ORDERED A-LIST TYPE DEFINITION

*D before_df before{<a:*>;<u:*>;<ps:*>}== before{}(<a>; <u>; <ps>)
*a before
  before(u;ps) == null(ps) \land (hd(ps) <_b u)
*T before_wf 1 2 \forall a:DSet. \forall ps:|a| List. \forall u:|a|. before(u;ps) \in B
*T comb_for_before_wf 0 0 (\lambda a,ps,u,z.before(u;ps)) \in a:DSet \rightarrow ps:|a| List \rightarrow u:|a| \rightarrow |True \rightarrow B
*M before_eval let before_nilC =
  MacroC 'before_nilC'
  (RepeatC (UnfoldsC 'before null') ANDTHENC ReduceC)
  IdC 'tt';
let before_consC =
  MacroC 'before_consC'
  (RepeatC (UnfoldsC 'before null hd') ANDTHENC ReduceC)
  IdC 'v <_b u';
add_AbReduce_conv 'before'
 (before_nilC ORELSEC before_consC);
*T before_trans 2 2 \forall a:LOSet. \forall u,v:|a|. \forall ws:|a| List. v <_a u \Rightarrow \exists before(v;ws) \Rightarrow \exists before(u;ws)
*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 trivial 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:*)}{(as:as:*)}== sd_ordered{}{(ss:); <as>}
sd_ordered{(as:as:*)}== sd_ordered{}{(ss:); <as>}
*M sd_ordered_ml
sd_ordered(as)
  ==r case as of [] => tt | a::bs => before(a;bs) \land_b sd_ordered(bs) esac
*T sd_ordered_wf 2 0 \forall s:DSet. \forall as:|s| List. sd_ordered(as) \in B
*T comb_for_sd_ordered_wf 0 0 (\lambda s,as,z.sd_ordered(as)) \in s:DSet \rightarrow as:|s| List \rightarrow |True \rightarrow B
*M sd_ordered_eval
let sd_ordered_nilC =
  MacroC 'sd_ordered_nilC'
  (RecUnfoldC 'sd_ordered' ANDTHENC PrimReduceC) [sd_ordered([])]
  IdC 'tt';
let sd_orderedConsC =
  MacroC 'sd_ordered_consC'
  (RecUnfoldC 'sd_ordered' ANDTHENC PrimReduceC) [sd_ordered(a::as)]
  IdC [before(a;as) \land_b sd_ordered(as)];
add_AbReduce_conv 'sd_ordered'
Alternate characterization of strictly-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.

\[ \forall s:QOSet. \forall us:|s| List. sd_ordered(us) = HTFor{\langle b, \land b \rangle \text{ if } \forall b.(\langle s, c \rangle < \langle a, b \rangle) \Rightarrow \exists cs:|s|. \exists a:|s|. \exists cs:|s|. \forall c:|s|. c \neq e \} \]

\[ \forall a:|s|. \forall b:|s|. \forall cs:|s| List. \left( \forall b. x:|s|. x < b \right) \Rightarrow \left( \exists cs:|s|. \exists a:|s|. \exists cs:|s|. \forall c:|s|. c \neq e \right) \]

This is an experimental use of ‘set’ constructors. Proofs are unlikely to be that clean.

Old Definition:
\[ oal(a;b) == \{ps:(a \times \{z:b \downarrow set| \neg (z = e)\}) List| sd_ordered(map(\lambda x.x.1;ps)) \} \]

This gave slower Inclusion proofs because dset a \times \{z:b \downarrow set| \neg (z = e)\} exposed when type checking with funs typed over concrete Lists

**.scrollTop**

**before_imp_before_all**
\[ \forall a:LOSet. \forall b:AbMon. \forall k:|a|. \forall ps:|oal(a;b)|. before(k;map(\lambda z.z.1;ps)) \Rightarrow \left( \exists cs:|s|. \exists a:|s|. \exists cs:|s|. \forall c:|s|. c \neq e \right) \]

