: AbMon
3. k: |a|
4. ps: |oal(a;b)|
5. ¬↑(k ∈b dom(ps))
⊢ ¬↑(k ∈b map(λx.x.1;ps))
|
BY (RepUnfolds ``mset_mem oal_dom mk_mset`` 5 ...)
*T lookup_oal_cons  25.3 sec.
⊢ ∀a:LOSet. ∀b:OCMon. ∀k,kp:|a|. ∀vp:|b|. ∀ps:|oal(a;b)|.
|    ↑before(kp;map(λz.z.1;ps)) ⇒ (<kp, vp>::ps)[k] = (when kp =b k. vp) * ps[k]
|
BY (RepD ...a)
|
1. a: LOSet
2. b: OCMon
3. k: |a|
4. kp: |a|
5. vp: |b|
6. ps: |oal(a;b)|
7. ↑before(kp;map(λz.z.1;ps))
⊢ (<kp, vp>::ps)[k] = (when kp =b k. vp) * ps[k]
|
BY Unfold `mon_when` 0 THEN Reduce 0
|  THEN (SplitOnConclITE ...a)
|\
| 8. kp = k
| ⊢ vp = vp * ps[k]
| |
1 BY (RWH (RevHypC 8) 0
|     THENM RWH (LemmaC `lookup_before_start`) 0
|     THENM RW GrpNormC 0 ...)
 \
```

```
  8. ¬(kp = k)
  ⊢ ps[k] = e * ps[k]
  |
  BY (RW GrpNormC 0 ...)
*C lookup_before_start_com
               lookup_before_start_a was
               a lot easier to prove. Shows the benefit
               of using a 'stronger' notion of beforeness.
*T lookup_before_start  94.1 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k:|a|. ∀ps:|oal(a;b)|.  ↑before(k;map(λz.z.1;ps)) ⇒ ps[k] = e
|
BY (RepeatMFor 4 (D 0) ...a)
|
1. a: LOSet
2. b: AbMon
3. k: |a|
4. ps: |oal(a;b)|
⊢ ↑before(k;map(λz.z.1;ps)) ⇒ ps[k] = e
|
BY RankInd ⌜λps.||ps||⌝ ⌜ℕ⌝ CompNatInd 4 THENA Auto'
|
5. ∀p1:|oal(a;b)|. ||p1|| < ||ps|| ⇒ ↑before(k;map(λz.z.1;p1)) ⇒ p1[k] = e
|
BY MoveToConcl 4 THEN
|   (BLemma 'oalist_cases' THENM RepD ...a)
|\
| 4. ∀p1:|oal(a;b)|. ||p1|| < ||[]|| ⇒ ↑before(k;map(λz.z.1;p1)) ⇒ p1[k] = e
| 5. ↑before(k;map(λz.z.1;[]))
| ⊢ [][k] = e
| |
1 BY (AbReduce 0 ...)
|\
| 4. ∀p1:|oal(a;b)|. ||p1|| < ||[]|| ⇒ ↑before(k;map(λz.z.1;p1)) ⇒ p1[k] = e
| ⊢ map(λz.z.1;[]) ∈ |a| List
| |
1 BY % Such goals should be taken care of automatically,
|      but type inference for lemma fails in this case.
|    % (AbReduce 0 ...)
 \
  5. x: |a|
  6. y: |b|
  7. ↑before(x;map(λx.x.1;ps))
  8. ¬(y = e)
  9. ∀p1:|oal(a;b)|. ||p1|| < ||<x, y>::ps|| ⇒ ↑before(k;map(λz.z.1;p1)) ⇒ p1[k] = e
  10. ↑before(k;map(λz.z.1;<x, y>::ps))
  ⊢ (<x, y>::ps)[k] = e
  |
  BY (AbReduce 0 THEN SplitOnConclITE ...a)
  |\
  | 11. x = k
  | ⊢ y = e
  | |
  1 BY AbEval ``before`` 10
  | |  THEN (RW bool_to_propC 10 ...a)
  | |
  | 10. x <a k
  | |
```

```
  1 BY % Need to patch up RelRST tactic to get this%
  |    (FLemma `qoset_lt_irrefl` [10] ...)
   \
    11. ¬(x = k)
    ⊢ ps[k] = e
    |
    BY BHyp 9 THEN Auto'
    |
    ⊢ ↑before(k;map(λz.z.1;ps))
    |
    BY AbReduce 10
    |  THEN (RW bool_to_propC 10 ...a)
    |
    10. x <a k
    |
    BY (FLemma `before_trans` [7;10] ...)
*T lookup_before_start_a  22.5 sec.
⊢ ∀a:QOSet. ∀b:AbMon. ∀k:|a|. ∀ps:(|a| × |b|) List.
|    ↑(∀b k'(:|a|) ∈ map(λz.z.1;ps). k' <b k) ⇒ ps[k] = e
|
BY (RepD THENM OnVar `ps` ListInd ...a)
|\
| 1. a: QOSet
| 2. b: AbMon
| 3. k: |a|
| 4. ps: (|a| × |b|) List
| ⊢ ↑(∀b k'(:|a|) ∈ map(λz.z.1;[]). k' <b k) ⇒ [][k] = e
| |
1 BY (AbReduce 0 ...)
 \
  1. a: QOSet
  2. b: AbMon
  3. k: |a|
  4. ps: (|a| × |b|) List
  5. u: |a| × |b|
  6. v: (|a| × |b|) List
  7. ↑(∀b k'(:|a|) ∈ map(λz.z.1;v). k' <b k) ⇒ v[k] = e
  ⊢ ↑(∀b k'(:|a|) ∈ map(λz.z.1;u::v). k' <b k) ⇒ (u::v)[k] = e
  |
  BY D 5 THEN AbReduce 0
  |  THEN (D 0 THENM RW bool_to_propC (-1) ...a)
  |  THEN (SplitOnConclITE ...a)
  |\
  | 5. u1: |a|
  | 6. u2: |b|
  | 7. v: (|a| × |b|) List
  | 8. ↑(∀b k'(:|a|) ∈ map(λz.z.1;v). k' <b k) ⇒ v[k] = e
  | 9. u1 <a k ∧ ↑(∀b k'(:|a|) ∈ map(λz.z.1;v). k' <b k)
  | 10. u1 = k
  | ⊢ u2 = e
  | |
  1 BY (D 9 THEN RelRST ...)
   \
    5. u1: |a|
    6. u2: |b|
    7. v: (|a| × |b|) List
    8. ↑(∀b k'(:|a|) ∈ map(λz.z.1;v). k' <b k) ⇒ v[k] = e
```

```
    9. u1 <a k ∧ ↑(∀b k'(:|a|) ∈ map(λz.z.1;v). k' <b k)
    10. ¬(u1 = k)
    ⊢ v[k] = e
    |
    BY (BHyp 8 ...)
*T lookups_same  219.6 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs:|oal(a;b)|.  (∀u:|a|. ps[u] = qs[u]) ⇒ ps = qs
|
BY (RepeatMFor 2 (D 0) ...a)
|  THEN (BLemma 'oalist_ind' THENM RepD ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. qs: |oal(a;b)|
| 4. ∀u:|a|. [][u] = qs[u]
| ⊢ [] = qs
| |
1 BY (OnVar 'qs' (MoveToConclFor (BLemma 'oalist_cases')) ...a)
| |\
| | 3. ∀u:|a|. [][u] = [][u]
| | ⊢ [] = []
| | |
1 2 BY Auto
|  \
|   4. x: |a|
|   5. y: |b|
|   6. ↑before(x;map(λx.x.1;qs))
|   7. ¬(y = e)
|   8. ∀u:|a|. [][u] = (<x, y>::qs)[u]
|   ⊢ [] = (<x, y>::qs)
|   |
1   BY (InstHyp [⌈x⌉] 8
|   |   THENM AbReduce 9 ...a)
|   |
|   9. e = if x =b x then y else qs[x] fi
|   |
1   BY (MoveToConclFor SplitOnConclITE 9 ...a)
|   |\
|   | 9. x = x
|   | 10. e = y
|   | |
1   2 BY (InvertRel 10 ...)
|    \
|     9. ¬(x = x)
|     10. e = qs[x]
|     |
1     BY (D 9 ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. ps: |oal(a;b)|
  4. ∀qs:|oal(a;b)|. (∀u:|a|. ps[u] = qs[u]) ⇒ ps = qs
  5. x: |a|
  6. y: |b|
  7. ↑before(x;map(λx.x.1;ps))
  8. ¬(y = e)
  9. qs: |oal(a;b)|
```

```
10. ∀u:|a|. (<x, y>::ps)[u] = qs[u]
⊢ (<x, y>::ps) = qs
|
BY (OnVar ‘qs’ (MoveToConclFor (BLemma ‘oalist_cases‘)) ...a)
|\
| 9. ∀u:|a|. (<x, y>::ps)[u] = [][u]
| ⊢ (<x, y>::ps) = []
| |
1 BY (InstHyp [⌈x⌉] 9 THENM AbReduce 10 ...a)
| |
| 10. if x =b x then y else ps[x] fi  = e
| |
1 BY (MoveToConclFor SplitOnConclITE 10 ...a)
| |\
| | 10. x = x
| | 11. y = e
| | |
1 2 BY Trivial
|  \
|   10. ¬(x = x)
|   11. ps[x] = e
|   |
1   BY (D 10 ...)
 \
  10. x1: |a|
  11. y1: |b|
  12. ↑before(x1;map(λx.x.1;qs))
  13. ¬(y1 = e)
  14. ∀u:|a|. (<x, y>::ps)[u] = (<x1, y1>::qs)[u]
  ⊢ (<x, y>::ps) = (<x1, y1>::qs)
  |
  BY (InstLemma ‘loset_trichot‘ [⌈a⌉;⌈x⌉;⌈x1⌉]
  |   THENM GenExRepD ...a)
  |\
  | 15. x <a x1
  | |
  1 BY (InstHyp [⌈x1⌉] 14
  | |   THENM RWH lookup_evalC (-1) ...a)
  | |
  | 16. ps[x1] = y1
  | |
  1 BY (RWH (LemmaC ‘lookup_before_start‘) (-1) THENM RelRST ...a)
  | |
  | ⊢ ↑before(x1;map(λz.z.1;ps))
  | |
  1 BY % Should fix RelRST to recognize such ‘heterogenous’
  |      transitivities %
  |    (FLemma ‘before_trans‘ [7;15] ...)
  |\
  | 15. x = x1
  | |
  1 BY EqCD
  | |\
  | | ⊢ <x, y> = <x1, y1>
  | | |
  1 2 BY (InstHyp [⌈x⌉] 14 THENM RWH lookup_evalC (-1) ...a)
  | | |
```

```
    | | 16. y = y1
    | | |
    1 2 BY (EqCD ...)
    |  \
    |   ⊢ ps = qs
    |   |
    1   BY (SeqOnM [BHyp 4;D 0;InstHyp [⌜u⌝] 14] ...a)
    |   |
    |   16. u: |a|
    |   17. (<x, y>::ps)[u] = (<x1, y1>::qs)[u]
    |   ⊢ ps[u] = qs[u]
    |   |
    1   BY (Decide ⌜u = x⌝
    |   |   THENM RWH lookup_evalC 17 ...a)
    |   |\
    |   | 17. y = y1
    |   | 18. u = x
    |   | |
    1   2 BY % A bit of congruence closure would be nice here %
    |   |
    |   |    (RWH (HypC 18) 0 THENM RWN 2 (HypC 15) 0
    |   |     THENM RWH (LemmaC `lookup_before_start`) 0 ...)
    |    \
    |     17. ps[u] = qs[u]
    |     18. ¬(u = x)
    |     |
    1     BY Trivial
     \
      15. x1 <a x
      |
      BY (InstHyp [⌜x⌝] 14
      |   THENM RWH lookup_evalC (-1) ...a)
      |
      16. y = qs[x]
      |
      BY (RWH (LemmaC `lookup_before_start`) (-1) THENM RelRST ...a)
      |
      ⊢ ↑before(x;map(λz.z.1;qs))
      |
      BY % Should fix RelRST to recognize such `heterogenous'
           transitivities %
         (FLemma `before_trans` [12;15] ...)
*T lookups_same_a  14.6 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs:|oal(a;b)|.  (∀u:|a|. ps[u] = qs[u]) ⇒ ps = qs
|
BY (RepD THENM AbReduce 0 ...a)
|
1. a: LOSet
2. b: AbMon
3. ps: |oal(a;b)|
4. qs: |oal(a;b)|
5. ∀u:|a|. ps[u] = qs[u]
⊢ ps = qs
|
BY (AddCarProperties 3
    THENM EqTypeCD
    THENM BLemma `lookups_same` ...)
```

```
*T oal_equal_char  13.9 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs:|oal(a;b)|.  ps = qs ⟺ (∀u:|a|. ps[u] = qs[u])
|
BY (GenRepD ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. ps: |oal(a;b)|
| 4. qs: |oal(a;b)|
| 5. ps = qs
| 6. u: |a|
| ⊢ ps[u] = qs[u]
| |
1 BY (RWH (HypC 5) 0 ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. ps: |oal(a;b)|
  4. qs: |oal(a;b)|
  5. ∀u:|a|. ps[u] = qs[u]
  ⊢ ps = qs
  |
  BY (BLemma `lookups_same_a` ...)
*C oal_merge_com_1
               ====================
               OALIST MERGE FUNCTION
               ====================
*C oal_merge_com
               Comments on this definition and the wf goal:
               1. AbMonoid typing not necessary. However, the
                 RepSplitITE tactic calls bool_to_propC which in
                 turn invokes the lemma `assert_of_mon_eq` which has
                 the monoid assumption. Would be messier to try to disable
                 this action.
               2. The destructor style definition is necessary here. The
                 wf goal cannot be proven if a constructor style
                 definition is used. (The problem is that the left list
                 decomp rule doesn't do substitutions in the hyp list.)
               3. Arith reasoning needs patching up to get rid of
                  clumsy need for `pos_length` lemmas.
               4. Would HO matching take care of cases in wf lemma
                  where IH has to be explicitly instantiated?
*D oal_merge_df
               Parens ::Prec(inop)::
                 <ps:ps:L> ++<a:a:*>,<b:b:L> <qs:qs:E>
                 == oal_merge{}(<a>; <b>; <ps>; <qs>)
               Parens ::Prec(inop)::
                 <ps:ps:L> ++ <qs:qs:E>
                 == oal_merge{}(<a>; <b>; <ps>; <qs>)
*M oal_merge_ml
               ps ++ qs
               ==r if null(ps) then qs
                   if null(qs) then ps
                   if hd(qs).1 <b hd(ps).1 then hd(ps)::(tl(ps) ++ qs)
                   if hd(ps).1 <b hd(qs).1 then hd(qs)::(ps ++ tl(qs))
                   if (hd(ps).2 * hd(qs).2) =b e then tl(ps) ++ tl(qs)
                   else <hd(ps).1, hd(ps).2 * hd(qs).2>::(tl(ps) ++ tl(qs))
```

```
                  fi
*M oal_merge_eval
               let oal_merge_left_nilC =
                 MacroC ‘oal_merge_left_nilC‘
                 (RecUnfoldC ‘oal_merge‘ ANDTHENC ReduceC)
                 ⌜[] ++ qs⌝
                 IdC
                  ⌜qs⌝ ;;
               let oal_merge_right_nilC =
                 MacroC ‘oal_merge_right_nilC‘
                 (RecUnfoldC ‘oal_merge‘ ANDTHENC ReduceC)
                 ⌜(p::ps) ++ []⌝
                 IdC
                  ⌜p::ps⌝ ;;
               let oal_merge_consesC =
                 MacroC ‘oal_merge_consesC‘
                 (RecUnfoldC ‘oal_merge‘ ANDTHENC ReduceC)
                 ⌜(<kp, vp>::ps) ++ (<kq, vq>::qs)⌝
                 IdC
                  ⌜if kq <b kp then <kp, vp>::(ps ++ (<kq, vq>::qs))
                   if kp <b kq then <kq, vq>::((<kp, vp>::ps) ++ qs)
                   if (vp * vq) =b e then ps ++ qs
                   else <kp, vp * vq>::(ps ++ qs)
                   fi ⌝ ;;
               add_AbReduce_conv ‘oal_merge‘
                 (FirstC [oal_merge_left_nilC
                         ;oal_merge_right_nilC
                         ;oal_merge_consesC
                         ]) ;;
               % tries to make as much headway as poss on ifthenelses %
               let oal_merge_evalC e t =
                 if is_term ‘oal_merge‘ t then
                  (FirstC
                    [oal_merge_left_nilC
                    ;oal_merge_right_nilC
                    ;oal_merge_consesC
                     ANDTHENC RepeatC (ITECondC IdC (RelRST THEN Auto))
                    ]) e t
                 else
                   failwith ‘oal_merge_evalC‘ ;;
*T oal_merge_wf  139.0 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs:(|a| × |b|) List.  ps ++ qs ∈ (|a| × |b|) List
|
BY (RepeatMFor 2 (D 0) ...a)
|
1. a: LOSet
2. b: AbMon
⊢ ∀ps,qs:(|a| × |b|) List.  ps ++ qs ∈ (|a| × |b|) List
|
BY Assert ⌜∀rs:(|a| × |b|) List × (|a| × |b|) List
|           rs.1 ++ rs.2 ∈ (|a| × |b|) List⌝
|  THEN (RepD ...a)
|\
| 3. rs: (|a| × |b|) List × (|a| × |b|) List
| ⊢ rs.1 ++ rs.2 ∈ (|a| × |b|) List
| |
```

```
1 BY RankInd ⌜λr.||r.1|| + ||r.2||⌝ ⌜ℕ⌝ CompNatInd 3
| |  THENA Auto'
| |
| 4. ∀r1:(|a| × |b|) List × (|a| × |b|) List
|      ||r1.1|| + ||r1.2|| < ||rs.1|| + ||rs.2|| ⇒ r1.1 ++ r1.2 ∈ (|a| × |b|) List
| |
1 BY New [‘ps’;‘qs’] (D 3) THEN
| |  OnCls [0;-1] AbReduce
| |
| 3. ps: (|a| × |b|) List
| 4. qs: (|a| × |b|) List
| 5. ∀r1:(|a| × |b|) List × (|a| × |b|) List
|      ||r1.1|| + ||r1.2|| < ||ps|| + ||qs|| ⇒ r1.1 ++ r1.2 ∈ (|a| × |b|) List
| ⊢ ps ++ qs ∈ (|a| × |b|) List
| |
1 BY % BLemma here is ugly. Needed because of hd partiality %
| |
| |  RecCaseSplit ‘oal_merge‘ THENA
| |  (Repeat (BLemma ‘pos_length‘ ORELSE Auto))
| |\
| | 6. ps = []
| | ⊢ qs ∈ (|a| × |b|) List
| | |
1 2 BY Auto
| |\
| | 6. ¬(ps = [])
| | 7. qs = []
| | ⊢ ps ∈ (|a| × |b|) List
| | |
1 2 BY Auto
| |\
| | 6. ¬(ps = [])
| | 7. ¬(qs = [])
| | 8. hd(qs).1 <a hd(ps).1
| | ⊢ (hd(ps)::(tl(ps) ++ qs)) ∈ (|a| × |b|) List
| | |
1 2 BY MemCD
| | |\
| | | ⊢ hd(ps) ∈ |a| × |b|
| | | |
1 2 3 BY Repeat (BLemma ‘pos_length‘ ORELSE Auto)
| |  \
| |   ⊢ tl(ps) ++ qs ∈ (|a| × |b|) List
| |   |
1 2   BY % Can’t match Inductive Hyp unless redo it %
| |   |  (InstHyp [⌜<tl(ps), qs>⌝] 5 THENM AbReduce (-1) ...)
| |   |  THEN AbReduce 0
| |   |
| |   ⊢ ||tl(ps)|| + ||qs|| < ||ps|| + ||qs||
| |   |
1 2   BY % Again, have problem with getting arith info
| |        in right form %
| |
| |      FLemma ‘pos_length‘ [6] THEN Auto'
| |\
| | 6. ¬(ps = [])
| | 7. ¬(qs = [])
```

