Online appendix for the paper

Evaluation of the Implementation of an Abstract Interpretation Algorithm using Tabled CLP [∗]

published in Theory and Practice of Logic Programming

JOAQUÍN ARIAS and MANUEL CARRO
IMDEA Software Institute and Universidad Politécnica de Madrid
joaquin.arias@{imdea.org,alumnos.upm.es}, manuel.carro@{imdea.org,upm.es}

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Appendix A PLAI Algorithm Implementation Using TCLP

In this appendix we include the code corresponding to the reimplementation of PLAI using TCLP. It is not expected to be used to understand the code (we did not add any facility or improve its functionality), but rather to compare the code length and complexity with that of the original PLAI in CiaoPP, which we include in Appendix B. Therefore, we have removed the comments that appear in the original files. The files with comments can be accessed at http://www.cliplab.org/papers/tclp-plai-iclp2019.

```
/*              Copyright (C)1990-2019 UPM-CLIP                           */

:- module(fixpo_plai_tabling,
        [
            query/8,
            init_fixpoint/0,
            cleanup_fixpoint/1,
            entry_to_exit/9
        ],
        [assertions, datafacts]).

% Ciao library
:- use_module(engine(io_basic)).

:- use_module(library(aggregates), [bagof/3, (^)/2]).
:- use_module(library(lists), [member/2, append/3]).
:- use_module(library(terms_vars), [varset/2]).
:- use_module(library(terms_check)).
:- use_module(library(sets), [merge/3, ord_subtract/3]).
:- use_module(library(sort), [sort/2]).
:- use_module(library(messages)).
:- use_module(library(write)).

% CiaoPP library
:- use_module(ciaopp(preprocess_flags), [current_pp_flag/2, set_pp_flag/2]).

```

∗ Work partially supported by EIT Digital (https://eitdigital.eu), MINECO project TIN2015-67522-C3-1-R (TRACES), and Comunidad de Madrid project S2018/TCS-4339 BLOQUES-CM co-funded by EIE Funds of the European Union.

```
% Plai library
:- use_module(ciaopp(plai/fixpo_ops), [inexistent/2, variable/2, bottom/1,
        singleton/2, fixpoint_id_reuse_prev/5, fixpoint_id/1, fixp_id/1,
        each_abs_sort/3,
        % each_concrete/4,
        each_extend/6, each_project/6, each_exit_to_prime/8, each_unknown_call/4,
        each_body_succ_builtin/12, body_succ_meta/7, reduce_equivalent/3,
        each_apply_trusted/7,   widen_succ/4,   decide_memo/6,clause_applies/2,
        abs_subset_/3]).

:- use_module(ciaopp(plai/domains)).
:- use_module(ciaopp(plai/trace_fixp), [fixpoint_trace/7, cleanup/0]).
:- use_module(ciaopp(plai/plai_db),
        [ complete/7, memo_call/5, memo_table/6, cleanup_plai_db/1, patch_parents/6 ]).
:- use_module(ciaopp(plai/psets), [update_if_member_idlist/3]).
:- use_module(ciaopp(plai/re_analysis), [erase_previous_memo_tables_and_parents/4]).
:- use_module(ciaopp(plai/transform), [body_info0/4, trans_clause/3]).
:- use_module(ciaopp(plai/apply_assertions_old),
        [ apply_trusted0/7,
          cleanup_trusts/1 ]).

:- doc(author,"Joaquin Arias").

:- doc(module,"This module adapts the implementation of the top-down
        fixpoint algorithm of PLAI using TCLP with aggregates and an
        extension that also checks call entailment.").

init_fixpoint.

cleanup_fixpoint(_AbsInt).

%-------------------------------------------------------------------------%
% call_to_success(+,+,+,+,+,+,-)                                          %
%-------------------------------------------------------------------------%

call_to_success(SgKey,Call,Proj,Sg,Sv,AbsInt,Succ) :-
        call_to_success_fixpoint(SgKey,Sg, st(Sv,Call,Proj,AbsInt,Prime)),
        each_extend(Sg,Prime,AbsInt,Sv,Call,Succ).

%%%%%%%%%%%% TCLP interface %%%%%%%%%%%%
 :- use_package(tclp_aggregate).
 :- table call_to_success_fixpoint(_,_,abst_lub).

call_entail(abst_lub, st(V,_,ProjA,AbsInt,_), st(V,_,ProjB,AbsInt,_)) :-
        identical_abstract(AbsInt,ProjA,ProjB), !.