**before_all_imp_before**
\[ \forall a:LOSet. \forall b:AbMon. \forall k:|a|. \forall ps:(|a| \times |b|) List. \left( \forall b. x:|s|. x < b \right) \Rightarrow \left( \exists cs:|s|. \exists a:|s|. \exists cs:|s|. \forall c:|s|. c \neq e \right) \]

**nil_in_oalist**
\[ \forall a:LOSet. \forall b:AbMon. \forall ws:|oal(a;b)|. \left( \exists bs:|s|. \exists a:|s|. \exists bs:|s|. \forall c:|s|. c \neq e \right) \]

**cons_in_oalist**
\[ \forall a:LOSet. \forall b:AbMon. \forall ws:|oal(a;b)|. \forall x:|a|. \forall y:|b|. \left( \exists cs:|s|. \exists a:|s|. \exists cs:|s|. \forall c:|s|. c \neq e \right) \]

**oal_nil_cons_com**

-----

**oal_nil_df**
\[ Paren : Prec(preop): 00<|a:|set:*>,<b:|mon:*|>= oal_nil{}(|a|; |b|) 00== oal_nil{}(|a|; |b|) \]

**oal_nil**
\[ 00 == [] \]

**oal_nil_wf**
\[ 00 \forall a:LOSet. \forall b:AbMon. 00 \in |oal(a;b)| \]

**oal_cons_pr_df**
\[ oal_cons_pr(|x:|set:*>,|y:|set:*>,|s:|set:*>,|w:|set:*>)== oal_cons_pr{}(|x|; |y|; |s|; |w|) \]

**oal_cons_pr**
\[ oal_cons_pr(x;y;ws) == \langle x, y|ws \rangle \]

**oal_cons_pr_wf**
\[ 00 \forall a:LOSet. \forall b:AbMon. \forall ws:|oal(a;b)|. \forall x:|a|. \forall y:|b|. \left( \exists cs:|s|. \exists a:|s|. \exists cs:|s|. \forall c:|s|. c \neq e \right) \]

2
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**

\[
\forall a:LOSet. \forall b:AbMon. \forall Q:|a| \times |b| \text{ List } \rightarrow P.
\]

\[
\forall a:LOSet. \forall b:AbMon. \forall Q:|oal(a;b)| \rightarrow P.
\]

\[
\forall a:LOSet. \forall b:AbMon. \forall Q:|oal(a;b)| \rightarrow P.
\]

\[
\forall a:LOSet. \forall b:AbMon. \forall Q:|oal(a;b)| \rightarrow P.
\]

\[
\forall a:LOSet. \forall b:AbMon. \forall Q:|oal(a;b)| \rightarrow P.
\]

\[
\forall a:LOSet. \forall b:AbMon. \forall Q:(|a| \times |b|) \text{ List } \rightarrow P.
\]

\[
\forall a:LOSet. \forall b:AbMon. \forall Q:|oal(a;b)| \rightarrow P.
\]

\[
\forall a:LOSet. \forall b:AbMon. \forall Q:|oal(a;b)| \rightarrow P.
\]

\[
\forall T:U. \forall Q:T \text{ List } \rightarrow T \text{ List } \rightarrow P.
\]
(∀us,vs:T List. ||us|| + ||vs|| < ||ps|| + ||qs|| ⇒ Q[us;vs])
⇒ Q[ps;qs])
⇒ {∀ps,qs:T List. Q[ps;qs]}

*T oalist_pr_length_ind 3 6
Va:LOSSet. ∀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]}

*C oal_fun_fin_sup_char

====================================
CHARACTERIZATION OF OALISTS AS
FUNCTIONS OFFINITE 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:oal:*>,<b:mon:*>)((<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
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 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
let 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 =k k then bv else bs[k] fi
esac