```
| | 8. ¬(hd(qs).1 <a hd(ps).1)
| | 9. hd(ps).1 <a hd(qs).1
| | ⊢ (hd(qs)::(ps ++ tl(qs))) ∈ (|a| × |b|) List
| | |
1 2 BY MemCD
| | |\
| | | ⊢ hd(qs) ∈ |a| × |b|
| | | |
1 2 3 BY Repeat (BLemma ‘pos_length‘ ORELSE Auto)
| |  \
| |   ⊢ ps ++ tl(qs) ∈ (|a| × |b|) List
| |   |
1 2   BY (InstHyp [⌜<ps, tl(qs)>⌝] 5 THENM AbReduce (-1) ...)
| |      THEN AbReduce 0 THEN FLemma ‘pos_length‘ [7] THEN Auto’
| |\
| | 6. ¬(ps = [])
| | 7. ¬(qs = [])
| | 8. ¬(hd(qs).1 <a hd(ps).1)
| | 9. ¬(hd(ps).1 <a hd(qs).1)
| | 10. hd(ps).2 * hd(qs).2 = e
| | ⊢ tl(ps) ++ tl(qs) ∈ (|a| × |b|) List
| | |
1 2 BY (InstHyp [⌜<tl(ps), tl(qs)>⌝] 5 THENM AbReduce (-1) ...)
| |    THEN AbReduce 0
| |    THEN OnMHyps [6;7] (\i.FLemma ‘pos_length‘ [i]) THEN Auto’
|  \
|   6. ¬(ps = [])
|   7. ¬(qs = [])
|   8. ¬(hd(qs).1 <a hd(ps).1)
|   9. ¬(hd(ps).1 <a hd(qs).1)
|   10. ¬(hd(ps).2 * hd(qs).2 = e)
|   ⊢ (<hd(ps).1, hd(ps).2 * hd(qs).2>::(tl(ps) ++ tl(qs))) ∈ (|a| × |b|) List
|   |
1   BY MemCD
|   |\
|   | ⊢ <hd(ps).1, hd(ps).2 * hd(qs).2> ∈ |a| × |b|
|   | |
1   2 BY Repeat (BLemma ‘pos_length‘ ORELSE Auto)
|    \
|     ⊢ tl(ps) ++ tl(qs) ∈ (|a| × |b|) List
|     |
1     BY (InstHyp [⌜<tl(ps), tl(qs)>⌝] 5 THENM AbReduce (-1) ...)
|        THEN AbReduce 0
|        THEN OnMHyps [6;7] (\i.FLemma ‘pos_length‘ [i]) THEN Auto’
 \
  3. ∀rs:(|a| × |b|) List × (|a| × |b|) List. rs.1 ++ rs.2 ∈ (|a| × |b|) List
  4. ps: (|a| × |b|) List
  5. qs: (|a| × |b|) List
  ⊢ ps ++ qs ∈ (|a| × |b|) List
  |
  BY (InstHyp [⌜<ps, qs>⌝] 3
      THENM AbReduce (-1) ...)
*T oal_merge_dom_pred  155.3 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀Q:|a| ⟶ 𝔹. ∀ps,qs:(|a| × |b|) List.
|    ↑(∀b x(:|a|) ∈ map(λx.x.1;ps). Q[x])
|    ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;qs). Q[x])
|    ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;ps ++ qs). Q[x])
```

```
|
BY (RepeatMFor 3 (D 0)
|   THENM BLemma ‘list_pr_length_ind‘
|   THENM D 0 THENM D 0 ...a)
|
1. a: LOSet
2. b: AbMon
3. Q: |a| ⟶ 𝔹
4. ps: (|a| × |b|) List
5. qs: (|a| × |b|) List
⊢ (∀us,vs:(|a| × |b|) List.
|     ||us|| + ||vs|| < ||ps|| + ||qs||
|     ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;us). Q[x])
|     ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;vs). Q[x])
|     ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;us ++ vs). Q[x]))
|  ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;ps). Q[x])
|  ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;qs). Q[x])
|  ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;ps ++ qs). Q[x])
|
BY New [‘q’;‘qs\’’] (D 5)
|  THEN New [‘p’;‘ps\’’] (D 4)
|  THEN OnHyps [5;4] Thin % D needs fixing for lists %
|  THEN (AbReduce 0 ...)
|
4. q: |a| × |b|
5. qs’: (|a| × |b|) List
6. p: |a| × |b|
7. ps’: (|a| × |b|) List
8. ∀us,vs:(|a| × |b|) List.
     ||us|| + ||vs|| < (||ps’|| + 1) + ||qs’|| + 1
     ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;us). Q[x])
     ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;vs). Q[x])
     ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;us ++ vs). Q[x])
9. ↑(Q[p.1] ∧b (∀b x(:|a|) ∈ map(λx.x.1;ps’). Q[x]))
10. ↑(Q[q.1] ∧b (∀b x(:|a|) ∈ map(λx.x.1;qs’). Q[x]))
⊢ ↑(∀b x(:|a|) ∈ map(λx.x.1;(p::ps’) ++ (q::qs’)). Q[x])
|
BY New [‘pk’;‘pv’] (D 6)
|  THEN New [‘qk’;‘qv’] (D 4)
|  THEN ( OnMHyps [-1;-2]
|          (\i.AbReduce i THEN RW bool_to_propC i) ...a)
|  THEN RepD
|
4. qk: |a|
5. qv: |b|
6. qs’: (|a| × |b|) List
7. pk: |a|
8. pv: |b|
9. ps’: (|a| × |b|) List
10. ∀us,vs:(|a| × |b|) List.
      ||us|| + ||vs|| < (||ps’|| + 1) + ||qs’|| + 1
      ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;us). Q[x])
      ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;vs). Q[x])
      ⇒ ↑(∀b x(:|a|) ∈ map(λx.x.1;us ++ vs). Q[x])
11. ↑Q[pk]
12. ↑(∀b x(:|a|) ∈ map(λx.x.1;ps’). Q[x])
13. ↑Q[qk]
```

```
14. ↑(∀b x(:|a|) ∈ map(λx.x.1;qs'). Q[x])
⊢ ↑(∀b x(:|a|) ∈ map(λx.x.1;(<pk, pv>::ps') ++ (<qk, qv>::qs')). Q[x])
|
BY AbReduce 0
|  THEN (RepeatM SplitOnConclITE ...a)
|\
| 15. qk <a pk
| ⊢ ↑(∀b x(:|a|) ∈ map(λx.x.1;<pk, pv>::(ps' ++ (<qk, qv>::qs'))). Q[x])
| |
1 BY Repeat (First
|     [Progress (AbReduce 0)
|     ;Progress (RW bool_to_propC 0 ...a)
|     ;Progress Auto
|     ;BHyp 10])
|\
| 15. ¬(qk <a pk)
| 16. pk <a qk
| ⊢ ↑(∀b x(:|a|) ∈ map(λx.x.1;<qk, qv>::((<pk, pv>::ps') ++ qs')). Q[x])
| |
1 BY Repeat (First
|     [Progress (AbReduce 0)
|     ;Progress (RW bool_to_propC 0 ...a)
|     ;Progress Auto
|     ;BHyp 10])
|\
| 15. ¬(qk <a pk)
| 16. ¬(pk <a qk)
| 17. pv * qv = e
| ⊢ ↑(∀b x(:|a|) ∈ map(λx.x.1;ps' ++ qs'). Q[x])
| |
1 BY Repeat (First
|     [Progress (AbReduce 0)
|     ;Progress (RW bool_to_propC 0 ...a)
|     ;Progress Auto
|     ;BHyp 10])
 \
  15. ¬(qk <a pk)
  16. ¬(pk <a qk)
  17. ¬(pv * qv = e)
  ⊢ ↑(∀b x(:|a|) ∈ map(λx.x.1;<pk, pv * qv>::(ps' ++ qs')). Q[x])
  |
  BY Repeat (First
      [Progress (AbReduce 0)
      ;Progress (RW bool_to_propC 0 ...a)
      ;Progress Auto
      ;BHyp 10])
*T oal_dom_merge  32.2 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs:|oal(a;b)|.  ↑(dom(ps ++ qs) ⊆b dom(ps) ∪ dom(qs))
|
BY (RepD ...a)
|
1. a: LOSet
2. b: AbMon
3. ps: |oal(a;b)|
4. qs: |oal(a;b)|
⊢ ↑(dom(ps ++ qs) ⊆b dom(ps) ∪ dom(qs))
|
```

```
BY (BLemma ‘mem_bsubmset‘ THENM RepD ...a)
|
5. x: |a|
6. ↑(x ∈b dom(ps ++ qs))
⊢ ↑(x ∈b dom(ps) ∪ dom(qs))
|
BY (RWH (LemmaC ‘fset_mem_union‘) 0
|   THENM RW bool_to_propC 0 ...a)
|
⊢ ↑(x ∈b dom(ps)) ∨ ↑(x ∈b dom(qs))
|
BY (Negate 0 THENM Negate 6
|   THENM RWH (LemmaC ‘not_over_or‘) 6 ...a)
|
6. ¬↑(x ∈b dom(ps)) ∧ ¬↑(x ∈b dom(qs))
⊢ ¬↑(x ∈b dom(ps ++ qs))
|
BY % blow away mset stuff %
|  RenameVar ‘u’ 5
|  THEN OnCls [0;6] (RepUnfolds ‘‘oal_dom mk_mset mset_mem mem‘‘)
|  THEN OnCls [0;6] (Fold ‘bexists‘)
|
5. u: |a|
6. ¬↑(∃bx(:|a|) ∈ map(λz.z.1;ps). x =b u) ∧ ¬↑(∃bx(:|a|) ∈ map(λz.z.1;qs). x =b u)
⊢ ¬↑(∃bx(:|a|) ∈ map(λz.z.1;ps ++ qs). x =b u)
|
BY (OnMCls [0;6] (RW (SweepDnC
|    (LemmaC ‘bnot_thru_exists‘
|     ORELSEC RevLemmaC ‘assert_of_bnot‘))) ...a)
|
6. ↑(∀bx(:|a|) ∈ map(λz.z.1;ps). ¬b(x =b u)) ∧ ↑(∀bx(:|a|) ∈ map(λz.z.1;qs). ¬b(x =b u))
⊢ ↑(∀bx(:|a|) ∈ map(λz.z.1;ps ++ qs). ¬b(x =b u))
|
BY % Finally, we can apply this lemma %
   (BLemma ‘oal_merge_dom_pred‘ ...)
*C oal_merge_sd_ordered_com
              Notes on proof:
              1. Includes a couple of examples of monotonicity reasoning.
              2. Should assert(ball...) be taken care of by bool_to_propC?
              3. Induction delicate, because have to keep around some unreduced
                 sd_ordered predicates.
                 If unrolling of recursive definitions were to be driven by
                 destructors (such as of oal_merge by hd and tl) then automation
                 would be more straightforward. I guess this happens in NQTHM.
*T oal_merge_sd_ordered  233.9 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs:(|a| × |b|) List.
|    ↑sd_ordered(map(λx.x.1;ps))
|    ⇒ ↑sd_ordered(map(λx.x.1;qs))
|    ⇒ ↑sd_ordered(map(λx.x.1;ps ++ qs))
|
BY (RepeatMFor 2 (D 0)
|   THENM BLemma ‘list_pr_length_ind‘
|   THENM D 0 THENM D 0 ...a)
|
1. a: LOSet
2. b: AbMon
3. ps: (|a| × |b|) List
```

```
4. qs: (|a| × |b|) List
⊢ (∀us,vs:(|a| × |b|) List.
|     ||us|| + ||vs|| < ||ps|| + ||qs||
|     ⇒ ↑sd_ordered(map(λx.x.1;us))
|     ⇒ ↑sd_ordered(map(λx.x.1;vs))
|     ⇒ ↑sd_ordered(map(λx.x.1;us ++ vs)))
|  ⇒ ↑sd_ordered(map(λx.x.1;ps))
|  ⇒ ↑sd_ordered(map(λx.x.1;qs))
|  ⇒ ↑sd_ordered(map(λx.x.1;ps ++ qs))
|
BY New [‘q’;‘qs\’’] (D 4)
|  THEN New [‘p’;‘ps\’’] (D 3)
|  THEN OnHyps [4;3] Thin % D needs fixing for lists %
|  THEN (Complete (AbReduce 0 ...) ORELSE (RepD ...a))
|
3. q: |a| × |b|
4. qs’: (|a| × |b|) List
5. p: |a| × |b|
6. ps’: (|a| × |b|) List
7. ∀us,vs:(|a| × |b|) List.
     ||us|| + ||vs|| < ||p::ps’|| + ||q::qs’||
     ⇒ ↑sd_ordered(map(λx.x.1;us))
     ⇒ ↑sd_ordered(map(λx.x.1;vs))
     ⇒ ↑sd_ordered(map(λx.x.1;us ++ vs))
8. ↑sd_ordered(map(λx.x.1;p::ps’))
9. ↑sd_ordered(map(λx.x.1;q::qs’))
⊢ ↑sd_ordered(map(λx.x.1;(p::ps’) ++ (q::qs’)))
|
BY New [‘pk’;‘pv’] (D 5)
|  THEN New [‘qk’;‘qv’] (D 3)
|
3. qk: |a|
4. qv: |b|
5. qs’: (|a| × |b|) List
6. pk: |a|
7. pv: |b|
8. ps’: (|a| × |b|) List
9. ∀us,vs:(|a| × |b|) List.
     ||us|| + ||vs|| < ||<pk, pv>::ps’|| + ||<qk, qv>::qs’||
     ⇒ ↑sd_ordered(map(λx.x.1;us))
     ⇒ ↑sd_ordered(map(λx.x.1;vs))
     ⇒ ↑sd_ordered(map(λx.x.1;us ++ vs))
10. ↑sd_ordered(map(λx.x.1;<pk, pv>::ps’))
11. ↑sd_ordered(map(λx.x.1;<qk, qv>::qs’))
⊢ ↑sd_ordered(map(λx.x.1;(<pk, pv>::ps’) ++ (<qk, qv>::qs’)))
|
BY % Here switch to more tractable alternate characterization
|    for sd_ordered, but take care to preserve unreduced
|    sd_ordered hyps for satisfying later IH preconditions %
|  (OnMHyps [11;10] CopyToEnd
|   THEN OnMCls [0;-1;-2] (RWH (LemmaC ‘sd_ordered_char‘))  ...a)
|
12. ↑(HTFor{<𝔹,∧b>} v::vs ∈ map(λx.x.1;<qk, qv>::qs’). ∀bw(:|a|) ∈ vs. w <b v)
13. ↑(HTFor{<𝔹,∧b>} v::vs ∈ map(λx.x.1;<pk, pv>::ps’). ∀bw(:|a|) ∈ vs. w <b v)
⊢ ↑(HTFor{<𝔹,∧b>} v::vs ∈ map(λx.x.1;(<pk, pv>::ps’) ++ (<qk, qv>::qs’)). ∀bw(:|a|) ∈ vs
|                                                                          w <b v)
|
```

```
BY (OnMHyps [-1;-2]
|     (\i.AbReduce i THEN RW bool_to_propC i)
|   THENM RepD
|   THENM OnMHyps [-1;-3] (RWH (RevLemmaC `sd_ordered_char`)) ...a)
|
12. ↑(∀b w(:|a|) ∈ map(λx.x.1;qs'). w <b qk)
13. ↑sd_ordered(map(λx.x.1;qs'))
14. ↑(∀b w(:|a|) ∈ map(λx.x.1;ps'). w <b pk)
15. ↑sd_ordered(map(λx.x.1;ps'))
|
BY AbReduce 0
|  THEN (RepeatM SplitOnConclITE ...a)
|\
| 16. qk <a pk
| ⊢ ↑(HTFor{<ℙ,∧b>} v::vs ∈ map(λx.x.1;<pk, pv>::(ps' ++ (<qk, qv>::qs')))). ∀b w(:|a|) ∈ vs
| |                                                                                  w <b v)
| |
1 BY Repeat (First
| |   [Progress (RW bool_to_propC 0 ...a)
| |   ;Progress Auto
| |   ;(BLemma `oal_merge_dom_pred` ...a)
| |   ;(BHyp 9 ...a)
| |   ;Progress (AbReduce 0 THEN Try (RWH (RevLemmaC `sd_ordered_char`) 0))
| |   ])
| |
| ⊢ ↑(∀b x(:|a|) ∈ map(λx.x.1;qs'). x <b pk)
| |
1 BY (RWH (HypC 16) 12 ...)
|\
| 16. ¬(qk <a pk)
| 17. pk <a qk
| ⊢ ↑(HTFor{<ℙ,∧b>} v::vs ∈ map(λx.x.1;<qk, qv>::((<pk, pv>::ps') ++ qs'))). ∀b w(:|a|) ∈ vs
| |                                                                                  w <b v)
| |
1 BY Repeat (First
| |   [Progress (RW bool_to_propC 0 ...a)
| |   ;Progress Auto
| |   ;(BLemma `oal_merge_dom_pred` ...a)
| |   ;(BHyp 9 ...a)
| |   ;Progress (AbReduce 0 THEN Try (RWH (RevLemmaC `sd_ordered_char`) 0))
| |   ])
| |
| ⊢ ↑(∀b x(:|a|) ∈ map(λx.x.1;ps'). x <b qk)
| |
1 BY % An example of monotonicity reasoning %
| |
| |  (RWH (HypC 17) 14 ...a)
| |
| 14. ↑(∀b w(:|a|) ∈ map(λx.x.1;ps'). w <b qk)
| |
1 BY Trivial
|\
| 16. ¬(qk <a pk)
| 17. ¬(pk <a qk)
| 18. pv * qv = e
| ⊢ ↑(HTFor{<ℙ,∧b>} v::vs ∈ map(λx.x.1;ps' ++ qs'). ∀b w(:|a|) ∈ vs. w <b v)
| |
```

```
1 BY Repeat (First
|     [Progress (RW bool_to_propC 0 ...a)
|     ;Progress Auto
|     ;(BLemma ‘oal_merge_dom_pred‘ ...a)
|     ;(BHyp 9 ...a)
|     ;Progress (AbReduce 0 THEN Try (RWH (RevLemmaC ‘sd_ordered_char‘) 0))
|     ])
 \
  16. ¬(qk <a pk)
  17. ¬(pk <a qk)
  18. ¬(pv * qv = e)
  ⊢ ↑(HTFor{<𝔹,∧b>} v::vs ∈ map(λx.x.1;<pk, pv * qv>::(ps' ++ qs')). ∀bw(:|a|) ∈ vs. w <b v)
  |
  BY Repeat (First
  |   [Progress (RW bool_to_propC 0 ...a)
  |   ;Progress Auto
  |   ;(BLemma ‘oal_merge_dom_pred‘ ...a)
  |   ;(BHyp 9 ...a)
  |   ;Progress (AbReduce 0 THEN Try (RWH (RevLemmaC ‘sd_ordered_char‘) 0))
  |   ])
  |
  ⊢ ↑(∀bx(:|a|) ∈ map(λx.x.1;qs'). x <b pk)
  |
  BY (Assert ⌜pk = qk⌝ THENA RelRST ...a)
  |
  19. pk = qk
  |
  BY (RWH (HypC (-1)) 0 ...)
*T oal_merge_non_id_vals  299.3 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs:(|a| × |b|) List.
|    ¬↑(e ∈b map(λx.x.2;ps)) ⇒ ¬↑(e ∈b map(λx.x.2;qs)) ⇒ ¬↑(e ∈b map(λx.x.2;ps ++ qs))
|
BY (RepeatMFor 2 (D 0)
|   THENM BLemma ‘list_pr_length_ind‘
|   THENM D 0 THENM D 0 ...a)
|
1. a: LOSet
2. b: AbMon
3. ps: (|a| × |b|) List
4. qs: (|a| × |b|) List
⊢ (∀us,vs:(|a| × |b|) List.
|     ||us|| + ||vs|| < ||ps|| + ||qs||
|     ⇒ ¬↑(e ∈b map(λx.x.2;us))
|     ⇒ ¬↑(e ∈b map(λx.x.2;vs))
|     ⇒ ¬↑(e ∈b map(λx.x.2;us ++ vs)))
|  ⇒ ¬↑(e ∈b map(λx.x.2;ps))
|  ⇒ ¬↑(e ∈b map(λx.x.2;qs))
|  ⇒ ¬↑(e ∈b map(λx.x.2;ps ++ qs))
|
BY New [‘q';‘qs\''] (D 4)
|  THEN New [‘p';‘ps\''] (D 3)
|  THEN OnHyps [4;3] Thin % D needs fixing for lists %
|  THEN (AbReduce 0 ...)
|
3. q: |a| × |b|
4. qs': (|a| × |b|) List
5. p: |a| × |b|
```

```
6. ps': (|a| × |b|) List
7. ∀us,vs:(|a| × |b|) List.
     ||us|| + ||vs|| < (||ps'|| + 1) + ||qs'|| + 1
     ⇒ ¬↑(e ∈b map(λx.x.2;us))
     ⇒ ¬↑(e ∈b map(λx.x.2;vs))
     ⇒ ¬↑(e ∈b map(λx.x.2;us ++ vs))
8. ¬↑((p.2 =b e) ∨b(e ∈b map(λx.x.2;ps')))
9. ¬↑((q.2 =b e) ∨b(e ∈b map(λx.x.2;qs')))
⊢ ¬↑(e ∈b map(λx.x.2;(p::ps') ++ (q::qs')))
|
BY New [‘pk’;‘pv’] (D 5)
|  THEN New [‘qk’;‘qv’] (D 3)
|  % fiddly stuff! %
|  THEN ( OnMHyps [-1;-2]
|          (\i.
|            AbReduce i
|            THENM RWH (RevLemmaC ‘assert_of_bnot‘) i
|            THENM RWH (LemmaC ‘bnot_thru_bor‘) i
|            THENM RW bool_to_propC i
|          ) ...a)
|  THEN RepD
|
3. qk: |a|
4. qv: |b|
5. qs': (|a| × |b|) List
6. pk: |a|
7. pv: |b|
8. ps': (|a| × |b|) List
9. ∀us,vs:(|a| × |b|) List.
     ||us|| + ||vs|| < (||ps'|| + 1) + ||qs'|| + 1
     ⇒ ¬↑(e ∈b map(λx.x.2;us))
     ⇒ ¬↑(e ∈b map(λx.x.2;vs))
     ⇒ ¬↑(e ∈b map(λx.x.2;us ++ vs))
10. ¬(pv = e)
11. ¬↑(e ∈b map(λx.x.2;ps'))
12. ¬(qv = e)
13. ¬↑(e ∈b map(λx.x.2;qs'))
⊢ ¬↑(e ∈b map(λx.x.2;(<pk, pv>::ps') ++ (<qk, qv>::qs')))
|
BY AbReduce 0
|  THEN (RepeatM SplitOnConclITE ...a)
|\
| 14. qk <a pk
| ⊢ ¬↑(e ∈b map(λx.x.2;<pk, pv>::(ps' ++ (<qk, qv>::qs'))))
| |
1 BY Repeat (First
|     [Progress (AbReduce 0)
|     ;Progress (RW (RevLemmaC ‘assert_of_bnot‘
|                    ANDTHENC (SubC (LemmaC ‘bnot_thru_bor‘))) 0 ...a)
|     ;Progress (RW bool_to_propC 0 ...a)
|     ;Progress Auto
|     ;BHyp 9])
|\
| 14. ¬(qk <a pk)
| 15. pk <a qk
| ⊢ ¬↑(e ∈b map(λx.x.2;<qk, qv>::((<pk, pv>::ps') ++ qs')))
| |
```