answer_entail(abst_lub, st(V,_,_,AbsInt,PrimeAs), st(V,_,_,AbsInt,PrimeBs),1) :-
        singleton(PrimeA,PrimeAs),
        singleton(PrimeB,PrimeBs),
        less_or_equal(AbsInt,PrimeA,PrimeB), !.

answer_join(abst_lub,st(V,_,_,AbsInt,PrimeAs), st(V,_,_,AbsInt,PrimeBs),
                                          st(V,_,_,AbsInt,PrimeNews)) :-
        singleton(PrimeA,PrimeAs),
        singleton(PrimeB,PrimeBs),
        singleton(PrimeNew,PrimeNews),
        compute_lub(AbsInt,[PrimeA,PrimeB],PrimeNew), !.

apply_answer(abst_lub, st(V,_,_,Ab,A), st(V,_,_,Ab,B)) :- A = B.

call_to_success_fixpoint(SgKey,Sg,st(Sv,Call,Proj,AbsInt,Primes)) :-
        trans_clause(SgKey,_,Clause),
        do_nr_cl(Clause,Sg,Sv,Call,Proj,AbsInt,Primes).
call_to_success_fixpoint(SgKey,Sg,st(Sv,_Call,Proj,AbsInt,Primes)) :-
        \+ trans_clause(SgKey,_,_),
        apply_trusted0(Proj,SgKey,Sg,Sv,AbsInt,_ClId,Prime),
        singleton(Prime,Primes).

do_nr_cl(Clause,Sg,Sv,Call,Proj,AbsInt,Primes):-
        Clause = clause(Head,Vars_u,K,Body),
        clause_applies(Head,Sg), !,
        varset(Head,Hv),
        sort(Vars_u,Vars),
        ord_subtract(Vars,Hv,Fv),
        process_body(Body,K,AbsInt,Sg,Hv,Fv,Vars_u,Head,Sv,Call,
                                                Proj,Primes,_Id).
do_nr_cl(_Clause,_Sg,_Sv,_Call,_Proj,_AbsInt,[[]]).

process_body(Body,K,AbsInt,Sg,Hv,_Fv,_,Head,Sv,Call,Proj,LPrime,_Id):-
        Body = g(_,[],'$built'(_,true,_),'true/0',true), !,
        singleton(Prime,LPrime),
        call_to_success_fact(AbsInt,Sg,Hv,Head,K,Sv,Call,Proj,Prime,_Succ).
process_body(Body,K,AbsInt,Sg,Hv,Fv,Vars_u,Head,Sv,_,Proj,Prime,Id):-
        call_to_entry(AbsInt,Sv,Sg,Hv,Head,K,Fv,Proj,Entry,ExtraInfo),
        singleton(Entry,LEntry),
        entry_to_exit(Body,K,LEntry,Exit,[],_,Vars_u,AbsInt,Id),
        each_exit_to_prime(Exit,AbsInt,Sg,Hv,Head,Sv,ExtraInfo,Prime).

%-----------------------------------------------------------------------%
% entry_to_exit(+,+,+,-,+,-,+,+,+)                                      %
%-----------------------------------------------------------------------%

entry_to_exit((Sg,Rest),K,Call,Exit,OldList,NewList,Vars_u,AbsInt,NewN):- !,
        body_succ(Call,Sg,Succ,OldList,IntList,Vars_u,AbsInt,K,NewN,_),
        entry_to_exit(Rest,K,Succ,Exit,IntList,NewList,Vars_u,AbsInt,NewN).
entry_to_exit(true,_,Call,Call,List,List,_,_,_):- !.
entry_to_exit(Sg,Key,Call,Exit,OldList,NewList,Vars_u,AbsInt,NewN):-
        body_succ(Call,Sg,Exit,OldList,NewList,Vars_u,AbsInt,Key,NewN,_),
        true.

body_succ(Call,_Atom,Succ,List,List,_HvFv_u,_AbsInt,_ClId,_ParentId,no):-
        bottom(Call), !,
        Succ = Call.
body_succ(Call,Atom,Succ,List,NewList,HvFv_u,AbsInt,ClId,ParentId,Id):-
        Atom=g(Key,Sv,Info,SgKey,Sg),
        body_succ_(Info,SgKey,Sg,Sv,HvFv_u,Call,Succ,List,NewList,AbsInt,
                   ClId,Key,ParentId,Id).