*T lookup_wf
1 1 Va:PosetSig. ∀B:U. ∀z:B. ∀k:|a|. ∀xs:([a] × B) List. xs[k] ∈ B
\[
(A, B, z, k, xs, z_1.xs[k]) \in a: \text{PosetSig} \\
\quad \rightarrow B: \text{U} \\
\quad z: B \\
\quad k: |a| \\
\quad xs: (|a| \times B) \ \text{List} \\
\quad \rightarrow |\text{True} \\
\quad \rightarrow B
\]

\[\begin{align*}
\text{T comb_for_lookup_wf} &\quad 0 \quad 0 \\
(\lambda a, B, z, k, xs, z_1.xs[k]) \in a: \text{PosetSig} &\quad \rightarrow B: \text{U} \\
&\quad \rightarrow z: B \\
&\quad \rightarrow k: |a| \\
&\quad \rightarrow xs: (|a| \times B) \ \text{List} \\
&\quad \rightarrow |\text{True} \\
&\quad \rightarrow B
\end{align*}\]

\[\begin{align*}
\text{T lookup_eval} \quad \text{let} \quad \text{lookup.nilC} &= \\
&\quad \text{MacroC} \ '\text{lookup.nilC}' \\
&\quad (\text{RecUnfoldTopC} \ '\text{lookup'} \ \text{ANDTHENC ReduceC}) \\
&\quad [[k]] \\
&\quad \text{IdC} \ 'z' \\
\text{;;} \\
\text{let} \quad \text{lookup.cons.prC} &= \\
&\quad \text{MacroC} \ '\text{lookup.cons.prC}' \\
&\quad (\text{RecUnfoldTopC} \ '\text{lookup'} \ \text{ANDTHENC ReduceC}) \\
&\quad [[<a, b>::cs[k]]] \\
&\quad \text{IdC} \ ['\text{if } a = b \ k \ \text{then } b \ \text{else } cs[k] \ \text{fi}'] \\
\text{;;} \\
\text{add.AbReduce.conv} \ '\text{lookup'} \\
&\quad (\text{lookup.cons.prC ORELSEC lookup.nilC}) \\
&\quad \% \text{would be interesting to see if ifthenelse makes this work} \\
&\quad \% \text{any faster than if matching used directly.} \\
\text{;;} \\
\text{let} \quad \text{lookup.evalC} \ e \ t = \\
&\quad \text{if is_term} \ '\text{lookup'} \ t \ \text{then} \\
&\quad (\text{lookup.nilC} \ \text{ORELSEC} \\
&\quad \text{(ITECondC lookup.cons.prC (RelRST THEN Auto))}) \\
&\quad \text{e } t \\
\text{else} \\
&\quad \text{failwith 'lookup.evalC'} \\
\text{;;} \\
\text{T lookup.oal.eval} \quad \text{DCL:} \quad \text{lookup.oal.nil: } 00[k] \sim \\
\text{;;} \\
\text{T lookup_fails} &\quad 3 \quad 4 \\
\forall a: \text{DSet}. \forall B: \text{U}. \forall z: B. \forall k: |a|. \forall ps: (|a| \times B) \ \text{List}.
&\quad \neg \uparrow(k \in_b \text{map}(\lambda x.x.1; ps)) \Rightarrow ps[k] = z \\
\text{T lookup_non_zero} &\quad 4 \quad 5 \\
\forall a: \text{LOSet}. \forall b: \text{AbMon}. \forall k: |a|. \forall ps: (\text{oal}(a;b)).
&\quad \neg (ps[k] = e) \iff \uparrow(k \in_b \text{dom}(ps)) \\
\text{T lookup_oal_eq_id} &\quad 1 \quad 2 \\
\forall a: \text{LOSet}. \forall b: \text{AbMon}. \forall k: |a|. \forall ps: (\text{oal}(a;b)).
&\quad \neg \uparrow(k \in_b \text{dom}(ps)) \Rightarrow ps[k] = e \\
\text{T lookup_oal_cons} &\quad 2 \quad 4 \\
\forall a: \text{LOSet}. \forall b: \text{OCMon}. \forall k, kp: |a|. \forall vp: |b|. \forall ps: \text{oal}(a;b).
&\quad \uparrow(\text{before}(kp; \text{map}((\lambda z.1; ps))) \Rightarrow (<kp, vp :: ps)[k] = (\text{when } kp =_b k. \ vp) \ast ps[k]) \\
\text{;;} \\
\text{T lookup_before_start_com} \\
\text{lookup_before_start_a was} \\
\text{a lot easier to prove. Shows the benefit} \\
of using a 'stronger' notion of beforeness. \\
\text{T lookup_before_start} &\quad 3 \quad 6 \\
\forall a: \text{LOSet}. \forall b: \text{AbMon}. \forall k: |a|. \forall ps: \text{oal}(a;b).
&\quad \uparrow(\text{before}(k; \text{map}(\lambda z.1; ps))) \Rightarrow ps[k] = e \\
\text{T lookup_before_start_a} &\quad 2 \quad 4 \\
\forall a: \text{QOSet}. \forall b: \text{AbMon}. \forall k: |a|. \forall ps: (|a| \times |b|) \ \text{List}.
&\quad \uparrow(\forall k' |a|) \in \text{map}(\lambda z.1; ps). k' <_b k \Rightarrow ps[k] = e \\
\text{T lookups_same} &\quad 5 \quad 7 \\
\forall a: \text{LOSet}. \forall b: \text{AbMon}. \forall ps, qs: \text{oal}(a;b).
&\quad (\forall u: |a|. ps[u] = qs[u]) \Rightarrow ps = qs
∀a:LOSet. ∀b:AbMon. ∀ps,qs:|oal(a;b)|. (∀u:|a|. ps[u] = qs[u]) ⇒ ps = qs