```
1 BY Repeat (First
|     [Progress (AbReduce 0)
|     ;Progress (RW (RevLemmaC ‘assert_of_bnot‘
|                    ANDTHENC (SubC (LemmaC ‘bnot_thru_bor‘))) 0 ...a)
|     ;Progress (RW bool_to_propC 0 ...a)
|     ;Progress Auto
|     ;BHyp 9])
|\
| 14. ¬(qk <a pk)
| 15. ¬(pk <a qk)
| 16. pv * qv = e
| ⊢ ¬↑(e ∈b map(λx.x.2;ps’ ++ qs’))
| |
1 BY Repeat (First
|     [Progress (AbReduce 0)
|     ;Progress (RW (RevLemmaC ‘assert_of_bnot‘
|                    ANDTHENC (SubC (LemmaC ‘bnot_thru_bor‘))) 0 ...a)
|     ;Progress (RW bool_to_propC 0 ...a)
|     ;Progress Auto
|     ;BHyp 9])
 \
  14. ¬(qk <a pk)
  15. ¬(pk <a qk)
  16. ¬(pv * qv = e)
  ⊢ ¬↑(e ∈b map(λx.x.2;<pk, pv * qv>::(ps’ ++ qs’)))
  |
  BY Repeat (First
      [Progress (AbReduce 0)
      ;Progress (RW (RevLemmaC ‘assert_of_bnot‘
                     ANDTHENC (SubC (LemmaC ‘bnot_thru_bor‘))) 0 ...a)
      ;Progress (RW bool_to_propC 0 ...a)
      ;Progress Auto
      ;BHyp 9])
*T lookup_merge  417.2 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k:|a|. ∀ps,qs:|oal(a;b)|.  (ps ++ qs)[k] = ps[k] * qs[k]
|
BY (RepeatMFor 3 (D 0) ...a)
|
1. a: LOSet
2. b: AbMon
3. k: |a|
⊢ ∀ps,qs:|oal(a;b)|.  (ps ++ qs)[k] = ps[k] * qs[k]
|
BY (BLemma ‘oalist_pr_length_ind‘ ...a)
|
⊢ ∀ps,qs:|oal(a;b)|.
|    (∀us,vs:|oal(a;b)|.  ||us|| + ||vs|| < ||ps|| + ||qs|| ⇒ (us ++ vs)[k] = us[k] * vs[k])
|    ⇒ (ps ++ qs)[k] = ps[k] * qs[k]
|
BY % a little fiddly here, getting case splits right %
|  (RWAddr [2] GuardC 0
|   THEN BLemma ‘oalist_cases‘ THENML [Id;UnivCD]
|   THENM BLemma ‘oalist_cases‘ THENM RepD ...a)
|\
| 4. ∀us,vs:|oal(a;b)|.  ||us|| + ||vs|| < ||[]|| + ||[]|| ⇒ (us ++ vs)[k] = us[k] * vs[k]
| ⊢ ([] ++ [])[k] = [][k] * [][k]
| |
```

```
1 BY (Reduce 0 THEN RW MonNormC 0 ...)
|\
| 4. qs: |oal(a;b)|
| 5. x: |a|
| 6. y: |b|
| 7. ↑before(x;map(λx.x.1;qs))
| 8. ¬(y = e)
| 9. ∀us,vs:|oal(a;b)|.
|      ||us|| + ||vs|| < ||[]|| + ||<x, y>::qs|| ⇒ (us ++ vs)[k] = us[k] * vs[k]
| ⊢ ([] ++ (<x, y>::qs))[k] = [][k] * (<x, y>::qs)[k]
| |
1 BY RWH (oal_merge_left_nilC ORELSEC lookup_nilC) 0
|    THEN (RW MonNormC 0 ...)
|\
| 4. ps: |oal(a;b)|
| 5. x: |a|
| 6. y: |b|
| 7. ↑before(x;map(λx.x.1;ps))
| 8. ¬(y = e)
| 9. ∀us,vs:|oal(a;b)|.
|      ||us|| + ||vs|| < ||<x, y>::ps|| + ||[]|| ⇒ (us ++ vs)[k] = us[k] * vs[k]
| ⊢ ((<x, y>::ps) ++ [])[k] = (<x, y>::ps)[k] * [][k]
| |
1 BY RWH (lookup_nilC ORELSEC oal_merge_right_nilC) 0
|    THEN (RW MonNormC 0 ...)
 \
  4. ps: |oal(a;b)|
  5. x: |a|
  6. y: |b|
  7. ↑before(x;map(λx.x.1;ps))
  8. ¬(y = e)
  9. qs: |oal(a;b)|
  10. x1: |a|
  11. y1: |b|
  12. ↑before(x1;map(λx.x.1;qs))
  13. ¬(y1 = e)
  14. ∀us,vs:|oal(a;b)|.
        ||us|| + ||vs|| < ||<x, y>::ps|| + ||<x1, y1>::qs|| ⇒ (us ++ vs)[k] = us[k] * vs[k]
  ⊢ ((<x, y>::ps) ++ (<x1, y1>::qs))[k] = (<x, y>::ps)[k] * (<x1, y1>::qs)[k]
  |
  BY (RWH oal_merge_consesC 0
  |   THEN SplitOnConclITEs ...a)
  |\
  | 15. x1 <a x
  | ⊢ (<x, y>::(ps ++ (<x1, y1>::qs)))[k] = (<x, y>::ps)[k] * (<x1, y1>::qs)[k]
  | |
  1 BY RWNs [1;2] lookup_cons_prC 0
  | |  THEN (SplitOnConclITE  ...a)
  | |\
  | | 16. x = k
  | | ⊢ y = y * (<x1, y1>::qs)[k]
  | | |
  1 2 BY (RWH (LemmaC `lookup_before_start`) 0 ...a)
  | | |\
  | | | ⊢ (<x1, y1>::qs) ∈ |oal(a;b)|
  | | | |
  1 2 3 BY (BLemma `cons_pr_in_oalist` ...)
```

```
| | |\
| | | ⊢ ↑before(k;map(λz.z.1;<x1, y1>::qs))
| | | |
1 2 3 BY AbReduce 0 THEN (RW bool_to_propC 0
| | |                      THENM RelRST ...)
| |  \
| |   ⊢ y = y * e
| |   |
1 2   BY (RW MonNormC 0 ...)
|  \
|   16. ¬(x = k)
|   ⊢ (ps ++ (<x1, y1>::qs))[k] = ps[k] * (<x1, y1>::qs)[k]
|   |
1   BY (BHyp 14 ...a) THEN
|      Try (BLemma `cons_pr_in_oalist` ...)
|      THEN AbReduce 0 THEN Auto'
|\
| 15. ¬(x1 <a x)
| 16. x <a x1
| ⊢ (<x1, y1>::((<x, y>::ps) ++ qs))[k] = (<x, y>::ps)[k] * (<x1, y1>::qs)[k]
| |
1 BY RWNs [1;3] lookup_cons_prC 0
| |  THEN (SplitOnConclITE  ...a)
| |\
| | 17. x1 = k
| | ⊢ y1 = (<x, y>::ps)[k] * y1
| | |
1 2 BY (RWH (LemmaC `lookup_before_start`) 0 ...a)
| | |\
| | | ⊢ (<x, y>::ps) ∈ |oal(a;b)|
| | | |
1 2 3 BY (BLemma `cons_pr_in_oalist` ...)
| | |\
| | | ⊢ ↑before(k;map(λz.z.1;<x, y>::ps))
| | | |
1 2 3 BY AbReduce 0 THEN (RW bool_to_propC 0
| | |                      THENM RelRST ...)
| |  \
| |   ⊢ y1 = e * y1
| |   |
1 2   BY (RW MonNormC 0 ...)
|  \
|   17. ¬(x1 = k)
|   ⊢ ((<x, y>::ps) ++ qs)[k] = (<x, y>::ps)[k] * qs[k]
|   |
1   BY (BHyp 14 ...a) THEN
|      Try (BLemma `cons_pr_in_oalist` ...)
|      THEN AbReduce 0 THEN Auto'
|\
| 15. ¬(x1 <a x)
| 16. ¬(x <a x1)
| 17. y * y1 = e
| ⊢ (ps ++ qs)[k] = (<x, y>::ps)[k] * (<x1, y1>::qs)[k]
| |
1 BY (RWH lookup_cons_prC 0
| |   THEN RepeatM (SplitOnConclITE THENM Try RelRST) ...a)
| |\
```

```
| | 18. x = k
| | 19. x1 = k
| | ⊢ (ps ++ qs)[k] = y * y1
| | |
1 2 BY (RWH (LemmaC ‘lookup_before_start_a‘) 0 ...a)
| | |\
| | | ⊢ ↑(∀b k’(:|a|) ∈ map(λz.z.1;ps ++ qs). k’ <b k)
| | | |
1 2 3 BY (BLemma ‘oal_merge_dom_pred‘ ...a)
| | | |\
| | | | ⊢ ↑(∀b z(:|a|) ∈ map(λz.z.1;ps). z <b k)
| | | | |
1 2 3 4 BY % Ugh. Have to reconstruct sd_ordered %
| | | | |  Assert ⌜↑sd_ordered(map(λz.z.1;<x, y>::ps))⌝
| | | | |  THENA
| | | | |   (AddCarProperties 4
| | | | |    THENM AbReduce 0
| | | | |    THENM RW bool_to_propC 0 ...)
| | | | |
| | | | 20. ↑sd_ordered(map(λz.z.1;<x, y>::ps))
| | | | |
1 2 3 4 BY (RWH (LemmaC ‘sd_ordered_char‘) (-1)
| | | | |   THENM AbReduce (-1)
| | | | |   THENM RW bool_to_propC (-1) ...)
| | | | |
| | | | 20. ↑(∀b w(:|a|) ∈ map(λz.z.1;ps). w <b x)
| | | | 21. ↑(HTFor{<ℙ,∧b>} v::vs ∈ map(λz.z.1;ps). ∀b w(:|a|) ∈ vs. w <b v)
| | | | |
1 2 3 4 BY (RWH (RevHypC 18) 0 ...)
| | |  \
| | |   ⊢ ↑(∀b z(:|a|) ∈ map(λz.z.1;qs). z <b k)
| | |   |
1 2 3   BY % Ugh. Have to reconstruct sd_ordered %
| | |   |  Assert ⌜↑sd_ordered(map(λz.z.1;<x1, y1>::qs))⌝
| | |   |  THENA
| | |   |   (AddCarProperties 9
| | |   |    THENM AbReduce 0
| | |   |    THENM RW bool_to_propC 0 ...)
| | |   |
| | |   20. ↑sd_ordered(map(λz.z.1;<x1, y1>::qs))
| | |   |
1 2 3   BY (RWH (LemmaC ‘sd_ordered_char‘) (-1)
| | |       THENM AbReduce (-1)
| | |       THENM RW bool_to_propC (-1)
| | |       THENM RWH (RevHypC 19) 0 ...)
| |  \
| |   ⊢ e = y * y1
| |   |
1 2   BY Auto
|  \
|   18. ¬(x = k)
|   19. ¬(x1 = k)
|   ⊢ (ps ++ qs)[k] = ps[k] * qs[k]
|   |
1   BY BHyp 14 THEN Auto’
 \
  15. ¬(x1 <a x)
```

```
    16. ¬(x <a x1)
    17. ¬(y * y1 = e)
    ⊢ (<x, y * y1>::(ps ++ qs))[k] = (<x, y>::ps)[k] * (<x1, y1>::qs)[k]
    |
    BY (RWH lookup_cons_prC 0
    |   THEN RepeatM (SplitOnConclITE THENM Try RelRST) ...a)
    |
    18. ¬(x = k)
    19. ¬(x1 = k)
    ⊢ (ps ++ qs)[k] = ps[k] * qs[k]
    |
    BY BHyp 14 THEN Auto'
*T oal_merge_wf2  12.6 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs:|oal(a;b)|.  ps ++ qs ∈ |oal(a;b)|
|
BY (RepD ...a)
|
1. a: LOSet
2. b: AbMon
3. ps: |oal(a;b)|
4. qs: |oal(a;b)|
⊢ ps ++ qs ∈ |oal(a;b)|
|
BY (OnHyps [4;3] AddCarProperties
|   THENM RepD ...a)
|
4. ↑sd_ordered(map(λx.x.1;ps))
5. ¬↑(e ∈b map(λx.x.2;ps))
6. qs: |oal(a;b)|
7. ↑sd_ordered(map(λx.x.1;qs))
8. ¬↑(e ∈b map(λx.x.2;qs))
|
BY AbReduce 0 THEN (MemTypeCD ...)
|\
| ⊢ ↑sd_ordered(map(λx.x.1;ps ++ qs))
| |
1 BY (BLemma `oal_merge_sd_ordered` ...)
 \
  ⊢ ¬↑(e ∈b map(λx.x.2;ps ++ qs))
  |
  BY (BLemma `oal_merge_non_id_vals` ...)
*C oal_mon_com ========================
               OALIST MONOID DEFINITION
               ========================
*T oal_nil_ident_r  19.5 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps:|oal(a;b)|.  ps ++ 00 = ps
|
BY (D 0 THENM D 0 THENM BLemma `oalist_cases_a`
|   THENM RepD  ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| ⊢ [] ++ 00 = []
| |
1 BY RWH oal_merge_left_nilC 0
| |
| ⊢ 00 = []
```

```
| |
1 BY Unfold ‘oal_nil‘ 0
|    THEN Fold ‘member‘ 0
|    THEN (BLemma ‘nil_in_oalist‘ ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. ps: |oal(a;b)|
  4. x: |a|
  5. y: |b|
  6. ↑before(x;map(λx.x.1;ps))
  7. ¬(y = e)
  ⊢ (<x, y>::ps) ++ 00 = (<x, y>::ps)
  |
  BY Unfold ‘oal_nil‘ 0
     THEN RWH oal_merge_right_nilC 0
     THEN Fold ‘member‘ 0
     THEN (BLemma ‘cons_in_oalist‘ ...)
*T oal_nil_ident_l  4.3 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps:|oal(a;b)|.  00 ++ ps = ps
|
BY (RepD
    THENM Unfold ‘oal_nil‘ 0
    THENM RWH oal_merge_left_nilC 0 ...)
*T oal_merge_comm  11.0 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs:|oal(a;b)|.  ps ++ qs = qs ++ ps
|
BY (RepD THENM BLemma ‘lookups_same_a‘ ...a)
|
1. a: LOSet
2. b: AbMon
3. ps: |oal(a;b)|
4. qs: |oal(a;b)|
⊢ ∀u:|a|. (ps ++ qs)[u] = (qs ++ ps)[u]
|
BY (D 0
|   THENM RWH (LemmaC ‘lookup_merge‘) 0 ...a)
|
5. u: |a|
⊢ ps[u] * qs[u] = qs[u] * ps[u]
|
BY (RW AbMonNormC 0 ...)
*T oal_merge_assoc  15.3 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps,qs,rs:|oal(a;b)|.  ps ++ qs ++ rs = (ps ++ qs) ++ rs
|
BY (RepD
    THENM BLemma ‘lookups_same_a‘
    THENM D 0
    THENM RW (SweepDnC (LemmaC ‘lookup_merge‘)) 0
    THENM RW AbMonNormC 0 ...)
*D oal_mon_df                 oal_mon(<a:a:*>;<b:b:*>)== oal_mon{}(<a>; <b>)
*A oal_mon                    oal_mon(a;b) ==  <|oal(a;b)|, =b, λx,y.tt, λx,y.x ++ y, 00, λx.x>
*T oal_mon_wf  5.3 sec.
⊢ ∀a:LOSet. ∀b:AbMon.  oal_mon(a;b) ∈ AbMon
|
BY (Unfold ‘oal_mon‘ 0 ...)
|
```

```
1. a: LOSet
2. b: AbMon
⊢ <|oal(a;b)|, =b, λx,y.tt, λx,y.x ++ y, 00, λx.x> ∈ AbMon
|
BY (BLemma `mk_abmonoid` ...)
|\
| ⊢ IsEqFun(|oal(a;b)|;=b)
| |
1 BY (Assert ⌈oal(a;b) ∈ DSet⌉ ...a)
| |
| 3. oal(a;b) ∈ DSet
| |
1 BY (AddProperties 3 ...)
|\
| ⊢ Assoc(|oal(a;b)|;λx,y.x ++ y)
| |
1 BY Unfold `assoc` 0
|    THEN AbReduceIf (\e t.not is_term `set_car` t) 0
|    THEN (RepD THENM BLemma `oal_merge_assoc` ...a)
|\
| ⊢ Ident(|oal(a;b)|;λx,y.x ++ y;00)
| |
1 BY Unfold `ident` 0
| |  THEN AbReduceIf (\e t.not is_term `set_car` t) 0
| |  THEN (GenRepD ...a)
| |\
| | 3. x: |oal(a;b)|
| | ⊢ x ++ 00 = x
| | |
1 2 BY (BLemma `oal_nil_ident_r` ...)
|  \
|   3. x: |oal(a;b)|
|   ⊢ 00 ++ x = x
|   |
1   BY (BLemma `oal_nil_ident_l` ...)
 \
  ⊢ Comm(|oal(a;b)|;λx,y.x ++ y)
  |
  BY % AbReduceIf needed to prevent oalist getting opened up.
       Probably should fix projection computation funs to not
       open oalist %
     Unfold `comm` 0
     THEN AbReduceIf (\e t.not is_term `set_car` t) 0
     THEN (RepD THENM BLemma `oal_merge_comm` ...a)
*M oal_mon_ml  % Conversions for folding up oal_mon components %
               let oal_monC,rem_oal_monC =
                 let cprs =
                  map (\t,t'. DoubleMacroC `oal_monC` IdC t (ForceReduceC `5`) t')
                 [⌈|oal(s;g)|⌉,⌈|oal_mon(s;g)|⌉
                 ;⌈ps ++ qs⌉,⌈ps * qs⌉
                 ;⌈00⌉,⌈e⌉
                 ]
                  in
                    FirstC (map fst cprs),FirstC (map snd cprs)
               ;;
*C oal_inj_com ======================
               INJECTION INTO OALISTS
```

```
               ======================
*D oal_inj_df  inj{<a:a:*>,<b:b:*>}(<k:k:*>,<v:v:*>)== oal_inj{}(<a>; <b>; <k>; <v>)
               inj(<k:k:*>,<v:v:*>)== oal_inj{}(<a>; <b>; <k>; <v>)
*A oal_inj                      inj(k,v) ==  if v =b e then [] else <k, v>::[] fi
*T oal_inj_wf  20.7 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k:|a|. ∀v:|b|.  inj(k,v) ∈ |oal(a;b)|
|
BY (Unfold `oal_inj` 0
|   THEN RepD THENM SplitOnConclITE ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. k: |a|
| 4. v: |b|
| 5. v = e
| ⊢ [] ∈ |oal(a;b)|
| |
1 BY (BLemma `nil_in_oalist` ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. k: |a|
  4. v: |b|
  5. ¬(v = e)
  ⊢ (<k, v>::[]) ∈ |oal(a;b)|
  |
  BY (Reduce 0 THEN MemTypeCD ...)
  |\
  | ⊢ ↑sd_ordered(map(λx.x.1;<k, v>::[]))
  | |
  1 BY (Reduce 0 ...)
   \
    ⊢ ¬↑(e ∈b map(λx.x.2;<k, v>::[]))
    |
    BY (Reduce 0
    |   THENM RW bool_to_propC 0 ...)
    |
    ⊢ ¬(v = e ∨ False)
    |
    BY (D 0 THENM D (-1) ...)
*T comb_for_oal_inj_wf  1.4 sec.
⊢ (λa,b,k,v,z.inj(k,v)) ∈ a:LOSet ⟶ b:AbMon ⟶ k:|a| ⟶ v:|b| ⟶ ↓True ⟶ |oal(a;b)|
|
BY ProveOpCombTyping `oal_inj_wf`
*T lookup_oal_inj  16.3 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k,k':|a|. ∀v:|b|.  inj(k,v)[k'] = when k =b k'. v
|
BY (RepD THENM Unfold `oal_inj` 0
|   THENM SplitOnConclITE
|   THENM Reduce 0 ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. k: |a|
| 4. k': |a|
| 5. v: |b|
| 6. v = e
```