body_succ_(Info,SgKey,Sg,Sv,HFv,Call,Succ,L,NewL,AbsInt,ClId,Key,PId,Id):-
        Info = [_|_], !,
        split_combined_domain(AbsInt,Call,Calls,Domains),
        map_body_succ(Info,SgKey,Sg,Sv,HFv,Calls,Succs,L,NewL,Domains,
                      ClId,Key,PId,Id),
        split_combined_domain(AbsInt,Succ,Succs,Domains).
body_succ_(Info,SgKey,Sg,Sv,HFv,Call,Succ,L,NewL,AbsInt,ClId,Key,PId,Id):-
        body_succ0(Info,SgKey,Sg,Sv,HFv,Call,Succ,L,NewL,AbsInt,
                   ClId,Key,PId,Id).

map_body_succ([],_SgKey,_Sg,_Sv,_HFv,[],[],L,L,[],_ClId,_Key,_PId,no).
map_body_succ([I|Info],SgKey,Sg,Sv,HFv,[Call|Calls],[Succ|Succs],L,NewL,
              [AbsInt|Domains],ClId,Key,PId,Id):-
        body_succ0(I,SgKey,Sg,Sv,HFv,Call,Succ,L,_NewL,AbsInt,
                   ClId,Key,PId,_Id), !,
        map_body_succ(Info,SgKey,Sg,Sv,HFv,Calls,Succs,L,NewL,Domains,
                   ClId,Key,PId,Id).

body_succ0('$var',SgKey,Sg,_Sv_u,HvFv_u,Calls,Succs,List0,List,AbsInt,
                 _ClId,F,_N,_Id):-
        !,
        ( Calls=[Call],
          concrete(AbsInt,Sg,Call,Concretes),
          concretes_to_body(Concretes,SgKey,AbsInt,B)
        -> meta_call(B,HvFv_u,Calls,[],Succs,List0,List,AbsInt,_ClId,_Id,_Ids)
         ; List=List0,
           each_unknown_call(Calls,AbsInt,[Sg],Succs) % Sg is a variable
        ).
body_succ0('$meta'(T,B,_),SgKey,Sg,Sv_u,HvFv_u,Call,Succ,List0,List,AbsInt,
            _ClId,_F,_N,_Id):-
        !,
        meta_call(B,HvFv_u,Call,[],Exits,List0,List,AbsInt,ClId,Id,_Ids),
        ( body_succ_meta(T,AbsInt,Sv_u,HvFv_u,Call,Exits,Succ) ->
          true
        ; % for the trusts, if any:
          varset(Sg,Sv_r), % Sv_u contains extra vars (from meta-term)
                           % which will confuse apply_trusted
          body_succ0(nr,SgKey,Sg,Sv_r,HvFv_u,Call,Succ,[],_List,AbsInt,
                        _ClId,_F,_N,_Id0)
        ).
body_succ0('$built'(T,Tg,Vs),SgKey,Sg,Sv_u,HvFv_u,Call,Succ,List0,List,AbsInt,
            _ClId,_F,_N,_Id):-
        !,
        List=List0,
        sort(Sv_u,Sv),
        each_body_succ_builtin_(Call,AbsInt,T,Tg,Vs,SgKey,Sg,Sv,HvFv_u,Succ).
body_succ0(_RFlag,SgKey,Sg,Sv_u,HvFv_u,Call,Succ,_List0,_List,AbsInt,
           _ClId,_F,_N,_Id):-
        sort(Sv_u,Sv),
        each_call_to_success(Call,SgKey,Sg,Sv,HvFv_u,AbsInt,Succ).

%% predicate adapted from fixpo_ops
each_body_succ_builtin_([],_,_T,_Tg,_,_,_Sg,_Sv,_HvFv_u,[]).
each_body_succ_builtin_([Call|Calls],AbsInt,T,Tg,Vs,SgKey,Sg,Sv,HvFv_u,[Succ|Succs]):-
        project(AbsInt,Sg,Sv,HvFv_u,Call,Proj),
        body_succ_builtin(T,AbsInt,Tg,Vs,Sv,HvFv_u,Call,Proj,Succ),!, %% Doamin call
        each_body_succ_builtin_tabling_(Calls,AbsInt,T,Tg,Vs,SgKey,Sg,Sv,HvFv_u,Succs).