∀a:LOSet. ∀b:AbMon. ∀ps,qs:|oal(a;b)|. ps = qs ⇐⇒ (∀u:|a|. ps[u] = qs[u])

OALIST MERGE FUNCTION

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?

Parentheses ::Prec(inop)::-
== oal_merge{}(<a>; <b>; <ps>; <qs>)

Parentheses ::Prec(inop)::-
<ps:ps:L> ++ <qs:qs:E>
== oal_merge{}(<a>; <b>; <ps>; <qs>)

let oal_merge_left_nilC =
MacroC ‘oal_merge_left_nilC’
(RecUnfoldC ‘oal_merge‘ ANDTHENC ReduceC)
[|] ++ qs
IdC
[q] ;;

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 \((vp \times vq) =_b e\) then ps ++ qs
else \(<kp, vp \times vq>::(ps ++ qs)\)
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 4 7
\(\forall a:LOSet. \forall b:AbMon. \forall ps,qs:(|a| \times |b|) List. ps ++ qs \in (|a| \times |b|) List\)
*T oal_merge_dom_pred 5 7
\(\forall a:LOSet. \forall b:AbMon. \forall Q:|a| \rightarrow B. \forall ps,qs:(|a| \times |b|) List.
\uparrow(\forall x:(|a|) \in map(\lambda x.x.1;ps). Q[x])
\Rightarrow \uparrow(\forall x:(|a|) \in map(\lambda x.x.1;qs). Q[x])
\Rightarrow \uparrow(\forall x:(|a|) \in map(\lambda x.x.1;ps ++ qs). Q[x])\)
*T oal_dom_merge 3 5
\(\forall a:LOSet. \forall b:AbMon. \forall ps,qs:|oal(a;b)|. \uparrow(dom(ps ++ qs) \subseteq_b dom(ps) \cup dom(qs))\)
*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 5 7
\(\forall a:LOSet. \forall b:AbMon. \forall ps,qs:(|a| \times |b|) List.
\uparrow sd_ordered(map(\lambda x.x.1;ps))
\Rightarrow \uparrow sd_ordered(map(\lambda x.x.1;qs))
\Rightarrow \uparrow sd_ordered(map(\lambda x.x.1;ps ++ qs))\)
*T oal_merge_non_id_vals 5 8
\(\forall a:LOSet. \forall b:AbMon. \forall ps,qs:(|a| \times |b|) List.
\neg\uparrow(e \in_b map(\lambda x.x.2;ps))
\Rightarrow \neg\uparrow(e \in_b map(\lambda x.x.2;qs))
\Rightarrow \neg\uparrow(e \in_b map(\lambda x.x.2;ps ++ qs))\)
*T lookup_merge 5 8
\(\forall a:LOSet. \forall b:AbMon. \forall k:|a|. \forall ps,qs:|oal(a;b)|. (ps ++ qs)[k] = ps[k] \times qs[k]\)
*T oal_merge_wf2 2 3 \(\forall a:LOSet. \forall b:AbMon. \forall ps,qs:|oal(a;b)|. ps ++ qs \in |oal(a;b)|\)
*C oal_mon_com ==============
OALIST MONOID DEFINITION
=============
*T oal_nil_ident_r 3 4 \(\forall a:LOSet. \forall b:AbMon. \forall ps:|oal(a;b)|. ps ++ 00 = ps\)
*T oal_nil_ident_l 1 2 \(\forall a:LOSet. \forall b:AbMon. \forall ps:|oal(a;b)|. 00 ++ ps = ps\)
∀ a:LOSet. ∀ b:AbMon. ∀ ps, qs:|oal(a;b)|. ps ++ qs = qs ++ ps