```
| ⊢ e = when k =_b k'. v
| |
1 BY (Unfold `mon_when` 0
|     THENM SplitOnConclITE ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. k: |a|
  4. k': |a|
  5. v: |b|
  6. ¬(v = e)
  ⊢ if k =_b k' then v else e fi  = when k =_b k'. v
  |
  BY (Fold `mon_when` 0 ...)
*T oal_dom_inj  11.9 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k:|a|. ∀v:|b|.
|    dom(inj(k,v)) = if v =_b e then 0{a} else mset_inj{a}(k) fi
|
BY (RepD ...a)
|
1. a: LOSet
2. b: AbMon
3. k: |a|
4. v: |b|
⊢ dom(inj(k,v)) = if v =_b e then 0{a} else mset_inj{a}(k) fi
|
BY  (WidenEqType
|    THENM Unfold `oal_inj` 0
|    THENM SplitOnConclITE ...a)
|\
| 5. v = e
| ⊢ dom([]) = 0{a}
| |
1 BY (Reduce 0 ...)
 \
  5. ¬(v = e)
  ⊢ dom(<k, v>::[]) = mset_inj{a}(k)
  |
  BY (Reduce 0 THENM RWW "mset_sum_ident_r" 0 ...)
*T oalist_fact  88.9 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps:|oal(a;b)|.  ps = msFor{oal_mon(a;b)} k' ∈ dom(ps). inj(k',ps[k'])
|
BY (RepD THENM BLemma `lookups_same_a` THENM D 0 ...a)
|
1. a: LOSet
2. b: AbMon
3. ps: |oal(a;b)|
4. u: |a|
⊢ ps[u] = (msFor{oal_mon(a;b)} k' ∈ dom(ps). inj(k',ps[k']))[u]
|
BY (MoveToConcl 3
|   THENM BLemma `oalist_ind_a`
|   THENM RepD THENM Reduce 0  ...a)
|\
| 3. u: |a|
| ⊢ e = 00[u]
| |
```

```
1 BY (Unfold ‘oal_nil‘ 0 THEN Reduce 0 ...)
 \
  3. u: |a|
  4. ps: |oal(a;b)|
  5. ps[u] = (msFor{oal_mon(a;b)} k’ ∈ dom(ps). inj(k’,ps[k’]))[u]
  6. x: |a|
  7. y: |b|
  8. ↑before(x;map(λx.x.1;ps))
  9. ¬(y = e)
  ⊢ if x =b u then y else ps[u] fi
  |  = (inj(x,if x =b x then y else ps[x] fi ) ++ (msFor{oal_mon(a;b)} k’ ∈ dom(ps)
  |                                                  inj(k’,if x =b k’
  |                                                 then y
  |                                                 else ps[k’]
  |                                                 fi )))[u]
  |
  BY (SplitOnNthConclITE 2 THENM Try RelRST ...a)
  |  THEN Thin (-1)
  |
  ⊢ if x =b u then y else ps[u] fi
  |  = (inj(x,y) ++ (msFor{oal_mon(a;b)} k’ ∈ dom(ps)
  |                    inj(k’,if x =b k’ then y else ps[k’] fi )))[u]
  |
  BY (RewriteWith [] ‘‘lookup_merge lookup_oal_inj‘‘ 0  ...a)
  |
  ⊢ if x =b u then y else ps[u] fi
  |  = (when x =b u. y)
  |    * (msFor{oal_mon(a;b)} k’ ∈ dom(ps). inj(k’,if x =b k’ then y else ps[k’] fi ))[u]
  |
  BY (RWN 2 (LemmaC ‘ite_rw_false‘) 0 ...a)
  |\
  | 10. x1: |a|
  | 11. ↑(x1 ∈b dom(ps))
  | ⊢ ¬↑(x =b x1)
  | |
  1 BY (RW bool_to_propC 0
  | |   THENM D 0
  | |   THENM FLemma ‘lookup_non_zero‘ [11] ...a)
  | |
  | 12. x = x1
  | 13. ¬(ps[x1] = e)
  | ⊢ False
  | |
  1 BY (RWH (LemmaC ‘lookup_before_start‘) 13 ...)
  | |\
  | | ⊢ ↑before(x1;map(λz.z.1;ps))
  | | |
  1 2 BY (RWH (RevHypC 12) 0 ...)
  |  \
  |   13. ¬(e = e)
  |   |
  1   BY (RelRST ...)
   \
    ⊢ if x =b u then y else ps[u] fi
    |  = (when x =b u. y) * (msFor{oal_mon(a;b)} k’ ∈ dom(ps). inj(k’,ps[k’]))[u]
    |
    BY (Unfold ‘mon_when‘ 0
```

```
    |   THEN SplitOnConclITE ...a)
    |\
    | 10. x = u
    | ⊢ y = y * (msFor{oal_mon(a;b)} k' ∈ dom(ps). inj(k',ps[k']))[u]
    | |
    1 BY % RWN needed because of false match against bound x in hyp 8
    |      This is due to rw env not being updated when descending
    |      through an untyped binding. Probably should use some dummy value... %
    |    (RWN 1 (HypC 10) 8
    |     THENM RWH (RevHypC 5) 0
    |     THENM RWH (LemmaC `lookup_before_start`) 0
    |     THENM RW MonNormC 0 ...)
     \
      10. ¬(x = u)
      ⊢ ps[u] = e * (msFor{oal_mon(a;b)} k' ∈ dom(ps). inj(k',ps[k']))[u]
      |
      BY (RW MonNormC 0 ...)
*C oal_neg_com ===============
               OALIST NEGATION
               ===============
               Definition & characterization of
               inverse function for oalist group.
*D oal_neg_df  Parens ::Prec(preop)::  --<a:a:*>,<b:b:*> <ps:ps:E>== oal_neg{}(<a>; <b>; <ps>)
               Parens ::Prec(preop)::  --<ps:ps:E>== oal_neg{}(<a>; <b>; <ps>)
*A oal_neg                    --ps ==  map(λkv.<kv.1, ∼ kv.2>;ps)
*T oal_neg_wf  1.8 sec.
⊢ ∀a:PosetSig. ∀b:GrpSig. ∀ps:(|a| × |b|) List.  --ps ∈ (|a| × |b|) List
|
BY (Unfold `oal_neg` 0 ...)
*M oal_neg_eval
               let oal_neg_nilC =
                 MacroC `oal_neg_nilC`
                 (EvalC ``oal_neg``)
                 ⌜--[]⌝
                 IdC
                  ⌜[]⌝
               ;;
               let oal_neg_cons_prC =
                 MacroC `oal_neg_cons_prC`
                  (EvalC ``oal_neg``)
                   ⌜--(<k, v>::ps)⌝
                  (UnfoldC `oal_neg`)
                   ⌜<k, ∼ v>::(--ps)⌝
               ;;
               add_AbReduce_conv `oal_neg`
                 (oal_neg_nilC ORELSEC oal_neg_cons_prC)
               ;;
*T oal_neg_keys_invar  2.7 sec.
⊢ ∀a:PosetSig. ∀b:GrpSig. ∀ps:(|a| × |b|) List.  map(λz.z.1;--ps) = map(λz.z.1;ps)
|
BY (RepD THENM Unfold `oal_neg` 0 ...a)
|
1. a: PosetSig
2. b: GrpSig
3. ps: (|a| × |b|) List
⊢ map(λz.z.1;map(λkv.<kv.1, ∼ kv.2>;ps)) = map(λz.z.1;ps)
|
```

```
BY (RWH (LemmaWithC [‘C’,⌜|a|⌝] ‘map_map‘) 0 ...a)
|
⊢ map((λz.z.1) o (λkv.<kv.1, ∼ kv.2>);ps) = map(λz.z.1;ps)
|
BY (Eval ‘‘compose‘‘ 0 ...)
*T oal_neg_sd_ordered  5.2 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀ps:(|a| × |b|) List.
|    ↑sd_ordered(map(λx.x.1;ps)) ⇒ ↑sd_ordered(map(λx.x.1;--ps))
|
BY (RepD
    THENM RWW "oal_neg_keys_invar" 0 ...)
*T oal_neg_non_id_vals  31.3 sec.
⊢ ∀a:LOSet. ∀b:AbGrp. ∀ps:(|a| × |b|) List.
|    ¬↑(e ∈b map(λx.x.2;ps)) ⇒ ¬↑(e ∈b map(λx.x.2;--ps))
|
BY (RepD THENM Unfold ‘oal_neg‘ 0 ...a)
|
1. a: LOSet
2. b: AbGrp
3. ps: (|a| × |b|) List
4. ¬↑(e ∈b map(λx.x.2;ps))
⊢ ¬↑(e ∈b map(λx.x.2;map(λkv.<kv.1, ∼ kv.2>;ps)))
|
BY (RWH (LemmaWithC [‘C’,⌜|b|⌝] ‘map_map‘) 0 ...a)
|
⊢ ¬↑(e ∈b map((λx.x.2) o (λkv.<kv.1, ∼ kv.2>);ps))
|
BY Eval ‘‘compose‘‘ 0
|
⊢ ¬↑(e ∈b map(λx.∼ x.2;ps))
|
BY (D 0 THENM D 4 ...a)
|
4. ↑(e ∈b map(λx.∼ x.2;ps))
⊢ ↑(e ∈b map(λx.x.2;ps))
|
BY (ListIndA 3 THEN Reduce 0
|   THENM RW bool_to_propC 0 THENM  D 0 ...a)
|\
| 3. False
| ⊢ False
| |
1 BY Auto
 \
  3. p: |a| × |b|
  4. ps: (|a| × |b|) List
  5. ↑(e ∈b map(λx.∼ x.2;ps)) ⇒ ↑(e ∈b map(λx.x.2;ps))
  6. ∼ p.2 = e ∨ ↑(e ∈b map(λx.∼ x.2;ps))
  ⊢ p.2 = e ∨ ↑(e ∈b map(λx.x.2;ps))
  |
  BY D 6
  |\
  | 6. ∼ p.2 = e
  | |
  1 BY (Sel 1 (D 0) ...a)
  | |
  | ⊢ p.2 = e
```

```
  | |
  1 BY (ApplyFunToHypEquands ⌜∼⌝ 6
  |     THENM RW GrpNormC 7 ...)
   \
    6. ↑(e ∈b map(λx.∼ x.2;ps))
    |
    BY (ProveProp ...)
*T oal_neg_wf2  11.3 sec.
⊢ ∀a:LOSet. ∀b:AbGrp. ∀ps:|oal(a;b)|.  --ps ∈ |oal(a;b)|
|
BY (RepD
|   THENM OnVar ‘ps’ AddCarProperties
|   THENM Force ‘8‘ (Reduce 0)
|   THENM MemTypeCD ...)
|\
| 1. a: LOSet
| 2. b: AbGrp
| 3. ps: |oal(a;b)|
| 4. ↑sd_ordered(map(λx.x.1;ps))
| 5. ¬↑(e ∈b map(λx.x.2;ps))
| ⊢ ↑sd_ordered(map(λx.x.1;--ps))
| |
1 BY (BLemma ‘oal_neg_sd_ordered‘ ...)
 \
  1. a: LOSet
  2. b: AbGrp
  3. ps: |oal(a;b)|
  4. ↑sd_ordered(map(λx.x.1;ps))
  5. ¬↑(e ∈b map(λx.x.2;ps))
  ⊢ ¬↑(e ∈b map(λx.x.2;--ps))
  |
  BY (BLemma ‘oal_neg_non_id_vals‘ ...)
*T lookup_oal_neg  10.7 sec.
⊢ ∀a:DSet. ∀b:IGroup. ∀k:|a|. ∀ps:(|a| × |b|) List.  (--ps)[k] = ∼ ps[k]
|
BY (CDToVarThen ‘ps’ ListIndA
|   THEN Reduce 0  ...a)
|\
| 1. a: DSet
| 2. b: IGroup
| 3. k: |a|
| ⊢ e = ∼ e
| |
1 BY (RW GrpNormC 0 ...)
 \
  1. a: DSet
  2. b: IGroup
  3. k: |a|
  4. p: |a| × |b|
  5. ps: (|a| × |b|) List
  6. (--ps)[k] = ∼ ps[k]
  ⊢ (--(p::ps))[k] = ∼ (p::ps)[k]
  |
  BY (D 4 THEN Reduce 0
      THEN SplitOnConclITE ...)
*T oal_dom_neg  15.8 sec.
⊢ ∀a:LOSet. ∀b:AbGrp. ∀ps:|oal(a;b)|.  dom(--ps) = dom(ps)
```

```
|
BY (RepD
|   THENM Assert ⌜dom(ps) ∈ FSet{a}⌝
|   THENM InvertRel 0
|   THENM EqTypeCD ...a)
|\
| 1. a: LOSet
| 2. b: AbGrp
| 3. ps: |oal(a;b)|
| 4. dom(ps) ∈ FSet{a}
| ⊢ dom(ps) = dom(--ps)
| |
1 BY (Thin 4 THEN Unfold `oal_dom` 0
| |   THENM RW mset_elimC 0 ...a)
| |
| ⊢ map(λz.z.1;ps) ≡(|a|) map(λz.z.1;--ps)
| |
1 BY (StrengthenRel ...a)
| |
| ⊢ map(λz.z.1;ps) = map(λz.z.1;--ps)
| |
1 BY (RWW "oal_neg_keys_invar" 0 ...)
 \
  1. a: LOSet
  2. b: AbGrp
  3. ps: |oal(a;b)|
  4. dom(ps) ∈ FSet{a}
  5. x: |a|
  ⊢ x #∈ dom(ps) ≤ 1
  |
  BY (AddProperties 4 THENM HypBackchain ...)
*T oal_neg_left_inv  14.0 sec.
⊢ ∀a:LOSet. ∀b:AbGrp. ∀ps:|oal(a;b)|.  --ps ++ ps = 00
|
BY (RepD THENM BLemma `lookups_same_a` THENM D 0 ...a)
|
1. a: LOSet
2. b: AbGrp
3. ps: |oal(a;b)|
4. u: |a|
⊢ (--ps ++ ps)[u] = 00[u]
|
BY (RWW "lookup_merge lookup_oal_neg" 0
|   THENM Eval ``oal_nil`` 0 ...a)
|
⊢ (∼ ps[u]) * ps[u] = e
|
BY (RW GrpNormC 0 ...)
*T oal_neg_right_inv  4.7 sec.
⊢ ∀a:LOSet. ∀b:AbGrp. ∀ps:|oal(a;b)|.  ps ++ --ps = 00
|
BY (RepD THENM RWH (LemmaC `oal_merge_comm`) 0
    THENM BLemma `oal_neg_left_inv` ...a)
*T oal_neg_eq_nil  7.3 sec.
⊢ ∀a:LOSet. ∀b:AbGrp. ∀ps:|oal(a;b)|.  --ps = 00 ⟺ ps = 00
|
BY (GenRepD ...a)
```

```
|\
| 1. a: LOSet
| 2. b: AbGrp
| 3. ps: |oal(a;b)|
| 4. --ps = 00
| ⊢ ps = 00
| |
1 BY (ApFunToHypEquands
| |     'qs' ⌈qs ++ ps⌉ ⌈|oal(a;b)|⌉ 4 ...a)
| |
| 5. --ps ++ ps = 00 ++ ps
| |
1 BY (RWW "oal_nil_ident_l oal_neg_left_inv" 5 ...)
 \
  1. a: LOSet
  2. b: AbGrp
  3. ps: |oal(a;b)|
  4. ps = 00
  ⊢ --ps = 00
  |
  BY (ApFunToHypEquands
  |     'qs' ⌈qs ++ --ps⌉ ⌈|oal(a;b)|⌉ 4 ...a)
  |
  5. ps ++ --ps = 00 ++ --ps
  |
  BY (RWW "oal_nil_ident_l oal_neg_right_inv" 5 ...)
*C oal_lv_and_lk_funs
               ==========================================
               LEADING KEY AND VALUE FUNCTIONS FOR OALISTS
               ==========================================
*C oal_null_com
               With most ps can infer s and g, but
               put args in, just in case get [] list
               before get chance for reduction.
               (e.g. from oal_cases)
*D oal_null_df null{<s:s:*>,<g:g:*>}(<ps:ps:*>)== oal_null{}(<s>; <g>; <ps>)
               null(<ps:ps:*>)== oal_null{}(<s>; <g>; <ps>)
*A oal_null                    null(ps) ==  null(ps)
*T oal_null_wf  4.5 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|.  null(ps) ∈ 𝔹
|
BY (Unfold 'oal_null' 0 ...)
*T assert_of_oal_null  15.3 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|.  ↑null(ps) ⟺ ps = 00
|
BY (RepD THENM Unfold 'oal_null' 0
|   THENM RWH (LemmaC 'assert_of_null') 0 ...a)
|
1. s: LOSet
2. g: AbMon
3. ps: |oal(s;g)|
⊢ ps = [] ⟺ ps = 00
|
BY (GenRepD ...a)
|\
| 4. ps = []
| ⊢ ps = 00
```

```
| |
1 BY (Reduce 4
|     THENM AddCarProperties 3
|     THENM Unfold `oal_nil` 0
|     THENM Force `8` (Reduce 0)
|     THENM EqTypeCD ...)
 \
  4. ps = 00
  ⊢ ps = []
  |
  BY (Unfold `oal_nil` 4
      THENM Inclusion 4 ...)
*M oal_null_ml add_reducible_ab `oal_null` ;;
               update_assert_elim_lemmas ``assert_of_oal_null``;;
*C oal_lk_com  lk = l(eading) k(ey)
               lv = l(eading) v(alue)
*D oal_lk_df                  lk(<ps:ps:*>)== oal_lk{}(<ps>)
*A oal_lk                     lk(ps) ==  hd(ps).1
*T oal_lk_wf  21.9 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|.  ¬(ps = 00) ⇒ lk(ps) ∈ |s|
|
BY (Unfold `oal_lk` 0 ...)
|
1. s: LOSet
2. g: AbMon
3. ps: |oal(s;g)|
4. ¬(ps = 00)
⊢ ||ps|| ≥ 1
|
BY (BLemma `length_of_not_nil` ...a)
|
⊢ ¬(ps = [])
|
BY (Unfold `oal_nil` 4 THEN D 0 THENM D 4 ...)
|
4. ps = []
⊢ ps = []
|
BY (AddCarProperties 3
    THENM Reduce 0
    THENM EqTypeCD
    THENM Inclusion (-1) ...)
*T oal_lk_in_dom  25.4 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|.  ¬(ps = 00) ⇒ ↑(lk(ps) ∈b dom(ps))
|
BY (D 0 THENM D 0
|   THENM BLemma `oalist_cases_a`
|   THENM Force `5` (Reduce 0)
|   THENM RepD ...a)
|\
| 1. s: LOSet
| 2. g: AbMon
| 3. ¬([] = 00)
| ⊢ False
| |
1 BY (D 3 THENM Unfold `oal_nil` 0
| |    ...)
```

```
| |
| ⊢ [] = []
| |
1 BY (BLemma ‘nil_in_oalist‘ ...)
|\
| 1. s: LOSet
| 2. g: AbMon
| ⊢ [] ∈ |oal(s;g)|
| |
1 BY (BLemma ‘nil_in_oalist‘ ...)
|\
| 1. s: LOSet
| 2. g: AbMon
| 3. ps: |oal(s;g)|
| 4. x: |s|
| 5. y: |g|
| 6. ↑before(x;map(λx.x.1;ps))
| 7. ¬(y = e)
| 8. ¬((<x, y>::ps) = 00)
| ⊢ ↑((x =b x) ∨b(x ∈b dom(ps)))
| |
1 BY (RW bool_to_propC 0
|     THENM Sel 1 (D 0) ...)
 \
  1. s: LOSet
  2. g: AbMon
  3. ps: |oal(s;g)|
  4. x: |s|
  5. y: |g|
  6. ↑before(x;map(λx.x.1;ps))
  7. ¬(y = e)
  ⊢ (<x, y>::ps) ∈ |oal(s;g)|
  |
  BY (BLemma ‘cons_in_oalist‘ ...)
*T oal_lk_bounds_dom  33.2 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀k:|s|. ∀ps:|oal(s;g)|.  ¬(ps = 00) ⇒ ↑(k ∈b dom(ps)) ⇒ k ≤ lk(ps)
|
BY (RepeatMFor 3 (D 0) THENM BLemma ‘oalist_cases_b‘ ...a)
|\
| 1. s: LOSet
| 2. g: AbMon
| 3. k: |s|
| ⊢ ¬([] = 00) ⇒ ↑(k ∈b dom([])) ⇒ k ≤ lk([])
| |
1 BY RWH (SimpleMacroC ‘*‘ ⌜[]⌝ ⌜00⌝ ‘‘oal_nil‘‘) 0
| |
| ⊢ ¬(00 = 00) ⇒ ↑(k ∈b dom(00)) ⇒ k ≤ lk(00)
| |
1 BY (D 0 THENM D (-1) ...)
 \
  1. s: LOSet
  2. g: AbMon
  3. k: |s|
  ⊢ ∀ps:|oal(s;g)|. ∀k1:|s|. ∀v:|g|.
  |    ↑(∀bx(:|s|) ∈ map(λz.z.1;ps). x <b k1)
  |    ⇒ ¬(v = e)
  |    ⇒ ¬((<k1, v>::ps) = 00)
```