each_call_to_success([Call],SgKey,Sg,Sv,HvFv_u,AbsInt,Succ):-
        !,
        project(AbsInt,Sg,Sv,HvFv_u,Call,Proj),
        call_to_success(SgKey,Call,Proj,Sg,Sv,AbsInt,Succ).

each_call_to_success(LCall,SgKey,Sg,Sv,HvFv_u,AbsInt,LSucc):-
        each_call_to_success0(LCall,SgKey,Sg,Sv,HvFv_u,AbsInt,
                              LSucc).

each_call_to_success0([],_SgK,_Sg,_Sv,_HvFv,_AbsInt,[]).
each_call_to_success0([Call|LCall],SgKey,Sg,Sv,HvFv_u,AbsInt,
                      LSucc):-
        project(AbsInt,Sg,Sv,HvFv_u,Call,Proj),
        call_to_success(SgKey,Call,Proj,Sg,Sv,AbsInt,LSucc0),
        append(LSucc0,LSucc1,LSucc),
        each_call_to_success0(LCall,SgKey,Sg,Sv,HvFv_u,AbsInt,
                              LSucc1).

meta_call([],_HvFv_u,Call,[],Call,List,List,_AbsInt,_ClId,_Id,[]).
meta_call([Body|Bodies],HvFv_u,Call,Succ0,Succ,L0,List,AbsInt,ClId,Id,Ids):-
        meta_call_([Body|Bodies],HvFv_u,Call,Succ0,Succ,L0,List,AbsInt,ClId,Id,Ids).
meta_call_([Body|Bodies],HvFv_u,Call,Succ0,Succ,L0,List,AbsInt,ClId,Id,Ids):-
        meta_call_body(Body,ClId,Call,Succ1,L0,L1,HvFv_u,AbsInt,Id,Ids0),
        widen_succ(AbsInt,Succ0,Succ1,Succ2),
        append(Succ0,Succ1,Succ2),
        append(Ids0,Ids1,Ids),
        meta_call_(Bodies,HvFv_u,Call,Succ2,Succ,L1,List,AbsInt,ClId,Id,Ids1).
meta_call_([],_HvFv_u,_Call,Succ,Succ,List,List,_AbsInt,_ClId,_Id,[]).

meta_call_body((Sg,Rest),K,Call,Exit,OldList,NewList,Vars_u,AbsInt,PId,CIds):-
        !,
        CIds=[Id|Ids],
        body_succ(Call,Sg,Succ,OldList,IntList,Vars_u,AbsInt,K,PId,Id),
        meta_call_body(Rest,K,Succ,Exit,IntList,NewList,Vars_u,AbsInt,PId,Ids).
meta_call_body(true,_,Call,Call,List,List,_,_,_,[no]):- !.
meta_call_body(Sg,Key,Call,Exit,OldList,NewList,Vars_u,AbsInt,PId,[Id]):-
        body_succ(Call,Sg,Exit,OldList,NewList,Vars_u,AbsInt,Key,PId,Id).

concretes_to_body([],_SgKey,_AbsInt,[]).
concretes_to_body([Sg|Sgs],SgKey,AbsInt,[B|Bs]):-
        body_info0(Sg:SgKey,[],AbsInt,B),
        concretes_to_body(Sgs,SgKey,AbsInt,Bs).

%-------------------------------------------------------------------------%
% query(+,+,+,+,+,+,+,-)                                                  %
%-------------------------------------------------------------------------%

:- doc(query(AbsInt,QKey,Query,Qv,RFlag,N,Call,Succ),
        "The success pattern of @var{Query} with @var{Call} is
        @var{Succ} in the analysis domain @var{AbsInt}. The predicate
        called is identified by @var{QKey}. The goal @var{Query} has
        variables @var{Qv}.").

query(AbsInt,QKey,Query,Qv,_RFlag,_N,Call,Succ) :-
        project(AbsInt,Query,Qv,Qv,Call,Proj),
        call_to_success(QKey,Call,Proj,Query,Qv,AbsInt,Succ), !.

query(_AbsInt,_QKey,_Query,_Qv,_RFlag,_N,_Call,_Succ):-
        % should never happen, but...
        error_message("SOMETHING HAS FAILED!").
```

Appendix B PLAI Algorithm Implementation in Ciao Prolog

We include here the Ciao Prolog implementation of PLAI. As mentioned before, we have removed the comments from the file since the goal of this appendix it to make it easier for the reader to compare the Ciao Prolog code w.r.t. the code using TCLP, which we include in Appendix A. The original version is available at `http://www.cliplab.org/papers/tclp-plai-iclp2019`.