∀ a:LOSet. ∀ b:AbMon. ∀ ps, qs, rs:|oal(a;b)|. ps ++ qs ++ rs = (ps ++ qs) ++ rs

∀ a:LOSet. ∀ b:AbMon. ∀ ps, qs:|oal(a;b)|. ps ++ qs = qs ++ ps

∀ a:LOSet. ∀ b:AbMon. oal_mon(a;b) == <|oal(a;b)|, =b, λx,y.tt, λx,y.x ++ y, 00, λx.x>

∀ a:LOSet. ∀ b:AbMon. oal_mon(a;b) ∈ AbMon

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)

let oal_injC =
  let inj{<a:a:*>,<b:b:*>}(<k:k:*>,<v:v:*>) = oal_inj{}(<a>; <b>; <k>; <v>)
  in
    inj{<a:a:*>,<b:b:*>}(<k:k:*>,<v:v:*>)

let oalist_fact =
  let ps = msFor{oal_mon(a;b)} k'
    ∈ dom(ps).
    inj(k', ps[k'])

let oal_negC =
  let oal_neg-nilC =
    MacroC 'oal_neg-nilC' (EvalC ''oal_neg'')
    [[[]]]
  IdC

let oal_neg-nilC =
  MacroC 'oal_neg-nilC'
  (EvalC ''oal_neg-nilC'')
  [[[]]]

let oal_neg-nilC =
  MacroC 'oal_neg-nilC'
  (EvalC ''oal_neg-nilC'')
  [[[]]]
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 1 1
∀a:PosetSig. ∀b:GrpSig. ∀ps:(|a| × |b|) List. map(λz.z.1;--ps) = map(λz.z.1;ps)

*T oal_neg_sd_ordered 1 2
∀a:LOSet. ∀b:AbMon. ∀ps:|a| × |b| List. 
↑sd_ordered(map(λx.x.1;ps)) ⇒ ↑sd_ordered(map(λx.x.1;--ps))

*T oal_neg_non_id_vals 3 4
∀a:LOSet. ∀b:AbGrp. ∀ps:|oal(a;b)|. ¬(e ∈ b map(λx.x.2;ps)) ⇒ ¬(e ∈ b map(λx.x.2;--ps))

*T oal_neg_wf2 2 3
∀a:LOSet. ∀b:AbGrp. ∀ps:|oal(a;b)|. --ps ∈ |oal(a;b)|

*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 0 2
∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|. null(ps) ∈ B

*T assert_of_oal_null 3 3
∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|. ↑null(ps) ≜ ps = 00

*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 2 4
∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|. ¬(ps = 00) ⇒ lk(ps) ∈ |s|

*T oal_lk_in_dom 3 4
∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|. ¬(ps = 00) ⇒ ↑(lk(ps) ∈ |s| dom(ps))