```
  |    ⇒ ↑(k ∈b dom(<k1, v>::ps))
  |    ⇒ k ≤ lk(<k1, v>::ps)
  |
  BY (RepD ...a)
  |\
  | 4. ps: |oal(s;g)|
  | 5. k1: |s|
  | 6. v: |g|
  | 7. ↑(∀bx(:|s|) ∈ map(λz.z.1;ps). x <b k1)
  | 8. ¬(v = e)
  | 9. ¬((<k1, v>::ps) = 00)
  | 10. ↑(k ∈b dom(<k1, v>::ps))
  | ⊢ k ≤ lk(<k1, v>::ps)
  | |
  1 BY OnCls [10;0] Reduce
  | |
  | 10. ↑((k1 =b k) ∨b(k ∈b dom(ps)))
  | ⊢ k ≤ k1
  | |
  1 BY % correct mismatch here between mset and list stuff %
  | |  RepUnfolds ``mset_mem oal_dom mk_mset`` 10
  | |
  | 10. ↑((k1 =b k) ∨b(k ∈b map(λz.z.1;ps)))
  | |
  1 BY (RWH (LemmaC `ball_char`) 7
  | |   THENM OnMCls [7;10] (RW bool_to_propC) ...a)
  | |
  | 7. ∀x:|s|. ↑(x ∈b map(λz.z.1;ps)) ⇒ x <s k1
  | 10. k1 = k ∨ ↑(k ∈b map(λz.z.1;ps))
  | |
  1 BY (D 10 THENM Try RelRST ...a)
  | |
  | 10. ↑(k ∈b map(λz.z.1;ps))
  | |
  1 BY (FHyp 7 [10] THENM RelRST ...)
   \
    4. ps: |oal(s;g)|
    5. k1: |s|
    6. v: |g|
    7. ↑(∀bx(:|s|) ∈ map(λz.z.1;ps). x <b k1)
    8. ¬(v = e)
    ⊢ (<k1, v>::ps) ∈ |oal(s;g)|
    |
    BY (Backchain ``cons_in_oalist before_all_imp_before`` ...)
*M oal_lk_eval let oal_lk_cons_prC =
                 MacroC `oal_lk_cons_prC`
                   (EvalC ``oal_lk``)
                    ⌜lk(<k, v>::ps)⌝
                   IdC ⌜k⌝
               ;;
               add_AbReduce_conv `oal_lk`
                 oal_lk_cons_prC ;;
*D oal_lv_df                   lv(<ps:ps:*>)== oal_lv{}(<ps>)
*A oal_lv                      lv(ps) ==  hd(ps).2
*T oal_lv_wf  23.0 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|.  ¬(ps = 00) ⇒ lv(ps) ∈ |g|
|
```

```
BY (Unfold ‘oal_lv‘ 0 ...)
|
1. s: LOSet
2. g: AbMon
3. ps: |oal(s;g)|
4. ¬(ps = 00)
⊢ ||ps|| ≥ 1
|
BY (BLemma ‘length_of_not_nil‘ ...a)
|
⊢ ¬(ps = [])
|
BY (Unfold ‘oal_nil‘ 4 THEN D 0 THENM D 4 ...)
|
4. ps = []
⊢ ps = []
|
BY (AddCarProperties 3
    THENM Reduce 0
    THENM EqTypeCD
    THENM Inclusion (-1) ...)
*T oal_lv_nid  14.0 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|.  ¬(ps = 00) ⇒ ¬(lv(ps) = e)
|
BY (D 0 THENM D 0
|   THENM BLemma ‘oalist_cases_a‘ ...a)
|\
| 1. s: LOSet
| 2. g: AbMon
| ⊢ ¬([] = 00) ⇒ ¬(lv([]) = e)
| |
1 BY (RWH (SimpleMacroC ‘*‘ ⌜[]⌝ ⌜00⌝ ‘‘oal_nil‘‘) 0
|     THENM D 0 THENM D (-1) ...)
 \
  1. s: LOSet
  2. g: AbMon
  ⊢ ∀ps:|oal(s;g)|. ∀x:|s|. ∀y:|g|.
  |    ↑before(x;map(λx.x.1;ps)) ⇒ ¬(y = e) ⇒ ¬((<x, y>::ps) = 00) ⇒ ¬(lv(<x, y>::ps) = e)
  |
  BY (RepD THENM Reduce 0 ...a)
  |\
  | 3. ps: |oal(s;g)|
  | 4. x: |s|
  | 5. y: |g|
  | 6. ↑before(x;map(λx.x.1;ps))
  | 7. ¬(y = e)
  | 8. ¬((<x, y>::ps) = 00)
  | ⊢ ¬(y = e)
  | |
  1 BY Trivial
   \
    3. ps: |oal(s;g)|
    4. x: |s|
    5. y: |g|
    6. ↑before(x;map(λx.x.1;ps))
    7. ¬(y = e)
    ⊢ (<x, y>::ps) ∈ |oal(s;g)|
```

```
    |
    BY (BLemma ‘cons_in_oalist‘ ...)
*M oal_lv_eval let oal_lv_cons_prC =
                 MacroC ‘oal_lv_cons_prC‘
                   (EvalC ‘‘oal_lv‘‘)
                    ⌈lv(<k, v>::ps)⌉
                   IdC ⌈v⌉
               ;;
               add_AbReduce_conv ‘oal_lv‘
                 oal_lv_cons_prC ;;
*T oal_lk_merge_1  76.3 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps,qs:|oal(s;g)|.
|    ¬(ps = 00) ⇒ ¬(qs = 00) ⇒ ¬(ps ++ qs = 00) ⇒ lk(ps) <s lk(qs) ⇒ lk(ps ++ qs) = lk(qs)
|
BY (D 0 THENM D 0
|   THENM BLemma ‘oalist_cases_a‘ ...a)
|\
| 1. s: LOSet
| 2. g: AbMon
| ⊢ ∀qs:|oal(s;g)|
| |    ¬([] = 00) ⇒ ¬(qs = 00) ⇒ ¬([] ++ qs = 00) ⇒ lk([]) <s lk(qs) ⇒ lk([] ++ qs) = lk(qs)
| |
1 BY RWH (SimpleMacroC ‘*‘ ⌈[]⌉ ⌈00⌉ ‘‘oal_nil‘‘) 0
|    THEN (D 0 THENM D 0 THENM D (-1) ...)
 \
  1. s: LOSet
  2. g: AbMon
  ⊢ ∀ps:|oal(s;g)|. ∀x:|s|. ∀y:|g|.
  |    ↑before(x;map(λx.x.1;ps))
  |    ⇒ ¬(y = e)
  |    ⇒ (∀qs:|oal(s;g)|
  |          ¬((<x, y>::ps) = 00)
  |          ⇒ ¬(qs = 00)
  |          ⇒ ¬((<x, y>::ps) ++ qs = 00)
  |          ⇒ lk(<x, y>::ps) <s lk(qs)
  |          ⇒ lk((<x, y>::ps) ++ qs) = lk(qs))
  |
  BY (RepeatMFor 5 (D 0)
  |   THENM BLemma ‘oalist_cases_a‘ ...a)
  |  THEN Try (BLemma ‘cons_in_oalist‘ ...)
  |\
  | 3. ps: |oal(s;g)|
  | 4. x: |s|
  | 5. y: |g|
  | 6. ↑before(x;map(λx.x.1;ps))
  | 7. ¬(y = e)
  | ⊢ ¬((<x, y>::ps) = 00)
  | |  ⇒ ¬([] = 00)
  | |  ⇒ ¬((<x, y>::ps) ++ [] = 00)
  | |  ⇒ lk(<x, y>::ps) <s lk([])
  | |  ⇒ lk((<x, y>::ps) ++ []) = lk([])
  | |
  1 BY RWH (SimpleMacroC ‘*‘ ⌈[]⌉ ⌈00⌉ ‘‘oal_nil‘‘) 0
  |    THEN (D 0 THENM D 0 THENM D (-1) ...)
  |    THEN (BLemma ‘cons_in_oalist‘ ...)
   \
    3. ps: |oal(s;g)|
```

```
4. x: |s|
5. y: |g|
6. ↑before(x;map(λx.x.1;ps))
7. ¬(y = e)
⊢ ∀qs:|oal(s;g)|. ∀x1:|s|. ∀y1:|g|.
|    ↑before(x1;map(λx.x.1;qs))
|    ⇒ ¬(y1 = e)
|    ⇒ ¬((<x, y>::ps) = 00)
|    ⇒ ¬((<x1, y1>::qs) = 00)
|    ⇒ ¬((<x, y>::ps) ++ (<x1, y1>::qs) = 00)
|    ⇒ lk(<x, y>::ps) <s lk(<x1, y1>::qs)
|    ⇒ lk((<x, y>::ps) ++ (<x1, y1>::qs)) = lk(<x1, y1>::qs)
|
BY (RepeatMFor 8 (D 0) ...a)
|  THEN Try (BLemma ‘cons_in_oalist‘ ...)
|
8. qs: |oal(s;g)|
9. x1: |s|
10. y1: |g|
11. ↑before(x1;map(λx.x.1;qs))
12. ¬(y1 = e)
13. ¬((<x, y>::ps) = 00)
14. ¬((<x1, y1>::qs) = 00)
15. ¬((<x, y>::ps) ++ (<x1, y1>::qs) = 00)
⊢ lk(<x, y>::ps) <s lk(<x1, y1>::qs)
|  ⇒ lk((<x, y>::ps) ++ (<x1, y1>::qs)) = lk(<x1, y1>::qs)
|
BY (Reduce 0
|   THENM SplitOnConclITEs
|   THENM Reduce 0
|   THENM D 0 ...a)
|\
| 16. x1 <s x
| 17. x <s x1
| ⊢ x = x1
| |
1 BY (RelRST ...)
|\
| 16. ¬(x1 <s x)
| 17. x <s x1
| 18. x <s x1
| ⊢ x1 = x1
| |
1 BY (RelRST ...)
|\
| 16. ¬(x1 <s x)
| 17. ¬(x <s x1)
| 18. y * y1 = e
| 19. x <s x1
| ⊢ lk(ps ++ qs) = x1
| |
1 BY (RelRST ...)
 \
  16. ¬(x1 <s x)
  17. ¬(x <s x1)
  18. ¬(y * y1 = e)
  19. x <s x1
```

```
      ⊢ x = x1
      |
      BY (RelRST ...)
*T oal_lk_merge_2  81.1 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps,qs:|oal(s;g)|.
|    ¬(ps = 00)
|    ⇒ ¬(qs = 00)
|    ⇒ ¬(ps ++ qs = 00)
|    ⇒ lk(ps) = lk(qs)
|    ⇒ ¬(lv(ps) * lv(qs) = e)
|    ⇒ lk(ps ++ qs) = lk(qs)
|
BY (D 0 THENM D 0
|   THENM BLemma `oalist_cases_a` ...a)
|\
| 1. s: LOSet
| 2. g: AbMon
| ⊢ ∀qs:|oal(s;g)|
| |    ¬([] = 00)
| |    ⇒ ¬(qs = 00)
| |    ⇒ ¬([] ++ qs = 00)
| |    ⇒ lk([]) = lk(qs)
| |    ⇒ ¬(lv([]) * lv(qs) = e)
| |    ⇒ lk([] ++ qs) = lk(qs)
| |
1 BY RWH (SimpleMacroC `*` ⌜[]⌝ ⌜00⌝ ``oal_nil``) 0
|    THEN (D 0 THENM D 0 THENM D (-1) ...)
 \
  1. s: LOSet
  2. g: AbMon
  ⊢ ∀ps:|oal(s;g)|. ∀x:|s|. ∀y:|g|.
  |    ↑before(x;map(λx.x.1;ps))
  |    ⇒ ¬(y = e)
  |    ⇒ (∀qs:|oal(s;g)|
  |          ¬((<x, y>::ps) = 00)
  |          ⇒ ¬(qs = 00)
  |          ⇒ ¬((<x, y>::ps) ++ qs = 00)
  |          ⇒ lk(<x, y>::ps) = lk(qs)
  |          ⇒ ¬(lv(<x, y>::ps) * lv(qs) = e)
  |          ⇒ lk((<x, y>::ps) ++ qs) = lk(qs))
  |
  BY (RepeatMFor 5 (D 0)
  |   THENM BLemma `oalist_cases_a` ...a)
  |  THEN Try (BLemma `cons_in_oalist` ...)
  |\
  | 3. ps: |oal(s;g)|
  | 4. x: |s|
  | 5. y: |g|
  | 6. ↑before(x;map(λx.x.1;ps))
  | 7. ¬(y = e)
  | ⊢ ¬((<x, y>::ps) = 00)
  | |  ⇒ ¬([] = 00)
  | |  ⇒ ¬((<x, y>::ps) ++ [] = 00)
  | |  ⇒ lk(<x, y>::ps) = lk([])
  | |  ⇒ ¬(lv(<x, y>::ps) * lv([]) = e)
  | |  ⇒ lk((<x, y>::ps) ++ []) = lk([])
  | |
```

```
1 BY RWH (SimpleMacroC `*` ⌜[]⌝ ⌜00⌝ ``oal_nil``) 0
|    THEN (D 0 THENM D 0 THENM D (-1) ...)
|    THEN (BLemma `cons_in_oalist` ...)
 \
  3. ps: |oal(s;g)|
  4. x: |s|
  5. y: |g|
  6. ↑before(x;map(λx.x.1;ps))
  7. ¬(y = e)
  ⊢ ∀qs:|oal(s;g)|. ∀x1:|s|. ∀y1:|g|.
  |    ↑before(x1;map(λx.x.1;qs))
  |    ⇒ ¬(y1 = e)
  |    ⇒ ¬((<x, y>::ps) = 00)
  |    ⇒ ¬((<x1, y1>::qs) = 00)
  |    ⇒ ¬((<x, y>::ps) ++ (<x1, y1>::qs) = 00)
  |    ⇒ lk(<x, y>::ps) = lk(<x1, y1>::qs)
  |    ⇒ ¬(lv(<x, y>::ps) * lv(<x1, y1>::qs) = e)
  |    ⇒ lk((<x, y>::ps) ++ (<x1, y1>::qs)) = lk(<x1, y1>::qs)
  |
  BY (RepeatMFor 8 (D 0) ...a)
  |  THEN Try (BLemma `cons_in_oalist` ...)
  |
  8. qs: |oal(s;g)|
  9. x1: |s|
  10. y1: |g|
  11. ↑before(x1;map(λx.x.1;qs))
  12. ¬(y1 = e)
  13. ¬((<x, y>::ps) = 00)
  14. ¬((<x1, y1>::qs) = 00)
  15. ¬((<x, y>::ps) ++ (<x1, y1>::qs) = 00)
  ⊢ lk(<x, y>::ps) = lk(<x1, y1>::qs)
  |  ⇒ ¬(lv(<x, y>::ps) * lv(<x1, y1>::qs) = e)
  |  ⇒ lk((<x, y>::ps) ++ (<x1, y1>::qs)) = lk(<x1, y1>::qs)
  |
  BY (Reduce 0
  |   THENM SplitOnConclITEs
  |   THENM Reduce 0
  |   THENM D 0 ...a)
  |\
  | 16. x1 <s x
  | 17. x = x1
  | ⊢ ¬(y * y1 = e) ⇒ x = x1
  | |
  1 BY (RelRST ...)
  |\
  | 16. ¬(x1 <s x)
  | 17. x <s x1
  | 18. x = x1
  | ⊢ ¬(y * y1 = e) ⇒ x1 = x1
  | |
  1 BY (RelRST ...)
  |\
  | 16. ¬(x1 <s x)
  | 17. ¬(x <s x1)
  | 18. y * y1 = e
  | 19. x = x1
  | ⊢ ¬(y * y1 = e) ⇒ lk(ps ++ qs) = x1
```

```
    | |
    1 BY Auto
     \
      16. ¬(x1 <s x)
      17. ¬(x <s x1)
      18. ¬(y * y1 = e)
      19. x = x1
      ⊢ ¬(y * y1 = e) ⇒ x = x1
      |
      BY Auto
*C oal_lk_neg_com
               With partial functions there is always the question
               of how explicit to make all the totality guaranteeiing
               conditions. Here at one point, have to check that
               oal_neg(ps) is not nil because it occurs as arg to oal_lk.
*T oal_lk_neg  18.3 sec.
⊢ ∀s:LOSet. ∀g:AbGrp. ∀ps:|oal(s;g)|.  ¬(ps = 00) ⇒ lk(--ps) = lk(ps)
|
BY (D 0 THENM D 0
|   THENM BLemma `oalist_cases_c`  ...a)
|\
| 1. s: LOSet
| 2. g: AbGrp
| 3. ps: |oal(s;g)|
| 4. ¬(ps = 00)
| ⊢ ¬(--ps = 00)
| |
1 BY (D 0 THENM D 4 ...a)
| |
| 4. --ps = 00
| ⊢ ps = 00
| |
1 BY (RWW "oal_neg_eq_nil" 4 ...)
|\
| 1. s: LOSet
| 2. g: AbGrp
| ⊢ ¬(00 = 00) ⇒ lk(--00) = lk(00)
| |
1 BY (D 0 THENM D (-1) ...)
 \
  1. s: LOSet
  2. g: AbGrp
  ⊢ ∀ps:|oal(s;g)|. ∀x:|s|. ∀y:|g|.
  |    ↑before(x;map(λx.x.1;ps))
  |    ⇒ ¬(y = e)
  |    ⇒ ¬(oal_cons_pr(x;y;ps) = 00)
  |    ⇒ lk(--oal_cons_pr(x;y;ps)) = lk(oal_cons_pr(x;y;ps))
  |
  BY (RepD ...a)
  |
  3. ps: |oal(s;g)|
  4. x: |s|
  5. y: |g|
  6. ↑before(x;map(λx.x.1;ps))
  7. ¬(y = e)
  8. ¬(oal_cons_pr(x;y;ps) = 00)
  ⊢ lk(--oal_cons_pr(x;y;ps)) = lk(oal_cons_pr(x;y;ps))
```

```
  |
  BY Force ‘5‘ (Eval ‘‘oal_cons_pr‘‘ 0)
  |
  ⊢ x = x
  |
  BY Auto
*T lookup_oal_lk  23.3 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|.  ¬(ps = 00) ⇒ ps[lk(ps)] = lv(ps)
|
BY (D 0 THENM D 0 THENM BLemma ‘oalist_cases_a‘ ...a)
|\
| 1. s: LOSet
| 2. g: AbMon
| ⊢ ¬([] = 00) ⇒ [][lk([])] = lv([])
| |
1 BY % ‘*‘ is special option that allows for extra vars on rhs %
| |  RWH (SimpleMacroC ‘*‘ ⌜[]⌝ ⌜00⌝ ‘‘oal_nil‘‘) 0
| |
| ⊢ ¬(00 = 00) ⇒ 00[lk(00)] = lv(00)
| |
1 BY (D 0 THENM D (-1) ...)
 \
  1. s: LOSet
  2. g: AbMon
  ⊢ ∀ps:|oal(s;g)|. ∀x:|s|. ∀y:|g|.
  |    ↑before(x;map(λx.x.1;ps))
  |    ⇒ ¬(y = e)
  |    ⇒ ¬((<x, y>::ps) = 00)
  |    ⇒ (<x, y>::ps)[lk(<x, y>::ps)] = lv(<x, y>::ps)
  |
  BY (Force ‘5‘ (Reduce 0) THEN RepD  ...a)
  |\
  | 3. ps: |oal(s;g)|
  | 4. x: |s|
  | 5. y: |g|
  | 6. ↑before(x;map(λx.x.1;ps))
  | 7. ¬(y = e)
  | 8. ¬((<x, y>::ps) = 00)
  | ⊢ if x =b x then y else ps[x] fi  = y
  | |
  1 BY (SplitOnConclITE THENM Try RelRST ...)
   \
    3. ps: |oal(s;g)|
    4. x: |s|
    5. y: |g|
    6. ↑before(x;map(λx.x.1;ps))
    7. ¬(y = e)
    ⊢ (<x, y>::ps) ∈ |oal(s;g)|
    |
    BY (BLemma ‘cons_in_oalist‘ ...)
*T oal_lv_neg  17.1 sec.
⊢ ∀s:LOSet. ∀g:AbGrp. ∀ps:|oal(s;g)|.  ¬(ps = 00) ⇒ lv(--ps) = ∼ lv(ps)
|
BY (RepD THENM RWW "lookup_oal_lk<" 0 ...a)
|\
| 1. s: LOSet
| 2. g: AbGrp
```

```
| 3. ps: |oal(s;g)|
| 4. ¬(ps = 00)
| ⊢ ¬(--ps = 00)
| |
1 BY (D 0 THENM D 4 ...a)
| |
| 4. --ps = 00
| ⊢ ps = 00
| |
1 BY (ApFunToHypEquands
| |     'qs' ⌈qs ++ ps⌉ ⌈|oal(s;g)|⌉ 4 ...a)
| |
| 5. --ps ++ ps = 00 ++ ps
| |
1 BY (RWW "oal_nil_ident_l oal_neg_left_inv" 5 ...)
 \
  1. s: LOSet
  2. g: AbGrp
  3. ps: |oal(s;g)|
  4. ¬(ps = 00)
  ⊢ (--ps)[lk(--ps)] = ∼ ps[lk(ps)]
  |
  BY (RWW "oal_lk_neg" 0 THENM RWW "lookup_oal_neg" 0 ...a)
  |
  ⊢ ∼ ps[lk(ps)] = ∼ ps[lk(ps)]
  |
  BY Auto
*C ocgrp_constr
               ====================================
               CONSTRUCTION OF ORDER REL ON OALISTS
               ====================================
               The simplest order (a lexicographic order)
               is induced on oalists by orders on the
               key and value domains of oalists.
               For the purposes of defining this order,
               it is convenient to assume that the value
               domain is an abelian group rather than an
               abelian monoid.
*D oal_bpos_df pos{<s:s:*>,<g:g:*>}(<ps:ps:*>)== oal_bpos{}(<s>; <g>; <ps>)
               pos(<ps:ps:*>)== oal_bpos{}(<s>; <g>; <ps>)
*A oal_bpos                    pos(ps) ==  ¬_b null(ps) ∧_b e <_b lv(ps)
*T oal_bpos_wf  8.8 sec.
⊢ ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|.  pos(ps) ∈ 𝔹
|
BY (RepD THENM Unfold 'oal_bpos' 0 ...a)
|
1. s: LOSet
2. g: AbMon
3. ps: |oal(s;g)|
⊢ (¬_b null(ps) ∧_b e <_b lv(ps)) ∈ 𝔹
|
BY (Unfold 'band' 0
|   THEN SplitOnConclITE ...a)
|\
| 4. ¬(ps = 00)
| ⊢ (e <_b lv(ps)) ∈ 𝔹
| |
```