```
/*             Copyright (C)1990-2019 UPM-CLIP                          */

:- module(fixpo_plai_with_comments,
        [ query/8,
          init_fixpoint/0,
          cleanup_fixpoint/1,
          entry_to_exit/9
        ],
        [assertions, datafacts]).

% Ciao library
:- use_module(library(aggregates), [bagof/3, (^)/2]).
:- use_module(library(lists), [member/2, append/3]).
:- use_module(library(terms_vars), [varset/2]).
:- use_module(library(sets), [merge/3, ord_subtract/3]).
:- use_module(library(sort), [sort/2]).
:- use_module(library(messages)).

% CiaoPP library
:- use_module(ciaopp(preprocess_flags), [current_pp_flag/2, set_pp_flag/2]).

% Plai library
:- use_module(ciaopp(plai/fixpo_ops), [inexistent/2, variable/2, bottom/1,
        singleton/2, fixpoint_id_reuse_prev/5, fixpoint_id/1, fixp_id/1,
        each_abs_sort/3,
        each_extend/6, each_project/6, each_exit_to_prime/8, each_unknown_call/4,
        each_body_succ_builtin/12, body_succ_meta/7, reduce_equivalent/3,
        each_apply_trusted/7,   widen_succ/4,   decide_memo/6,clause_applies/2,
        abs_subset_/3]).

:- use_module(ciaopp(plai/domains)).
:- use_module(ciaopp(plai/trace_fixp), [fixpoint_trace/7, cleanup/0]).
:- use_module(ciaopp(plai/plai_db),
        [ complete/7, memo_call/5, memo_table/6, cleanup_plai_db/1, patch_parents/6 ]).
:- use_module(ciaopp(plai/psets), [update_if_member_idlist/3]).
:- use_module(ciaopp(plai/re_analysis), [erase_previous_memo_tables_and_parents/4]).
:- use_module(ciaopp(plai/transform), [body_info0/4, trans_clause/3]).
:- use_module(ciaopp(plai/apply_assertions_old),
        [ apply_trusted0/7,
          cleanup_trusts/1 ]).

:- doc(author,"Kalyan Muthukumar").
:- doc(author,"Maria Garcia de la Banda").
:- doc(author,"Francisco Bueno").

:- doc(module,"This module implements the top-down fixpoint
        algorithm of PLAI, both in its mono-variant and multi-variant
        on successes versions. It is always multi-variant on calls.
        The algorithm is parametric on the particular analysis domain.").


:- data '$depend_list'/3.
:- data ch_id/2.

:- data approx/6.
:- data fixpoint/6.
:- data fixpoint_variant/6.
:- data approx_variant/7.

init_fixpoint:-
        retractall_fact(approx(_,_,_,_,_,_)),
        retractall_fact(fixpoint(_,_,_,_,_,_)),
        retractall_fact('$depend_list'(_,_,_)),
        retractall_fact(ch_id(_,_)),
        retractall_fact(fixpoint_variant(_,_,_,_,_,_)),
        retractall_fact(approx_variant(_,_,_,_,_,_,_)),
        trace_fixp:cleanup.

cleanup_fixpoint(AbsInt):-
        cleanup_plai_db(AbsInt),
        cleanup_trusts(AbsInt),
        retractall_fact(fixp_id(_)),
        asserta_fact(fixp_id(0)), % there is no way to recover this
        init_fixpoint.            % if several analyses coexist!

approx_to_completes(AbsInt):-
        current_fact(approx(SgKey,Sg,Proj,Prime,Pid,Fs),Ref),
        asserta_fact(complete(SgKey,AbsInt,Sg,Proj,Prime,Pid,Fs)),
        erase(Ref),
        fail.
approx_to_completes(AbsInt):-
        current_fact(approx_variant(_Id,Pid,SgKey,Sg,Proj,Prime,Fs),Ref),
        asserta_fact(complete(SgKey,AbsInt,Sg,Proj,Prime,Pid,Fs)),
        erase(Ref),
        fail.
approx_to_completes(_AbsInt).


%-------------------------------------------------------------------------%
% call_to_success(+,+,+,+,+,+,+,-,-,+,+,+)                                %
```

```
%-----------------------------------------------------------------------%

call_to_success(_RFlag,SgKey,Call,Proj,Sg,Sv,AbsInt,_ClId,Succ,List,F,N,Id):-
        % ClId = number identifying the clause?... for an entry point is 0...