*T oal_lk_bounds_dom 3 5
∀s:LOSet. ∀g:AbMon. ∀k:|s|. ∀ps:|oal(s;g)|. 
¬(ps = 00) ⇒ ↑(k ∈ |s| dom(ps)) ⇒ k ≤ lk(ps)

*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

    2 4 ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|. ¬(ps = 00) ⇒ lv(ps) ∈ |g|

*T oal_lv_nid

    2 3 ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|. ¬(ps = 00) ⇒ ¬(lv(ps) = e)

*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

    4 6 ∀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)

*T oal_lk_merge_2

    4 6 ∀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)

*C oal_lk_neg_com

    With partial functions there is always the question
    of how explicit to make all the totality guaranteeing
    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

    3 4 ∀s:LOSet. ∀g:AbGrp. ∀ps:|oal(s;g)|.

    ¬(ps = 00) ⇒ lk(--ps) = lk(ps)

*T lookup_oal_lk

    2 4 ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|.

    ¬(ps = 00) ⇒ ps[lk(ps)] = lv(ps)

*T oal_lv_neg

    2 4 ∀s:LOSet. ∀g:AbGrp. ∀ps:|oal(s;g)|.

    ¬(ps = 00) ⇒ lv(--ps) = lv(ps)

*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>)

*A oal_bpos

    pos(ps) == ¬null(ps) ∧ b e <b lv(ps)

*T oal_bpos_wf

    2 3 ∀s:LOSet. ∀g:AbMon. ∀ps:|oal(s;g)|. pos(ps) ∈ B

*T comb_for_oal_bpos wf

    0 0 (λs,g,ps,z.pos(ps)) ∈ s:LOSet → g:AbMon → ps:|oal(s;g)| → ↓True → B
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.

INTRODUCTION OF OALIST GROUP

---

Introduction of oal_list group

---

This is not ideally placed. Definition had to wait
for introduction of order function.
However, characterization as ordered group comes later.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.
*T oal_lt_irrefl 2 2 ∀s:LOSet. ∀g:OCMon. Irrefl(|oal(s;g)|;ps,qs.ps << qs)
*T oal_lt_trans 3 6 ∀s:LOSet. ∀g:OCMon. Trans(|oal(s;g)|;ps,qs.ps << qs)
*T oal_bpos_trichot 3 5 ∀s:LOSet. ∀g:OCMon. Irrefl(|oal(s;g)|;ps,qs.ps << qs)

*T oal_lt_trans 3 6 ∀s:LOSet. ∀g:OCMon. Trans(|oal(s;g)|;ps,qs.ps << qs)

*T oal_lt_trichot 3 6 ∀s:LOSet. ∀g:OCMon. Irrefl(|oal(s;g)|;ps,qs.ps << qs)

*T oal_bpos_trichot 3 5 ∀s:LOSet. ∀g:OCMon. Irrefl(|oal(s;g)|;ps,qs.ps << qs)

*T oal_merge_preserves_lt 2 5 ∀s:LOSet. ∀g:OCMon. ∀ps,qs,rs:|oal(s;g)|. ps << qs ⇒ ps ++ qs << ps ++ rs
*M oal_le_is_order 3 3 ∀s:LOSet. ∀g:OCMon. Order(|oal(s;g)|;ps,qs.ps ≤ {s,g} qs)
*T oal_merge_preserves_le 1 4 ∀s:LOSet. ∀g:OCGrp. ∀ps,qs,rs:|oal(s;g)|. ps ≤ {s,g} qs ⇒ ps ++ qs ≤ {s,g} ps ++ rs

*T oal_merge_preserves_le 1 4 ∀s:LOSet. ∀g:OCGrp. ∀ps,qs,rs:|oal(s;g)|. ps ≤ {s,g} qs ⇒ ps ++ qs ≤ {s,g} ps ++ rs

*C oal_hgp_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.