```
1 BY % Note how bpos set up so this is automatic %
|    Auto
 \
  4. ¬¬(ps = 00)
  ⊢ ff ∈ 𝔹
  |
  BY Auto
*T comb_for_oal_bpos_wf  1.1 sec.
⊢ (λs,g,ps,z.pos(ps)) ∈ s:LOSet ⟶ g:AbMon ⟶ ps:|oal(s;g)| ⟶ ↓True ⟶ 𝔹
|
BY ProveOpCombTyping ‘oal_bpos_wf‘
*D oal_blt_df  Parens ::Prec(inop)::
                 <ps:ps:L> <<b<s:s:L>,<g:g:L> <qs:qs:L>
                 == oal_blt{}(<s>; <g>; <ps>; <qs>)
               Parens ::Prec(inop)::  <ps:ps:L> <<b <qs:qs:L>== oal_blt{}(<s>; <g>; <ps>; <qs>)
*A oal_blt                    ps <<b qs ==  pos(qs ++ --ps)
*T oal_blt_wf  5.3 sec.
⊢ ∀s:LOSet. ∀g:AbGrp. ∀ps,qs:|oal(s;g)|.  ps <<b qs ∈ 𝔹
|
BY (Unfold ‘oal_blt‘ 0 ...)
*D oal_ble_df  <ps:ps:L> ≤≤b<s:s:L>,<g:g:L> <qs:qs:L>== oal_ble{}(<s>; <g>; <ps>; <qs>)
               <ps:ps:L> ≤≤b <qs:qs:L>== oal_ble{}(<s>; <g>; <ps>; <qs>)
*A oal_ble                    ps ≤≤b qs ==  (ps =b qs) ∨b(ps <<b qs)
*T oal_ble_wf  5.9 sec.
⊢ ∀s:LOSet. ∀g:AbGrp. ∀ps,qs:|oal(s;g)|.  ps ≤≤b qs ∈ 𝔹
|
BY (Unfold ‘oal_ble‘ 0 ...)
*D oal_le_df   Parens ::Prec(atomrel)::
                 <ps:ps:L> ≤{<s:s:*>,<g:g:*>} <qs:qs:L>
                 == oal_le{}(<s>; <g>; <ps>; <qs>)
*A oal_le                     ps ≤{s,g} qs ==  ↑ps ≤≤b qs
*T oal_le_wf  4.1 sec.
⊢ ∀s:LOSet. ∀g:AbGrp. ∀ps,qs:|oal(s;g)|.  (ps ≤{s,g} qs) ∈ ℙ1
|
BY (Unfold ‘oal_le‘ 0 ...)
*C oal_grp_com ============================
               INTRODUCTION OF OALIST GROUP
               ============================
               This is not ideally placed. Definition had to wait
               for introduction of order function.
               However, characterization as ordered group comes later.
*D oal_grp_df                 oal_grp(<s:s:*>;<g:g:*>)== oal_grp{}(<s>; <g>)
*A oal_grp     oal_grp(s;g) ==  <|oal(s;g)|, =b, λx,y.x ≤≤b y, λx,y.x ++ y, 00, λx.--x>
*T oal_grp_wf  18.4 sec.
⊢ ∀s:LOSet. ∀g:AbGrp.  oal_grp(s;g) ∈ AbGrp
|
BY (Unfold ‘oal_grp‘ 0
|   THENM RepD ...a)
|
1. s: LOSet
2. g: AbGrp
⊢ <|oal(s;g)|, =b, λx,y.x ≤≤b y, λx,y.x ++ y, 00, λx.--x> ∈ AbGrp
|
BY (RepeatM MemTypeCD ...a)
|  THEN Force ‘5‘ (Reduce 0)
|\
| ⊢ <|oal(s;g)|, =b, λx,y.x ≤≤b y, λx,y.x ++ y, 00, λx.--x> ∈ GrpSig
```

```
| |
1 BY (Unfold ‘grp_sig‘ 0 ...)
|\
| ⊢ IsMonoid(|oal(s;g)|;λx,y.x ++ y;00)
| |
1 BY (AGenRepD ["compound";"basic"]
|     THENM Force ‘5‘ (Reduce 0)
|     THENM Backchain ‘‘oal_merge_assoc oal_nil_ident_l oal_nil_ident_r‘‘ ...)
|\
| ⊢ IsEqFun(|oal(s;g)|;=_b)
| |
1 BY (Assert ⌈oal(s;g) ∈ DSet⌉
|     THENM AddProperties (-1) ...)
|\
| ⊢ Inverse(|oal(s;g)|;λx,y.x ++ y;00;λx.--x)
| |
1 BY (Force ‘5‘ (Eval ‘‘inverse‘‘ 0)
|     THENM Backchain ‘‘oal_neg_left_inv oal_neg_right_inv‘‘ ...)
 \
  ⊢ Comm(|oal(s;g)|;λx,y.x ++ y)
  |
  BY (Force ‘5‘ (Eval ‘‘comm‘‘ 0)
      THENM Backchain ‘‘oal_merge_comm‘‘ ...)
*M oal_grp_ml  % Conversions for folding up oal_grp components %
               let oal_grpC,rem_oal_grpC =
                 let cprs =
                  map (\t,t'. DoubleMacroC ‘oal_grpC‘ IdC t (ForceReduceC ‘5‘) t')
                 [⌈|oal(s;g)|⌉,⌈|oal_grp(s;g)|⌉
                 ;⌈ps ++ qs⌉,⌈ps * qs⌉
                 ;⌈--ps⌉,⌈∼ ps⌉
                 ;⌈00⌉,⌈e⌉
                 ]
                  in
                    FirstC (map fst cprs),FirstC (map snd cprs)
               ;;
               let oal_grp_leC =
                 MacroC ‘oal_grp_leC‘
                   (UnfoldC ‘oal_le‘) ⌈ps ≤{s,g} qs⌉
                   (EvalC ‘‘grp_leq‘‘) ⌈ps ≤ qs⌉
               ;;
               let WithOalAsGrp T i =
                 RWD oal_grpC i
                 THENM T i
                 THENM RWD rem_oal_grpC i
               ;;
*D oal_lt_df   Parens ::Prec(atomrel)::
                 <ps:ps:L> <<<s:s:*>,<g:g:*> <qs:qs:L>
                 == oal_lt{}(<s>; <g>; <ps>; <qs>)
               Parens ::Prec(atomrel)::
                 <ps:ps:L> << <qs:qs:L>
                 == oal_lt{}(<s>; <g>; <ps>; <qs>)
*A oal_lt      ps << qs ==  ∃k:|s|. (∀k':|s|. k <s k' ⇒ ps[k'] = qs[k']) ∧ ps[k] < qs[k]
*T oal_lt_wf  13.3 sec.
⊢ ∀s:LOSet. ∀g:OCMon. ∀ps,qs:|oal(s;g)|.  (ps << qs) ∈ ℙ
|
BY (Unfold ‘oal_lt‘ 0 ...)
*T decidable__oal_lt  6.3 sec.
```

```
⊢ ∀s:LOSet. ∀g:OGrp. ∀ps,qs:|oal(s;g)|.  Dec(ps << qs)
|
BY (RepD THENM RWW "assert_of_oal_blt<" 0 ...)
*T assert_of_oal_blt  140.1 sec.
⊢ ∀s:LOSet. ∀g:OGrp. ∀ps,qs:|oal(s;g)|.  ↑(ps <<b qs) ⟺ ps << qs
|
BY (RepD THENM Unfold `oal_lt` 0 ...a)
|
1. s: LOSet
2. g: OGrp
3. ps: |oal(s;g)|
4. qs: |oal(s;g)|
⊢ ↑(ps <<b qs) ⟺ (∃k:|s|. (∀k':|s|. k <s k' ⇒ ps[k'] = qs[k']) ∧ ps[k] < qs[k])
|
BY (RWH (PolyC "grp_lt_shift_right grp_eq_shift_right") 0 ...a)
|
⊢ ↑(ps <<b qs)
|  ⟺ (∃k:|s|. (∀k':|s|. k <s k' ⇒ e = qs[k'] * (∼ ps[k'])) ∧ e < qs[k] * (∼ ps[k]))
|
BY RepUnfolds ``oal_blt oal_bpos band`` 0
|  THEN (SplitOnConclITE THENM Reduce 0 ...a)
|\
| 5. ¬(qs ++ --ps = 00)
| ⊢ ↑(e <b lv(qs ++ --ps))
| |  ⟺ (∃k:|s|. (∀k':|s|. k <s k' ⇒ e = qs[k'] * (∼ ps[k'])) ∧ e < qs[k] * (∼ ps[k]))
| |
1 BY (GenExRepD ...a)
| |\
| | 6. ↑(e <b lv(qs ++ --ps))
| | ⊢ ∃k:|s|. (∀k':|s|. k <s k' ⇒ e = qs[k'] * (∼ ps[k'])) ∧ e < qs[k] * (∼ ps[k])
| | |
1 2 BY (With ⌈lk(qs ++ --ps)⌉ (D 0)
| | |   THENM GenRepD ...a)
| | |\
| | | 7. k': |s|
| | | 8. lk(qs ++ --ps) <s k'
| | | ⊢ e = qs[k'] * (∼ ps[k'])
| | | |
1 2 3 BY (RWW "lookup_merge< lookup_oal_neg<" 0 ...a)
| | | |
| | | ⊢ e = (qs ++ --ps)[k']
| | | |
1 2 3 BY (RWW "lookup_oal_eq_id" 0 ...)
| | | |
| | | ⊢ ¬↑(k' ∈b dom(qs ++ --ps))
| | | |
1 2 3 BY (D 0 THENM FLemma `oal_lk_bounds_dom` [-1]
| | |     THENM RelRST ...a)
| |  \
| |   ⊢ e < qs[lk(qs ++ --ps)] * (∼ ps[lk(qs ++ --ps)])
| |   |
1 2   BY (RWW "lookup_merge< lookup_oal_neg< lookup_oal_lk" 0 ...a)
| |   |
| |   ⊢ e < lv(qs ++ --ps)
| |   |
1 2   BY (RW bool_to_propC 6 ...)
|  \
```

```
|   6. k: |s|
|   7. ∀k':|s|. k <s k' ⇒ e = qs[k'] * (∼ ps[k'])
|   8. e < qs[k] * (∼ ps[k])
|   ⊢ ↑(e <b lv(qs ++ --ps))
|   |
1   BY (OnMCls [7;8] (RWW "lookup_merge< lookup_oal_neg< lookup_oal_lk")
|   |   THENM RW bool_to_propC 0
|   |   THENM InstLemma `loset_trichot` [⌈s⌉;⌈k⌉;⌈lk(qs ++ --ps)⌉]
|   |   THENM GenExRepD ...a)
|   |\
|   | 7. ∀k':|s|. k <s k' ⇒ e = (qs ++ --ps)[k']
|   | 8. e < (qs ++ --ps)[k]
|   | 9. k <s lk(qs ++ --ps)
|   | ⊢ e < lv(qs ++ --ps)
|   | |
1   2 BY (FHyp 7 [9]
|   | |   THENM RWW "lookup_oal_lk" 10 ...a)
|   | |
|   | 10. e = lv(qs ++ --ps)
|   | |
1   2 BY (InvertRel 10 THENM Negate 10 THENM BLemma `oal_lv_nid` ...)
|   |\
|   | 7. ∀k':|s|. k <s k' ⇒ e = (qs ++ --ps)[k']
|   | 8. e < (qs ++ --ps)[k]
|   | 9. k = lk(qs ++ --ps)
|   | ⊢ e < lv(qs ++ --ps)
|   | |
1   2 BY (RWW "9 lookup_oal_lk" 8 ...)
|    \
|     7. ∀k':|s|. k <s k' ⇒ e = (qs ++ --ps)[k']
|     8. e < (qs ++ --ps)[k]
|     9. lk(qs ++ --ps) <s k
|     ⊢ e < lv(qs ++ --ps)
|     |
1     BY (RWW "lookup_oal_eq_id" 8 THENM RelRST ...a)
|     |
|     ⊢ ¬↑(k ∈b dom(qs ++ --ps))
|     |
1     BY (D 0 THENM FLemma `oal_lk_bounds_dom` [-1]
|         THENM RelRST ...)
 \
  5. ¬¬(qs ++ --ps = 00)
  ⊢ False ⟺ (∃k:|s|. (∀k':|s|. k <s k' ⇒ e = qs[k'] * (∼ ps[k'])) ∧ e < qs[k] * (∼ ps[k]))
  |
  BY (FLemma `dneg_elim` [5] THENM Thin 5 ...a)
  |
  5. qs ++ --ps = 00
  |
  BY (RWH (LemmaC `oal_equal_char`) 5 ...a)
  |
  5. ∀u:|s|. (qs ++ --ps)[u] = 00[u]
  |
  BY (RWW "lookup_merge lookup_oal_neg" 5 ...a)
  |  THEN Eval ``oal_nil`` 5
  |
  5. ∀u:|s|. qs[u] * (∼ ps[u]) = e
  |
```

```
  BY (GenExRepD ...)
  |
  6. k: |s|
  7. ∀k':|s|. k <s k' ⇒ e = qs[k'] * (∼ ps[k'])
  8. e < qs[k] * (∼ ps[k])
  ⊢ False
  |
  BY (With ⌈k⌉ (D 5) THENM RelRST ...)
*M assert_of_oal_blt_ml       update_assert_elim_lemmas ``assert_of_oal_blt`` ;;
*T oal_lt_irrefl  4.8 sec.
⊢ ∀s:LOSet. ∀g:OCMon.  Irrefl(|oal(s;g)|;ps,qs.ps << qs)
|
BY (RepUnfolds ``irrefl not`` 0
|   THENM RepD
|   THENM Unfold `oal_lt` (-1)
|   THENM ExRepD ...a)
|
1. s: LOSet
2. g: OCMon
3. a: |oal(s;g)|
4. k: |s|
5. ∀k':|s|. k <s k' ⇒ a[k'] = a[k']
6. a[k] < a[k]
⊢ False
|
BY (RelRST ...)
*T oal_lt_trans  66.0 sec.
⊢ ∀s:LOSet. ∀g:OCMon.  Trans(|oal(s;g)|;ps,qs.ps << qs)
|
BY (ARepD ["basic"]
|   THENM OnCls [0;-1;-2] (Unfold `oal_lt`)
|   THENM ExRepD ...a)
|
1. s: LOSet
2. g: OCMon
3. a: |oal(s;g)|
4. b: |oal(s;g)|
5. c: |oal(s;g)|
6. k1: |s|
7. ∀k':|s|. k1 <s k' ⇒ a[k'] = b[k']
8. a[k1] < b[k1]
9. k: |s|
10. ∀k':|s|. k <s k' ⇒ b[k'] = c[k']
11. b[k] < c[k]
⊢ ∃k:|s|. (∀k':|s|. k <s k' ⇒ a[k'] = c[k']) ∧ a[k] < c[k]
|
BY % Strategy: pick k in concl as being where a and c first differ %
|  (Decide ⌈k ≤ k1⌉ ...a)
|\
| 12. k ≤ k1
| |
1 BY (With ⌈k1⌉ (D 0)THENM GenRepD ...a)
| |\
| | 13. k': |s|
| | 14. k1 <s k'
| | ⊢ a[k'] = c[k']
| | |
```

```
1 2 BY (OnMHyps [10;7] (InstHyp [⌈k'⌉])
| |     THEN Try RelRST  ...)
|  \
|   ⊢ a[k1] < c[k1]
|   |
1   BY (RWW "set_leq_iff_lt_or_eq" 12
|   |   THENM D 12 ...a)
|   |\
|   | 12. k <s k1
|   | |
1   2 BY (InstHyp [⌈k1⌉] 10 THENM RelRST ...)
|    \
|     12. k = k1
|     |
1     BY (RWW "12" 11 THENM RelRST ...)
 \
  12. ¬(k ≤ k1)
  |
  BY (With ⌈k⌉ (D 0) THENM GenRepD ...a)
  |\
  | 13. k': |s|
  | 14. k <s k'
  | ⊢ a[k'] = c[k']
  | |
  1 BY (OnMHyps [10;7] (InstHyp [⌈k'⌉])
  |     THEN Try RelRST  ...)
   \
    ⊢ a[k] < c[k]
    |
    BY (InstHyp [⌈k⌉] 7 THEN Try RelRST ...)
*T oal_bpos_trichot  54.8 sec.
⊢ ∀s:LOSet. ∀g:OGrp. ∀rs:|oal(s;g)|.  ↑pos(rs) ∨ rs = 00 ∨ ↑pos(--rs)
|
BY (RepD THENM RepUnfolds ``oal_bpos band`` 0
|   THENM SplitOnConclITEs ...a)
|\
| 1. s: LOSet
| 2. g: OGrp
| 3. rs: |oal(s;g)|
| 4. ¬(rs = 00)
| 5. ¬(--rs = 00)
| ⊢ ↑(e <b lv(rs)) ∨ rs = 00 ∨ ↑(e <b lv(--rs))
| |
1 BY (RWW "assert_of_grp_blt oal_lv_neg" 0 ...a)
| |
| ⊢ e < lv(rs) ∨ rs = 00 ∨ e < ∼ lv(rs)
| |
1 BY (InstLemma `grp_lt_trichot` [⌈g⌉;⌈e⌉;⌈lv(rs)⌉]
| |   THENM GenExRepD ...a)
| |\
| | 6. e < lv(rs)
| | |
1 2 BY (Sel 1 (D 0) ...)
| |\
| | 6. e = lv(rs)
| | |
1 2 BY (InvertRel 6 THENM Negate 6
```

```
| |     THENM BLemma `oal_lv_nid` ...)
|  \
|   6. lv(rs) < e
|   |
1   BY (Sel 3 (D 0) ...a)
|   |
|   ⊢ e < ∼ lv(rs)
|   |
1   BY (RWO "grp_lt_shift_right" 6
|       THENM RW GrpNormC 6 ...)
|\
| 1. s: LOSet
| 2. g: OGrp
| 3. rs: |oal(s;g)|
| 4. ¬(rs = 00)
| 5. ¬¬(--rs = 00)
| ⊢ ↑(e <b lv(rs)) ∨ rs = 00 ∨ ↑ff
| |
1 BY (RWW "oal_neg_eq_nil" 5 ...)
|\
| 1. s: LOSet
| 2. g: OGrp
| 3. rs: |oal(s;g)|
| 4. ¬¬(rs = 00)
| 5. ¬(--rs = 00)
| ⊢ ↑ff ∨ rs = 00 ∨ ↑(e <b lv(--rs))
| |
1 BY (RWW "oal_neg_eq_nil" 5 ...)
 \
  1. s: LOSet
  2. g: OGrp
  3. rs: |oal(s;g)|
  4. ¬¬(rs = 00)
  5. ¬¬(--rs = 00)
  ⊢ ↑ff ∨ rs = 00 ∨ ↑ff
  |
  BY (RWW "dneg_elim_a" 4 THENM ProveProp  ...)
*T oal_lt_trichot  63.5 sec.
⊢ ∀s:LOSet. ∀g:OGrp. ∀ps,qs:|oal(s;g)|.  ps << qs ∨ ps = qs ∨ qs << ps
|
BY (RepD THENM RWW "assert_of_oal_blt<" 0 ...a)
|
1. s: LOSet
2. g: OGrp
3. ps: |oal(s;g)|
4. qs: |oal(s;g)|
⊢ ↑(ps <<b qs) ∨ ps = qs ∨ ↑(qs <<b ps)
|
BY Unfold `oal_blt` 0
|
⊢ ↑pos(qs ++ --ps) ∨ ps = qs ∨ ↑pos(ps ++ --qs)
|
BY (RWD oal_grpC 0
|   THENM RWN 2 (PolyC "grp_inv_diff<") 0
|   THENM RWO "grp_eq_shift_right" 0
|   THENM RWO "equal_symmetry" 0
|   THENM RWD rem_oal_grpC 0 ...a)
```