        % F = program point of the call. clauseId+/0 for an entry call
        current_fact(complete(SgKey,AbsInt,Subg,Proj1,Prime1,Id,Fs),R),
        identical_proj(AbsInt,Sg,Proj,Subg,Proj1), !,
        patch_parents(R,complete(SgKey,AbsInt,Subg,Proj1,Prime1,Id,Ps),F,N,Ps,Fs),
        List = [],
        each_abs_sort(Prime1,AbsInt,Prime),
        each_extend(Sg,Prime,AbsInt,Sv,Call,Succ).
call_to_success(r,SgKey,Call,Proj,Sg,Sv,AbsInt,_ClId,Succ,List,F,N,Id) :-
        current_fact(approx(SgKey,Subg,Proj1,Prime1,Id,Fs),Ref),
        identical_proj(AbsInt,Sg,Proj,Subg,Proj1), !,
        each_abs_sort(Prime1,AbsInt,TempPrime),
        current_fact('$depend_list'(Id,SgKey,IdList)),
        call_to_success_approx(SgKey,Subg,Call,Proj,Proj1,Sg,Sv,AbsInt,F,N,Fs,
                               Id,Ref,IdList,Prime1,TempPrime,List,Prime),
        each_extend(Sg,Prime,AbsInt,Sv,Call,Succ).
call_to_success(r,SgKey,Call,Proj,Sg,Sv,AbsInt,_ClId,Succ,List,F,N,Id):-
        current_fact(fixpoint(SgKey,Subg,Proj1,Prime1,Id,Fs),Ref),
        identical_proj(AbsInt,Sg,Proj,Subg,Proj1), !,
        patch_parents(Ref,fixpoint(SgKey,Subg,Proj1,Prime1,Id,Ps),F,N,Ps,Fs),
        current_fact(ch_id(Id,Num)),
        List = [Id/Num],
        each_abs_sort(Prime1,AbsInt,Prime),
        each_extend(Sg,Prime,AbsInt,Sv,Call,Succ).
call_to_success(_RFlag,SgKey,Call,Proj,Sg,Sv,AbsInt,_ClId,Succ,List,F,N,Id):-
        current_pp_flag(variants,on),
        current_fact(complete(SgKey,AbsInt,Subg,Proj1,Prime1,_Id1,_Fs),_R),
        identical_proj_1(AbsInt,Sg,Proj,Subg,Proj1,Prime1,Prime2), !,
        format("call to success tipe _RFlag SgKey",[]),
        ( current_pp_flag(reuse_fixp_id,on) ->
            fixpoint_id_reuse_prev(SgKey,AbsInt,Sg,Proj,Id)
        ;
            fixpoint_id(Id)
        ),
        each_abs_sort(Prime2,AbsInt,Prime),
        List = [],
        asserta_fact(complete(SgKey,AbsInt,Sg,Proj,Prime,Id,[(F,N)])),
        each_extend(Sg,Prime,AbsInt,Sv,Call,Succ).
call_to_success(r,SgKey,Call,Proj,Sg,Sv,AbsInt,_ClId,Succ,List,F,N,Id) :-
        current_pp_flag(variants,on),
        current_fact(approx(SgKey,Subg,Proj1,Prime1,Id1,Fs),Ref),
        identical_proj_1(AbsInt,Sg,Proj,Subg,Proj1,Prime1,Prime2), !,
        each_abs_sort(Prime2,AbsInt,TempPrime),
        current_fact('$depend_list'(Id1,SgKey,IdList)),
        call_to_success_approx_variant(SgKey,Subg,Call,Proj,Proj1,Sg,Sv,AbsInt,F,N,Fs,
                               Id,Id1,Ref,IdList,Prime1,TempPrime,List,Prime),
        each_extend(Sg,Prime,AbsInt,Sv,Call,Succ).