*M oal_hgp_ml let oal_hgpC =

FirstC (map (\t1,t2.MacroC 'oal_hgrpC' IdC t1 (ForceReduceC '5') t2)
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 (λ t1,t2.
           DoubleMacroC 'add_oal_hgp' C t1 C t2)
        [[|=b|],[|=b|],
         [[ps ++ qs],[ps ++ qs]],[[00],[00]]))

let oal_hgp_ml2 let oal_hgp_to_monC,oal_mon_to_hgpC =
let C1 = AbRedexC in
let C2 = AbRedexC in
(FirstC # FirstC )
  (unzip
    (map (λ t1,t2. DoubleMacroC 'oal_hgp_to_monC' C1 t1 C2 t2)
        [[|oal_hgp(s;g)|],[|oal_mon(s;g)\downarrow hgrp)|],
         [[|=b|],[|=b|],
         [[*],[*]],
         [[e],[e]]))

let 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'.

let oalist_hgrp_eqs 4 5 ∀s:LOSet. ∀g:OGrp. ∀a1,a2:|oal(s;g)\downarrow hgrp)|. a1 = a2 ⇒ a1 = a2
let oal_hgp_wf2 3 5 ∀s:LOSet. ∀g:OGrp. oal_hgp(s;g) ∈ OCMon

let oal_omcp_com

let oal_inj_mon_hom 3 3 ∀s:LOSet. ∀b:AbMon. ∀k:|a|. IsMonHom{b,oal_mon(a;b)}(λ v.inj(k,v))

let oal_omap_char_com
Desperately need here to throw together
automatic tactics for solving rewrite rule
antecedents

let oal_omap_char 5 7 ∀s:LOSet. ∀g,h:AbMon. ∀f:|s| → MonHom(g,h).
(λ ps:oal(s;g)). msFor{h} k ∈ dom(ps)
\[ f(k ps[k]) = !v:|\text{oal}(s;g)| \rightarrow |h| \]
\[
\text{IsMonHom}(\text{oal}_\text{mon}(s;g),h)(v) \wedge (\forall j:|s|. f(j) = v \circ (\lambda w.\text{inj}(j,w)))
\]

*D oal_\text{umap}_\text{df} umap\langle s:s:*,g:g:*\rangle\langle h:h:*,f:f:*\rangle = oal_\text{umap}\langle s; g; h; f \rangle

*A oal_\text{umap} umap(h,f) = \lambda ps:|\text{oal}(s;g)|. \text{msFor}(h) k \in \text{dom}(ps). f(k ps[k])

*T oal_\text{umap}_\text{wf} 0 3

\forall s:|OSet. \forall g,h:|\text{AbMon}. \forall f:|s| \rightarrow |g| \rightarrow |h|. umap(h,f) \in |\text{oal}(s;g)| \rightarrow |h|

*T oal_\text{umap}_\text{char}_\text{a} 0 0

\forall s:|OSet. \forall g,h:|\text{AbMon}. \forall f:|s| \rightarrow |\text{MonHom}(g,h)|.

\text{umap}(h,f) = !v:|\text{oal}(s;g)| \rightarrow |h|
\text{IsMonHom}(\text{oal}_\text{mon}(s;g),h)(v) \wedge (\forall j:|s|. f(j) = v \circ (\lambda w.\text{inj}(j,w)))

*D oal_\text{omcp}_\text{df} oal_\text{omcp}\langle s:s:*,g:g:*\rangle = oal_\text{omcp}\langle s; g \rangle

*A oal_\text{omcp} oal_\text{omcp}(s,g) = <\text{oal}_\text{hgp}(s;g), \lambda k,v.\text{inj}(k,v), \lambda h,f.\text{umap}(h,f)>

*T oal_\text{omcp}_\text{wf} 3 5

\forall s:|OSet. \forall g:|\text{OGrp}. oal_\text{omcp}(s,g) \in |\text{MCopower}(s;g)| \downarrow hgrp

*C polynom_2_end ***********************************

Thm stats: <log2 (# pscript lines)> <log2 (expansion time in sec)>