```
|
⊢ ↑pos(qs ++ --ps) ∨ qs ++ --ps = 00 ∨ ↑pos(--(qs ++ --ps))
|
BY (BLemma ‘oal_bpos_trichot‘ ...)
#T oal_lt_trichot_a  38.6 sec.
⊢ ∀s:LOSet. ∀g:OGrp. ∀ps,qs:|oal(s;g)|.  ps << qs ∨ ps = qs ∨ qs << ps
|
BY (RepD THENM RWW "assert_of_oal_blt<" 0 ...a)
|
1. s: LOSet
2. g: OGrp
3. ps: |oal(s;g)|
4. qs: |oal(s;g)|
⊢ ↑(ps <<b qs) ∨ ps = qs ∨ ↑(qs <<b ps)
|
BY Unfold ‘oal_blt‘ 0
|
⊢ ↑pos(qs ++ --ps) ∨ ps = qs ∨ ↑pos(ps ++ --qs)
|
BY RWD oal_grpC 0
|
⊢ ↑pos(qs * (∼ ps)) ∨ ps = qs ∨ ↑pos(ps * (∼ qs))
|
BY (RWN 2 (PolyC "grp_inv_diff<") 0
|   THENM RWO "grp_eq_shift_right" 0
|   THENM RWO "equal_symmetry" 0 ...a)
|
⊢ ↑pos(qs * (∼ ps)) ∨ qs * (∼ ps) = e ∨ ↑pos(∼ (qs * (∼ ps)))
|
BY RWD rem_oal_grpC 0
|
⊢ ↑pos(qs ++ --ps) ∨ qs ++ --ps = 00 ∨ ↑pos(--(qs ++ --ps))
|
INCOMPLETE
*T oal_merge_preserves_lt  32.7 sec.
⊢ ∀s:LOSet. ∀g:OCMon. ∀ps,qs,rs:|oal(s;g)|.  qs << rs ⇒ ps ++ qs << ps ++ rs
|
BY (RepD THENM OnCls [0;-1] (Unfold ‘oal_lt‘)
|   THENM ExRepD  ...a)
|
1. s: LOSet
2. g: OCMon
3. ps: |oal(s;g)|
4. qs: |oal(s;g)|
5. rs: |oal(s;g)|
6. k: |s|
7. ∀k’:|s|. k <s k’ ⇒ qs[k’] = rs[k’]
8. qs[k] < rs[k]
⊢ ∃k:|s|. (∀k’:|s|. k <s k’ ⇒ (ps ++ qs)[k’] = (ps ++ rs)[k’]) ∧ (ps ++ qs)[k] < (ps ++ rs)[k]
|
BY (With ⌈k⌉ (D 0) THENM GenRepD
|   THENM RWH (LemmaC ‘lookup_merge‘) 0 ...a)
|\
| 9. k’: |s|
| 10. k <s k’
| ⊢ ps[k’] * qs[k’] = ps[k’] * rs[k’]
| |
```

```
1 BY (RWH (HypC 7) 0 ...)
 \
  ⊢ ps[k] * qs[k] < ps[k] * rs[k]
  |
  BY (BLemma ‘grp_op_preserves_lt‘ ...)
*M oal_le_order_decl
               Relation Family
                <: ps << qs
                ≤: ps ≤{s,g} qs
                ≡: ps = qs
                ≥: ?
                >: ?
               add_lin_order_check_fun ‘oal_lt‘ (\p r.true) ;;
*T oal_le_char  12.4 sec.
⊢ ∀s:LOSet. ∀g:OGrp.
|    (ps,qs:|oal(s;g)|. ps ≤{s,g} qs) <≡>{|oal(s;g)|} (ps,qs:|oal(s;g)|. ps << qs)^0
|
BY (Unfold ‘binrel_eqv‘ 0
|   THEN RepD ...a)
|
1. s: LOSet
2. g: OGrp
3. x: |oal(s;g)|
4. y: |oal(s;g)|
⊢ (ps,qs:|oal(s;g)|. ps ≤{s,g} qs) x y ⟺ (ps,qs:|oal(s;g)|. ps << qs)^0 x y
|
BY Force ‘5‘ (Eval ‘‘refl_cl‘‘ 0)
|
⊢ x ≤{s,g} y ⟺ x = y ∨ x << y
|
BY RepUnfolds ‘‘oal_le oal_ble‘‘ 0
|
⊢ ↑((x =_b y) ∨_b(x <<_b y)) ⟺ x = y ∨ x << y
|
BY (RW bool_to_propC 0 ...a)
|
⊢ x = y ∨ x << y ⟺ x = y ∨ x << y
|
BY (RelRST ...)
*T oal_lt_char  20.7 sec.
⊢ ∀s:LOSet. ∀g:OGrp.
|    (ps,qs:|oal(s;g)|. ps << qs) <≡>{|oal(s;g)|} (ps,qs:|oal(s;g)|. ps ≤{s,g} qs)\
|
BY (RepD ...a)
|
1. s: LOSet
2. g: OGrp
⊢ (ps,qs:|oal(s;g)|. ps << qs) <≡>{|oal(s;g)|} (ps,qs:|oal(s;g)|. ps ≤{s,g} qs)\
|
BY (RWW "oal_le_char sp_refl_cl_cancel" 0 ...a)
|\
| ⊢ irrefl(|oal(s;g)|;ps,qs:|oal(s;g)|. ps << qs)
| |
1 BY (Force ‘5‘ (Eval ‘‘xxirrefl‘‘ 0)
|     THENM BLemma ‘oal_lt_irrefl‘ ...a)
|\
| ⊢ st_anti_sym(|oal(s;g)|;ps,qs:|oal(s;g)|. ps << qs)
```

```
| |
1 BY (BLemma `irrefl_trans_imp_sasym`
|     THENM Force `5` (Eval ``xxirrefl xxtrans`` 0)
|     THENM Backchain ``oal_lt_irrefl oal_lt_trans``  ...a)
 \
  ⊢ (ps,qs:|oal(s;g)|. ps << qs) <≡>{|oal(s;g)|} (ps,qs:|oal(s;g)|. ps << qs)
  |
  BY (RelRST ...)
*T oal_le_is_order  9.5 sec.
⊢ ∀s:LOSet. ∀g:OGrp.  Order(|oal(s;g)|;ps,qs.ps ≤{s,g} qs)
|
BY (RepD ...a)
|
1. s: LOSet
2. g: OGrp
⊢ Order(|oal(s;g)|;ps,qs.ps ≤{s,g} qs)
|
BY % Switch to treating relations-as-1st-class-objects %
|  (RWH ab_binrelC 0
|   THENM RWW "xxorder_eq_order<" 0 ...a)
|
⊢ order(|oal(s;g)|;x,y:|oal(s;g)|. x ≤{s,g} y)
|
BY (RWW "oal_le_char" 0 ...a)
|
⊢ order(|oal(s;g)|;(x,y:|oal(s;g)|. x << y)^o)
|
BY (BLemma `refl_cl_is_order` ...a)
|\
| ⊢ irrefl(|oal(s;g)|;x,y:|oal(s;g)|. x << y)
| |
1 BY Force `5` (Eval ``xxirrefl`` 0)
|    THEN (BLemma `oal_lt_irrefl` ...)
 \
  ⊢ trans(|oal(s;g)|;x,y:|oal(s;g)|. x << y)
  |
  BY Force `5` (Eval ``xxtrans`` 0)
     THEN (BLemma `oal_lt_trans` ...)
*T oal_le_connex  8.0 sec.
⊢ ∀s:LOSet. ∀g:OGrp.  Connex(|oal(s;g)|;ps,qs.ps ≤{s,g} qs)
|
BY (RepD ...a)
|
1. s: LOSet
2. g: OGrp
⊢ Connex(|oal(s;g)|;ps,qs.ps ≤{s,g} qs)
|
BY % Switch to treating relations-as-1st-class-objects %
|  RWH ab_binrelC 0 THEN Fold `xxconnex` 0
|
⊢ connex(|oal(s;g)|;x,y:|oal(s;g)|. x ≤{s,g} y)
|
BY (RWW "oal_le_char" 0 ...a)
|
⊢ connex(|oal(s;g)|;(x,y:|oal(s;g)|. x << y)^o)
|
BY (BLemma `xxconnex_iff_trichot_a` ...a)
```

```
|
⊢ ∀a,b:|oal(s;g)|.  (x,y:|oal(s;g)|. x << y) a b ∨ a = b ∨ (x,y:|oal(s;g)|. x << y) b a
|
BY Force ‘5‘ (Reduce 0)
   THEN (Backchain ‘‘oal_lt_trichot‘‘ ...)
*T oal_grp_wf1  6.0 sec.
⊢ ∀s:LOSet. ∀g:OGrp.  oal_grp(s;g) ∈ OMon
|
BY (RepD THENM MemTypeCD ...a)
|\
| 1. s: LOSet
| 2. g: OGrp
| ⊢ oal_grp(s;g) ∈ AbMon
| |
1 BY (BLemma ‘oal_grp_wf‘ ...)
 \
  1. s: LOSet
  2. g: OGrp
  ⊢ Linorder(|oal_grp(s;g)|;x,y.↑(x ≤b y))
  |
  BY Force ‘5‘ (Reduce 0) THEN Fold ‘oal_le‘ 0
  |
  ⊢ Linorder(|oal(s;g)|;x,y.x ≤{s,g} y)
  |
  BY (Unfold ‘linorder‘ 0
      THEN Backchain ‘‘oal_le_is_order oal_le_connex‘‘ ...)
*T oal_lt_iff_grp_lt  24.0 sec.
⊢ ∀s:LOSet. ∀g:OGrp. ∀ps,qs:|oal(s;g)|.  ps << qs ⟺ ps < qs
|
BY (RepD ...a)
|
1. s: LOSet
2. g: OGrp
3. ps: |oal(s;g)|
4. qs: |oal(s;g)|
⊢ ps << qs ⟺ ps < qs
|
BY (RWW "grp_lt_is_sp_of_leq_a" 0 ...a)
|
⊢ ps << qs ⟺ ps ≤ qs ∧ ¬(qs ≤ ps)
|
BY % Need binrel_ap abstraction here for
|    functionality reasoning in subsequent rewrite %
|
|  RWN 2 ab_binrel_apC 0
|
⊢ ps [x,y:|oal(s;g)|. x << y] qs ⟺ ps ≤ qs ∧ ¬(qs ≤ ps)
|
BY (RWW "oal_lt_char" 0 ...a)
|
⊢ ps [(x,y:|oal(s;g)|. x ≤{s,g} y)\] qs ⟺ ps ≤ qs ∧ ¬(qs ≤ ps)
|
BY Force ‘5‘ (Eval ‘‘binrel_ap s_part‘‘ 0)
|
⊢ ps ≤{s,g} qs ∧ ¬(qs ≤{s,g} ps) ⟺ ps ≤ qs ∧ ¬(qs ≤ ps)
|
BY Force ‘5‘ (Eval ‘‘grp_leq‘‘ 0) THEN Fold ‘oal_le‘ 0
```

```
|
⊢ ps ≤{s,g} qs ∧ ¬(qs ≤{s,g} ps) ⟺ ps ≤{s,g} qs ∧ ¬(qs ≤{s,g} ps)
|
BY (RelRST ...)
*C oal_grp_wf2_com
               subtype reasoning needs looking at here.
               The identification of the extra properties of
               oal_grp that have to be proven should not
               depend on the particular inheritance structure
               adopted for the algebraic classes. It should also
               be automatic.
*T oal_grp_wf2  31.1 sec.
⊢ ∀s:LOSet. ∀g:OGrp.  oal_grp(s;g) ∈ OGrp
|
BY (RepD ...a)
|
1. s: LOSet
2. g: OGrp
⊢ oal_grp(s;g) ∈ OGrp
|
BY (Assert ⌈oal_grp(s;g) ∈ AbGrp⌉ ...a)
|
3. oal_grp(s;g) ∈ AbGrp
|
BY % This gets job done but is too definition sensitive %
|  (RepeatMFor 2 (MemTypeD 3)
|   THENM RepeatMFor 3 (MemTypeD 0) ...a)
|\
| 3. oal_grp(s;g) ∈ Mon
| 4. Inverse(|oal_grp(s;g)|;*;e;∼)
| 5. Comm(|oal_grp(s;g)|;*)
| ⊢ oal_grp(s;g) ∈ Mon
| |
1 BY Trivial
|\
| 3. oal_grp(s;g) ∈ Mon
| 4. Inverse(|oal_grp(s;g)|;*;e;∼)
| 5. Comm(|oal_grp(s;g)|;*)
| ⊢ Linorder(|oal_grp(s;g)|;x,y.↑(x ≤b y))
| |
1 BY Force `5` (Reduce 0)
| |  THENM Fold `oal_le` 0
| |
| ⊢ Linorder(|oal(s;g)|;x,y.x ≤{s,g} y)
| |
1 BY (Unfold `linorder` 0
|     THEN Backchain ``oal_le_connex oal_le_is_order`` ...)
|\
| 3. oal_grp(s;g) ∈ Mon
| 4. Inverse(|oal_grp(s;g)|;*;e;∼)
| 5. Comm(|oal_grp(s;g)|;*)
| ⊢ Cancel(|oal_grp(s;g)|;|oal_grp(s;g)|;*)
| |
1 BY % a property of any group... %
|
|     (Unfold `cancel` 0
|      THENM RepD
```

```
|      THENM FLemma `grp_op_cancel_l` [-1] ...)
 \
  3. oal_grp(s;g) ∈ Mon
  4. Inverse(|oal_grp(s;g)|;*;e;∼)
  5. Comm(|oal_grp(s;g)|;*)
  6. z: |oal_grp(s;g)|
  ⊢ monot(|oal_grp(s;g)|;x,y.↑(x ≤b y);λw.z * w)
  |
  BY MoveToConcl 6
  |  THEN Fold `grp_leq` 0
  |  THEN (BLemma `omon_lt_mono_imp_leq_mono` ...a)
  |
  ⊢ ∀z:|oal_grp(s;g)|. monot(|oal_grp(s;g)|;x,y.x < y;λw.z * w)
  |
  BY (Force `5` (Eval ``monot`` 0)
  |   THENM RepD ...a)
  |
  6. z: |oal(s;g)|
  7. x: |oal(s;g)|
  8. y: |oal(s;g)|
  9. x < y
  ⊢ z ++ x < z ++ y
  |
  BY (OnMCls [0;9] (RWW "oal_lt_iff_grp_lt<")  ...a)
  |
  9. x << y
  ⊢ z ++ x << z ++ y
  |
  BY (BLemma `oal_merge_preserves_lt` ...)
*T oal_merge_preserves_le  17.4 sec.
⊢ ∀s:LOSet. ∀g:OGrp. ∀ps,qs,rs:|oal(s;g)|.  qs ≤{s,g} rs ⇒ ps ++ qs ≤{s,g} ps ++ rs
|
BY (RepD ...a)
|
1. s: LOSet
2. g: OGrp
3. ps: |oal(s;g)|
4. qs: |oal(s;g)|
5. rs: |oal(s;g)|
6. qs ≤{s,g} rs
⊢ ps ++ qs ≤{s,g} ps ++ rs
|
BY OnCls [0;6] (RWD (oal_grpC ORELSEC oal_grp_leC))
|
6. qs ≤ rs
⊢ ps * qs ≤ ps * rs
|
BY (BLemma `grp_op_preserves_le` ...)
*C oal_hgp_com ====================
               ORDERED `HALF` GROUP
               ====================
               A `half group` is the ordered cancellation monoid
               of the non-negative elements of an ordered group.
*C set_car_inc_tcom
               This inclusion lemma is needed in proof of oal_hgp_wf.
               Need better way of finding appropriate inclusion lemmas.
               Ideally, this lemma's proof should be completely automatic.
```

```
              Problem is that type information makes clauses look different,
              even though they eventually normalize to the same thing.
*T set_car_inc  26.9 sec.
⊢ ∀s:LOSet. ∀g:OGrp.  |oal(s;g↓hgrp)| ⊆ |oal(s;g)|
|
BY (Unfold `subtype` 0
|   THENM RepD  ...a)
|
1. s: LOSet
2. g: OGrp
3. x: |oal(s;g↓hgrp)|
⊢ x ∈ |oal(s;g)|
|
BY SetupInclusion 3
|  THEN ProveSubtypingAdditions
|  THENM ProveSubtypingAdditions
|\
| 3. x: |((s × g↓hgrp↓set) List)|
| 4. ↑sd_ordered(map(λx.x.1;x)) ∧ ¬↑(e ∈b map(λx.x.2;x))
| ⊢ x = x
| |
1 BY Auto
|\
| 3. x: |((s × g↓hgrp↓set) List)|
| 4. ↑sd_ordered(map(λx.x.1;x)) ∧ ¬↑(e ∈b map(λx.x.2;x))
| ⊢ ↑sd_ordered(map(λx.x.1;x)) ∧ ¬↑(e ∈b map(λx.x.2;x))
| |
1 BY % This is really ugly. %
| |
| |  OnCls [0;4]
| |   (\i.Reduce i
| |     THEN RepUnfolds ``mem mon_for for tlambda`` i
| |     THEN Reduce i)
| |
| 4. ↑sd_ordered(map(λx.x.1;x)) ∧ ¬↑reduce(λx,y.(x ∨by);ff;map(λx.x =b e;map(λx.x.2;x)))
| ⊢ ↑sd_ordered(map(λx.x.1;x)) ∧ ¬↑reduce(λx,y.(x ∨by);ff;map(λx.x =b e;map(λx.x.2;x)))
| |
1 BY Trivial
 \
  3. x: |((s × g↓hgrp↓set) List)|
  4. ↑sd_ordered(map(λx.x.1;x)) ∧ ¬↑(e ∈b map(λx.x.2;x))
  5. ps: |((s × g↓set) List)|
  ⊢ (↑sd_ordered(map(λx.x.1;ps)) ∧ ¬↑(e ∈b map(λx.x.2;ps)))
  |  = (↑sd_ordered(map(λx.x.1;ps)) ∧ ¬↑(e ∈b map(λx.x.2;ps)))
  |
  BY Auto
*D oal_hgp_df                  oal_hgp(<s:s:*>;<g:g:*>)== oal_hgp{}(<s>; <g>)
*A oal_hgp     oal_hgp(s;g) ==  <|oal(s;g↓hgrp)|, =b, λx,y.x ≤≤b y, λx,y.x ++ y, 00, λx.x>
*T oal_hgp_wf  11.1 sec.
⊢ ∀s:LOSet. ∀g:OGrp.  oal_hgp(s;g) ∈ GrpSig
|
BY (RepD THENM Unfolds ``oal_hgp grp_sig`` 0 ...)
*M oal_hgp_ml  let oal_hgpC =
                 FirstC
                 (map (\t1,t2.MacroC `oal_hgrpC` IdC t1 (ForceReduceC `5`) t2)
                  [⌈|oal(s;g↓hgrp)|⌉,⌈|oal_hgp(s;g)|⌉
                  ;⌈=b⌉,⌈=b⌉
```

```
                  ;⌜x ++ y⌝,⌜x * y⌝
                  ;⌜00⌝,⌜e⌝
                  ])
               ;;
               let oal_add_hgrp_of_ocgrpC,oal_rem_hgrp_of_ocgrpC =
                 let C = RepeatC (EvalC
                   ``oalist eq_list eq_pair
                     oal_merge oal_nil infix_ap
                   ``) in
                 (FirstC # FirstC )
                 (unzip
                   (map (\t1,t2.
                            DoubleMacroC `add_oal_hgp` C t1 C t2)
                 [⌜=b⌝,⌜=b⌝
                 ;⌜ps ++ qs⌝,⌜ps ++ qs⌝
                 ;⌜00⌝,⌜00⌝
                 ]))
               ;;
*M oal_hgp_ml2 let oal_hgp_to_monC,oal_mon_to_hgpC =
                 let C1 = AbRedexC in
                 let C2 = AbRedexC in
                 (FirstC # FirstC )
                 (unzip
                   (map (\t1,t2. DoubleMacroC `oal_hgp_to_monC` C1 t1 C2 t2)
                 [⌜|oal_hgp(s;g)|⌝,⌜|oal_mon(s;g↓hgrp)|⌝
                 ;⌜=b⌝,⌜=b⌝
                 ;⌜*⌝,⌜*⌝
                 ;⌜e⌝,⌜e⌝
                 ]))
               ;;
*C oalist_hgrp_eqs_com
               This lemma proves exactly that property
               that should be true of subtypes, but that
               isn't with the current definition of
               the subtype `predicate' (`' because it
               isn't a fully fledged predicate well-formed
               for any pair of types.
               It also demonstrates proof patterns that could
               be pulled out into `template proofs'.
*T oalist_hgrp_eqs  44.6 sec.
⊢ ∀s:LOSet. ∀g:OGrp. ∀a1,a2:|oal(s;g↓hgrp)|.  a1 = a2 ⇒ a1 = a2
|
BY (RepD ...a)
|
1. s: LOSet
2. g: OGrp
3. a1: |oal(s;g↓hgrp)|
4. a2: |oal(s;g↓hgrp)|
5. a1 = a2
⊢ a1 = a2
|
BY (Reduce 5 THEN EqTypeD 5 ...a)
|  THEN OnCls [4;3] (\i.Reduce i THEN D i)
|  THEN (Reduce 0 THEN EqTypeCD ...a)
|  THEN OnHyps [8;6;4] Thin
|
3. a1: (|s| × |g|+) List
```