call_to_success(r,SgKey,Call,Proj,Sg,Sv,AbsInt,_ClId,Succ,List,F,N,Id):-
        current_pp_flag(variants,on),
        current_fact(fixpoint(SgKey,Subg,Proj1,Prime1,Id1,_Fs),_Ref),
        identical_proj_1(AbsInt,Sg,Proj,Subg,Proj1,Prime1,Prime2), !,
        (
            current_fact(fixpoint_variant(Id1,Id,SgKey,Sgv,Projv,Fsv),Refv),
            identical_proj(AbsInt,Sg,Proj,Sgv,Projv) ->
            patch_parents(Refv,fixpoint_variant(Id1,Id,SgKey,Sgv,Projv,Ps),F,N,Ps,Fsv)
        ;
            (
                current_pp_flag(reuse_fixp_id,on) ->
                fixpoint_id_reuse_prev(SgKey,AbsInt,Sg,Proj,Id)
            ;
                fixpoint_id(Id)
            ),
            asserta_fact(fixpoint_variant(Id1,Id,SgKey,Sg,Proj,[(F,N)]))
        ),
        each_abs_sort(Prime2,AbsInt,Prime),
        current_fact(ch_id(Id1,Num)),
        List = [Id1/Num],
        each_extend(Sg,Prime,AbsInt,Sv,Call,Succ).
call_to_success(r,SgKey,Call,Proj,Sg,Sv,AbsInt,_ClId,Succ,List,F,N,Id) :-
        init_fixpoint0(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,[(F,N)],Id,List,Prime),
        each_extend(Sg,Prime,AbsInt,Sv,Call,Succ).
call_to_success(nr,SgKey,Call,Proj,Sg,Sv,AbsInt,ClId,Succ,[],F,N,Id):-
        ( current_pp_flag(reuse_fixp_id,on) ->
            fixpoint_id_reuse_prev(SgKey,AbsInt,Sg,Proj,Id)
        ;
            fixpoint_id(Id)
        ),
        proj_to_prime_nr(SgKey,Sg,Sv,Call,Proj,AbsInt,ClId,Prime,Id),
        asserta_fact(complete(SgKey,AbsInt,Sg,Proj,Prime,Id,[(F,N)])),
        each_extend(Sg,Prime,AbsInt,Sv,Call,Succ).

call_to_success_approx(SgKey,Subg,_Call,Proj,Proj1,Sg,_Sv,_AbsInt,F,N,Fs,
                             Id,Ref,IdList,Prime1,TempPrime,List,Prime):-
        not_modified(IdList), !,
        patch_parents(Ref,approx(SgKey,Subg,Proj1,Prime1,Id,Ps),F,N,Ps,Fs),
        Prime = TempPrime,
        List = IdList.
call_to_success_approx(SgKey,_Subg,Call,Proj,_Proj1,Sg,Sv,AbsInt,F,N,Fs,
                             Id,Ref,_IdList,_Prime1,TempPrime,List,Prime):-
        erase(Ref),
        init_fixpoint_(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,Fs,Id,
                             TempPrime,List,Prime).

aproxs_to_fixpoint_variant(Id):-
        current_fact(approx_variant(Id,Idv,SgKey,Sgv,Projv,_Primev,Fs),Ref),!,
        erase(Ref),
        asserta_fact(fixpoint_variant(Id,Idv,SgKey,Sgv,Projv,Fs)),
        aproxs_to_fixpoint_variant(Id).
aproxs_to_fixpoint_variant(_).


call_to_success_approx_variant(SgKey,_Subg,_Call,Proj,_Proj1,Sg,_Sv,AbsInt,F,N,_Fs,
                             Id,Id1,_Ref,IdList,_Prime1,TempPrime,List,Prime):-
        not_modified(IdList), !,
        (
            current_fact(approx_variant(Id1,Id,SgKey,Sgv,Projv,Primev,Fsv),Refv),
            identical_proj(AbsInt,Sg,Proj,Sgv,Projv) ->
            patch_parents(Refv,approx_variant(Id1,Id,SgKey,Sgv,Projv,Primev,Ps),F,N,Ps,Fsv)
        ;
            (
                current_pp_flag(reuse_fixp_id,on) ->
                fixpoint_id_reuse_prev(SgKey,AbsInt,Sg,Proj,Id)
            ;
                fixpoint_id(Id)
            ),
            asserta_fact(approx_variant(Id1,Id,SgKey,Sg,Proj,TempPrime,[(F,N)]))
        ),
        Prime = TempPrime,
        List = IdList.