```
4. a2: (|s| × |g|+) List
5. a1 = a2
⊢ a1 = a2
|
BY % uggh...%
|  MoveToConcl 4 THENM ListIndA 3
|\
| ⊢ ∀a2:(|s| × |g|+) List. [] = a2 ⇒ [] = a2
| |
1 BY (D 0 THENM D (-1) ...)
| |
| 3. a2: (|s| × |g|+) List
| 4. u: |s| × |g|+
| 5. v: (|s| × |g|+) List
| 6. [] = (u::v)
| ⊢ [] = (u::v)
| |
1 BY (InvertRel 6
|     THENM Negate 6
|     THENM BLemma ‘cons_neq_nil‘ ...a)
 \
  3. a1: |s| × |g|+
  5. ∀a2@0:(|s| × |g|+) List. a2 = a2@0 ⇒ a2 = a2@0
  ⊢ ∀a2@0:(|s| × |g|+) List. (a1::a2) = a2@0 ⇒ (a1::a2) = a2@0
  |
  BY (D 0 THENM D (-1) THENM RepD  ...a)
  |\
  | 6. a2@0: (|s| × |g|+) List
  | 7. (a1::a2) = []
  | ⊢ (a1::a2) = []
  | |
  1 BY (Negate 7 THENM BLemma ‘cons_neq_nil‘ ...)
   \
    6. a2@0: (|s| × |g|+) List
    7. u: |s| × |g|+
    8. v: (|s| × |g|+) List
    9. (a1::a2) = (u::v)
    ⊢ (a1::a2) = (u::v)
    |
    BY EqCD
    |\
    | ⊢ a1 = u
    | |
    1 BY (FLemma ‘eq_cons_imp_eq_hds‘ [9] ...a)
    | |
    | 10. a1 = u
    | |
    1 BY (D 7 THENM D 3 THENM
    | |   EqD (-1) THENM All Reduce THENM EqD 0 ...)
    | |
    | 3. a3: |s|
    | 4. a4: |g|+
    | 5. a2: (|s| × |g|+) List
    | 6. ∀a2@0:(|s| × |g|+) List. a2 = a2@0 ⇒ a2 = a2@0
    | 7. a2@0: (|s| × |g|+) List
    | 8. u1: |s|
    | 9. u2: |g|+
```

```
    | 10. v: (|s| × |g|⁺) List
    | 11. (<a3, a4>::a2) = (<u1, u2>::v)
    | 12. a3 = u1
    | 13. a4 = u2
    | ⊢ a4 = u2
    | |
    1 BY (AddProperties 4 THENM MemTypeCD ...)
     \
      ⊢ a2 = v
      |
      BY (BHyp 5
          THENM ApFunToHypEquands 'b' ⌜tl(b)⌝ ⌜(|s| × |g|) List⌝ 9
          THENM Reduce 10 ...)
*T oal_hgp_wf2  52.6 sec.
⊢ ∀s:LOSet. ∀g:OGrp.  oal_hgp(s;g) ∈ OCMon
|
BY (RepD THENM BLemma 'inj_into_ocmon' ...a)
|
1. s: LOSet
2. g: OGrp
⊢ ∃h:OCMon
|   ∃f:|oal_hgp(s;g)| ⟶ |h|
|    IsMonHomInj(oal_hgp(s;g);h;f)
|    ∧ RelsIso(|oal_hgp(s;g)|;|h|;x,y.↑(x =b y);x,y.↑(x =b y);f)
|    ∧ RelsIso(|oal_hgp(s;g)|;|h|;x,y.↑(x ≤b y);x,y.↑(x ≤b y);f)
|
BY (InstConcl [⌜oal_grp(s;g)⌝;⌜λz.z⌝] ...a)
|
⊢ IsMonHomInj(oal_hgp(s;g);oal_grp(s;g);λz.z)
|  ∧ RelsIso(|oal_hgp(s;g)|;|oal_grp(s;g)|;x,y.↑(x =b y);x,y.↑(x =b y);λz.z)
|  ∧ RelsIso(|oal_hgp(s;g)|;|oal_grp(s;g)|;x,y.↑(x ≤b y);x,y.↑(x ≤b y);λz.z)
|
BY (Force '5' (Reduce 0) THENM GenRepD ...a)
|\
| ⊢ IsMonHomInj(oal_hgp(s;g);oal_grp(s;g);λz.z)
| |
1 BY (AGenRepD ["basic";"compound"]
| |   THENM Force '5' (All Reduce) ...a)
| |\
| | 3. a1: |oal(s;g↓hgrp)|
| | 4. a2: |oal(s;g↓hgrp)|
| | ⊢ a1 ++ a2 = a1 ++ a2
| | |
1 2 BY (RWH oal_rem_hgrp_of_ocgrpC 0 ...)
| |\
| | ⊢ 00 = 00
| | |
1 2 BY (RWH oal_rem_hgrp_of_ocgrpC 0 ...)
| |\
| | 3. a1: |oal(s;g↓hgrp)|
| | 4. a2: |oal(s;g↓hgrp)|
| | 5. a1 = a2
| | ⊢ a1 = a2
| | |
1 2 BY (BLemma 'oalist_hgrp_eqs' ...)
| |\
| | 3. a1: |oal_hgp(s;g)|
```

```
| | 4. a2: |oal_hgp(s;g)|
| | ⊢ (λz.z) a1 ∈ |oal_grp(s;g)|
| | |
1 2 BY (Force ‘5‘ (Reduce 0) ...)
|  \
|   3. a1: |oal_hgp(s;g)|
|   4. a2: |oal_hgp(s;g)|
|   ⊢ (λz.z) a2 ∈ |oal_grp(s;g)|
|   |
1   BY (Force ‘5‘ (Reduce 0) ...)
|\
| ⊢ RelsIso(|oal(s;g↓hgrp)|;|oal(s;g)|;x,y.↑(x =b y);x,y.↑(x =b y);λz.z)
| |
1 BY (RWN 2 oal_add_hgrp_of_ocgrpC 0
|     THEN Force ‘5‘ (Eval ‘‘rels_iso‘‘ 0) ...)
 \
  ⊢ RelsIso(|oal(s;g↓hgrp)|;|oal(s;g)|;x,y.↑x ≤≤b y;x,y.↑x ≤≤b y;λz.z)
  |
  BY (Force ‘5‘ (Eval ‘‘rels_iso‘‘ 0) ...)
*C oal_omcp_com
               ==========================
               ASSEMBLY OF MONOID COPOWER
               ==========================
*T oal_inj_mon_hom  13.1 sec.
⊢ ∀a:LOSet. ∀b:AbMon. ∀k:|a|.  IsMonHom{b,oal_mon(a;b)}(λv.inj(k,v))
|
BY (Force ‘5‘ (Eval ‘‘monoid_hom_p fun_thru_2op‘‘ 0)
|   THEN GenRepD ...a)
|\
| 1. a: LOSet
| 2. b: AbMon
| 3. k: |a|
| 4. a1: |b|
| 5. a2: |b|
| ⊢ inj(k,a1 * a2) = inj(k,a1) ++ inj(k,a2)
| |
1 BY (BLemma ‘lookups_same_a‘
| |   THENM D 0
| |   THENM RWW "lookup_merge lookup_oal_inj" 0 ...a)
| |
| 6. u: |a|
| ⊢ when k =b u. a1 * a2 = (when k =b u. a1) * (when k =b u. a2)
| |
1 BY (RWW "mon_when_thru_op" 0 ...)
 \
  1. a: LOSet
  2. b: AbMon
  3. k: |a|
  ⊢ inj(k,e) = 00
  |
  BY (BLemma ‘lookups_same_a‘ THENM D 0 ...a)
  |
  4. u: |a|
  ⊢ inj(k,e)[u] = 00[u]
  |
  BY Eval ‘‘oal_nil‘‘ 0
  |  THENM (RWW "lookup_oal_inj" 0 ...a)
```

```
  |
  ⊢ when k =_b u. e = e
  |
  BY (RWW "mon_when_of_id" 0 ...)
*C oal_umap_char_com
               Desparately need here to throw together
               automatic tactics for solving rewrite rule
               antecedents
*T oal_umap_char  166.1 sec.
⊢ ∀s:LOSet. ∀g,h:AbMon. ∀f:|s| ⟶ MonHom(g,h).
|    (λps:|oal(s;g)|. msFor{h} k ∈ dom(ps). f k ps[k]) = !v:|oal(s;g)| ⟶ |h|
|                                                          IsMonHom{oal_mon(s;g),h}(v)
|                                                          ∧ (∀j:|s|. f j = v o (λw.inj(j,w)))
|
BY (RepD ...a)
|
1. s: LOSet
2. g: AbMon
3. h: AbMon
4. f: |s| ⟶ MonHom(g,h)
⊢ (λps:|oal(s;g)|. msFor{h} k ∈ dom(ps). f k ps[k]) = !v:|oal(s;g)| ⟶ |h|
|                                                        IsMonHom{oal_mon(s;g),h}(v)
|                                                        ∧ (∀j:|s|. f j = v o (λw.inj(j,w)))
|
BY % Useful to have properties of f available.
|    Should be able to do this without explicit introduction %
|  (Assert ⌜∀k:|s|. IsMonHom{g,h}(f k)⌝
|   THENA D 0 ...a)
|\
| 5. k: |s|
| ⊢ IsMonHom{g,h}(f k)
| |
1 BY (Assert ⌜f k ∈ MonHom(g,h)⌝
|     THENM AddProperties (-1) ...)
 \
  5. ∀k:|s|. IsMonHom{g,h}(f k)
  |
  BY Eval ``monoid_hom_p fun_thru_2op`` 5
  |
  5. ∀k:|s|. (∀a1,a2:|g|.  f k (a1 * a2) = (f k a1) * (f k a2)) ∧ f k e = e
  |
  BY (Unfold `uni_sat` 0 THEN GenRepD ...a)
  |\
  | ⊢ IsMonHom{oal_mon(s;g),h}(λps:|oal(s;g)|. msFor{h} k ∈ dom(ps). f k ps[k])
  | |
  1 BY (Force `5` (Eval ``monoid_hom_p fun_thru_2op`` 0)
  | |   THENM GenRepD ...a)
  | |\
  | | 6. a1: |oal(s;g)|
  | | 7. a2: |oal(s;g)|
  | | ⊢ msFor{h} k ∈ dom(a1 ++ a2). f k (a1 ++ a2)[k]
  | | |  = (msFor{h} k ∈ dom(a1). f k a1[k]) * (msFor{h} k ∈ dom(a2). f k a2[k])
  | | |
  1 2 BY % First concern is to equalize domains of sums %
  | | |  (RWH (LemmaWithC [`q`,⌜dom(a1) ∪ dom(a2)⌝]
  | | |                   `mset_for_dom_shift`) 0 ...a)
  | | |\
```

```
| | | ⊢ ↑(dom(a1 ++ a2) ⊆_b dom(a1) ∪ dom(a2))
| | | |
1 2 3 BY (BLemma ‘oal_dom_merge‘ ...a)
| | |\
| | | 8. x: |s|
| | | 9. ↑(x ∈_b (dom(a1) ∪ dom(a2)) - dom(a1 ++ a2))
| | | ⊢ f x (a1 ++ a2)[x] = e
| | | |
1 2 3 BY (RWW "lookup_oal_eq_id 5.2" 0 ...)
| | | |
| | | ⊢ ¬↑(x ∈_b dom(a1 ++ a2))
| | | |
1 2 3 BY (RWW "fset_mem_union mset_mem_diff" 9
| | |     THENM RW bool_to_propC 9
| | |     THENM ProveProp ...)
| | |\
| | | ⊢ ↑(dom(a1) ⊆_b dom(a1) ∪ dom(a2))
| | | |
1 2 3 BY (RWW "mem_bsubmset fset_mem_union" 0
| | |     THENM RepD
| | |     THENM RW bool_to_propC 0
| | |     THENM ProveProp ...a)
| | |\
| | | 8. x: |s|
| | | 9. ↑(x ∈_b (dom(a1) ∪ dom(a2)) - dom(a1))
| | | ⊢ f x a1[x] = e
| | | |
1 2 3 BY (RWW "lookup_oal_eq_id 5.2" 0 ...)
| | | |
| | | ⊢ ¬↑(x ∈_b dom(a1))
| | | |
1 2 3 BY (RWW "fset_mem_union mset_mem_diff" 9
| | |     THENM RW bool_to_propC 9
| | |     THENM ProveProp ...)
| | |\
| | | ⊢ ↑(dom(a2) ⊆_b dom(a1) ∪ dom(a2))
| | | |
1 2 3 BY (RWW "mem_bsubmset fset_mem_union" 0
| | |     THENM RepD
| | |     THENM RW bool_to_propC 0
| | |     THENM ProveProp ...a)
| | |\
| | | 8. x: |s|
| | | 9. ↑(x ∈_b (dom(a1) ∪ dom(a2)) - dom(a2))
| | | ⊢ f x a2[x] = e
| | | |
1 2 3 BY (RWW "lookup_oal_eq_id 5.2" 0 ...)
| | | |
| | | ⊢ ¬↑(x ∈_b dom(a2))
| | | |
1 2 3 BY (RWW "fset_mem_union mset_mem_diff" 9
| | |     THENM RW bool_to_propC 9
| | |     THENM ProveProp ...)
| |  \
| |   ⊢ msFor{h} k ∈ dom(a1) ∪ dom(a2). f k (a1 ++ a2)[k]
| |   |  = (msFor{h} k ∈ dom(a1) ∪ dom(a2). f k a1[k])
| |   |    * (msFor{h} k ∈ dom(a1) ∪ dom(a2). f k a2[k])
```

```
| |   |
1 2   BY (RWW "lookup_merge 5.1 mset_for_of_op<" 0  ...)
|  \
|   ⊢ msFor{h} k ∈ dom(00). f k 00[k] = e
|   |
1   BY (Force ‘5‘ (Eval ‘‘oal_nil‘‘ 0) ...)
|\
| 6. j: |s|
| ⊢ f j = (λps:|oal(s;g)|. msFor{h} k ∈ dom(ps). f k ps[k]) o (λw.inj(j,w))
| |
1 BY (ExtWith [‘u’] []
| |   THENM Force ‘5‘ (Reduce 0) ...a)
| |
| 7. u: |g|
| ⊢ f j u = msFor{h} k ∈ dom(inj(j,u)). f k inj(j,u)[k]
| |
1 BY (RWW "lookup_oal_inj oal_dom_inj" 0 ...a)
| |
| ⊢ f j u = msFor{h} k ∈ if u =b e then 0{s} else mset_inj{s}(j) fi . f k (when j =b k. u)
| |
1 BY (SplitOnConclITE ...a)
| |\
| | 8. u = e
| | ⊢ f j u = msFor{h} k ∈ 0{s}. f k (when j =b k. u)
| | |
1 2 BY (Reduce 0
| |     THENM RWW "8 5.2" 0 ...)
|  \
|   8. ¬(u = e)
|   ⊢ f j u = msFor{h} k ∈ mset_inj{s}(j). f k (when j =b k. u)
|   |
1   BY (RWW "mset_for_mset_inj" 0 ...a)
|   |
|   ⊢ f j u = f j (when j =b j. u)
|   |
1   BY (Unfold ‘mon_when‘ 0
|       THEN SplitOnConclITE THENM Try RelRST ...)
 \
  6. a’: |oal(s;g)| ⟶ |h|
  7. IsMonHom{oal_mon(s;g),h}(a’)
  8. ∀j:|s|. f j = a’ o (λw.inj(j,w))
  ⊢ a’ = (λps:|oal(s;g)|. msFor{h} k ∈ dom(ps). f k ps[k])
  |
  BY (ExtWith [‘ps’] []
  |   THENM Reduce 0 ...a)
  |
  9. ps: |oal(s;g)|
  ⊢ a’ ps = msFor{h} k ∈ dom(ps). f k ps[k]
  |
  BY (RWH (FoldWithC [⌜|g|⌝] ‘tlambda‘) 8
  |   THENM RWW "8" 0 THENM Reduce 0 ...)
  |
  8. ∀j:|s|. f j = a’ o (λw:|g|. inj(j,w))
  ⊢ a’ ps = msFor{h} k ∈ dom(ps). a’ inj(k,ps[k])
  |
  BY (RWH oal_monC 6
      THEN RWW "dist_hom_over_mset_for<
```

```
                 oalist_fact<" 0 ...)
*D oal_umap_df umap{<s:s:*>,<g:g:*>}(<h:h:*>,<f:f:*>)== oal_umap{}(<s>; <g>; <h>; <f>)
              umap(<h:h:*>,<f:f:*>)== oal_umap{}(<s>; <g>; <h>; <f>)
*A oal_umap                   umap(h,f) ==  λps:|oal(s;g)|. msFor{h} k ∈ dom(ps). f k ps[k]
*T oal_umap_wf  10.1 sec.
⊢ ∀s:LOSet. ∀g,h:AbMon. ∀f:|s| ⟶ |g| ⟶ |h|.  umap(h,f) ∈ |oal(s;g)| ⟶ |h|
|
BY (Unfold ‘oal_umap‘ 0 ...)
*T oal_umap_char_a  0.0 sec.
⊢ ∀s:LOSet. ∀g,h:AbMon. ∀f:|s| ⟶ MonHom(g,h).
|    umap(h,f) = !v:|oal(s;g)| ⟶ |h|
|                  IsMonHom{oal_mon(s;g),h}(v) ∧ (∀j:|s|. f j = v o (λw.inj(j,w)))
|
BY Unfold ‘oal_umap‘ 0 THEN Lemma ‘oal_umap_char‘
*D oal_mcp_df                 oal_mcp(<s:s:*>;<g:g:*>)== oal_mcp{}(<s>; <g>)
*A oal_mcp     oal_mcp(s;g) ==
                 <oal_mon(s;g), λk,v.inj(k,v), λh,f,ps.msFor{h} k ∈ dom(ps). f k ps[k]>
*T oal_mcp_wf  41.0 sec.
⊢ ∀s:LOSet. ∀g:AbMon.  oal_mcp(s;g) ∈ MCopower(s;g)
|
BY (Unfold ‘oal_mcp‘ 0 ...)
|
1. s: LOSet
2. g: AbMon
⊢ <oal_mon(s;g), λk,v.inj(k,v), λh,f,ps.msFor{h} k ∈ dom(ps). f k ps[k]> ∈ MCopower(s;g)
|
BY (MemTypeCD ...) THEN Force ‘5‘ (Reduce 0)
|\
| ⊢ <oal_mon(s;g), λk,v.inj(k,v), λh,f,ps.msFor{h} k ∈ dom(ps). f k ps[k]> ∈ MCopowerSig(s;g)
| |
1 BY (Unfold ‘mcopower_sig‘ 0 ...)
|\
| 3. j: |s|
| ⊢ IsMonHom{g,oal_mon(s;g)}(λv.inj(j,v))
| |
1 BY (BLemma ‘oal_inj_mon_hom‘ ...)
 \
  3. h: AbMon
  4. f: |s| ⟶ MonHom(g,h)
  ⊢ (λps.msFor{h} k ∈ dom(ps). f k ps[k]) = !v:|oal(s;g)| ⟶ |h|
  |                                          IsMonHom{oal_mon(s;g),h}(v)
  |                                          ∧ (∀j:|s|. f j = v o (λv.inj(j,v)))
  |
  BY (BLemma ‘oal_umap_char‘ ...)
*D oal_omcp_df                oal_omcp{<s:s:*>,<g:g:*>}== oal_omcp{}(<s>; <g>)
*A oal_omcp                   oal_omcp{s,g} ==  <oal_hgp(s;g), λk,v.inj(k,v), λh,f.umap(h,f)>
*T oal_omcp_wf  47.0 sec.
⊢ ∀s:LOSet. ∀g:OGrp.  oal_omcp{s,g} ∈ MCopower(s;g↓hgrp)
|
BY % The proof here needs cleaning up. It is quickly hacked
|    together from an older proof when the oal_umap definition had not
|    been made %
|  (Unfold ‘oal_omcp‘ 0 THENM RepD ...a)
|
1. s: LOSet
2. g: OGrp
⊢ <oal_hgp(s;g), λk,v.inj(k,v), λh,f.umap(h,f)> ∈ MCopower(s;g↓hgrp)
```

```
|
BY (MemTypeCD ...) THEN Force ‘6‘ (Reduce 0)
|\
| ⊢ <oal_hgp(s;g), λk,v.inj(k,v), λh,f.umap(h,f)> ∈ MCopowerSig(s;g↓hgrp)
| |
1 BY (Unfold ‘mcopower_sig‘ 0 ...)
|\
| 3. j: |s|
| ⊢ IsMonHom{g↓hgrp,oal_hgp(s;g)}(λv.inj(j,v))
| |
1 BY (Unfold ‘monoid_hom_p‘ 0
|     THEN RWH oal_hgp_to_monC 0
|     THEN Fold ‘monoid_hom_p‘ 0
|     THEN BLemma ‘oal_inj_mon_hom‘ ...)
 \
  3. h: AbMon
  4. f: |s| ⟶ MonHom(g↓hgrp,h)
  ⊢ umap(h,f) = !v:|oal(s;g↓hgrp)| ⟶ |h|
  |               IsMonHom{oal_hgp(s;g),h}(v) ∧ (∀j:|s|. f j = v o (λv.inj(j,v)))
  |
  BY Unfold ‘monoid_hom_p‘ 0
  |  THEN RWH oal_hgp_to_monC 0
  |  THEN Fold ‘monoid_hom_p‘ 0
  |  THEN RWH hgrp_of_ocgrpC 0
  |
  ⊢ umap(h,f) = !v:|oal(s;g↓hgrp)| ⟶ |h|
  |               IsMonHom{oal_mon(s;g↓hgrp),h}(v) ∧ (∀j:|s|. f j = v o (λv.inj(j,v)))
  |
  BY (BLemma ‘oal_umap_char_a‘ ...)
*C polynom_2_end              **********************************
```