call_to_success_approx_variant(SgKey,Subg,Call,Proj,Proj1,Sg,Sv,AbsInt,F,N,Fs,
                             Id,Id1,Ref,_IdList,Prime1,_TempPrime,List,Prime):-
        (
            current_fact(approx_variant(Id1,Id,SgKey,Sgv,Projv,_Primev,Fsv),Refv),
            identical_proj(AbsInt,Sg,Proj,Sgv,Projv) ->
            erase(Refv),
            ( member((F,N),Fsv) -> NewFs = Fsv ; NewFs = [(F,N)|Fsv] %)
        ;
            (
                current_pp_flag(reuse_fixp_id,on) ->
                fixpoint_id_reuse_prev(SgKey,AbsInt,Sg,Proj,Id)
            ;
                fixpoint_id(Id)
            ),
            NewFs = [(F,N)]
        ),
        aproxs_to_fixpoint_variant(Id1),
        erase(Ref),
        asserta_fact(fixpoint_variant(Id1,Id,SgKey,Sg,Proj,NewFs)),
        varset(Subg,Subv),
        init_fixpoint_(SgKey,Call,Proj1,Subg,Subv,AbsInt,F,N,Fs,Id1,
                       Prime1,List,Prime0),
        each_exit_to_prime(Prime0,AbsInt,Sg,Subv,Subg,Sv,(no,Proj),Prime).

init_fixpoint0(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,Fs,Id,List,Prime):-
        init_fixpoint2(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,Fs,Id,List,Prime).

init_fixpoint1(SgKey,_Call,Proj,Sg,_Sv,AbsInt,F,N,_Fs0,Id,List,Prime):-
        current_fact(complete(SgKey,AbsInt,Subg,Proj1,Prime1,Id,Fs),R),
        identical_proj(AbsInt,Sg,Proj,Subg,Proj1), !,
        patch_parents(R,complete(SgKey,AbsInt,Subg,Proj1,Prime1,Id,Ps),F,N,Ps,Fs),
        List = [],
        each_abs_sort(Prime1,AbsInt,Prime).
init_fixpoint1(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,_Fs0,Id,List,Prime):-
        current_fact(approx(SgKey,Subg,Proj1,Prime1,Id,Fs),Ref),
        identical_proj(AbsInt,Sg,Proj,Subg,Proj1), !,
        each_abs_sort(Prime1,AbsInt,TempPrime),
        current_fact('$depend_list'(Id,SgKey,IdList)),
        call_to_success_approx(SgKey,Subg,Call,Proj,Proj1,Sg,Sv,AbsInt,F,N,Fs,
                               Id,Ref,IdList,Prime1,TempPrime,List,Prime).
init_fixpoint1(SgKey,_,Proj,Sg,_Sv,AbsInt,F,N,_Fs0,Id,List,Prime):-
        current_fact(fixpoint(SgKey,Subg,Proj1,Prime1,Id,Fs),Ref),
        identical_proj(AbsInt,Sg,Proj,Subg,Proj1), !,
        patch_parents(Ref,fixpoint(SgKey,Subg,Proj1,Prime1,Id,Ps),F,N,Ps,Fs),
        current_fact(ch_id(Id,Num)),
        List = [Id/Num],
        each_abs_sort(Prime1,AbsInt,Prime).
init_fixpoint1(SgKey,_Call,Proj,Sg,_Sv,AbsInt,F,N,_Fs0,Id,List,Prime):-
        current_pp_flag(variants,on),
        current_fact(complete(SgKey,AbsInt,Subg,Proj1,Prime1,_Id1,_Fs),_R),
        identical_proj_1(AbsInt,Sg,Proj,Subg,Proj1,Prime1,Prime2), !,
        ( current_pp_flag(reuse_fixp_id,on) ->
            fixpoint_id_reuse_prev(SgKey,AbsInt,Sg,Proj,Id)
        ;
            fixpoint_id(Id)
        ),
        each_abs_sort(Prime2,AbsInt,Prime),
        List = [],
        asserta_fact(complete(SgKey,AbsInt,Sg,Proj,Prime,Id,[(F,N)])).
init_fixpoint1(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,_Fs0,Id,List,Prime):-
        current_pp_flag(variants,on),
        current_fact(approx(SgKey,Subg,Proj1,Prime1,Id1,Fs),Ref),
        identical_proj_1(AbsInt,Sg,Proj,Subg,Proj1,Prime1,Prime2), !,
        each_abs_sort(Prime2,AbsInt,TempPrime),
        current_fact('$depend_list'(Id1,SgKey,IdList)),
        call_to_success_variant(SgKey,Subg,Call,Proj,Proj1,Sg,Sv,AbsInt,F,N,Fs,
                               Id,Id1,Ref,IdList,Prime1,TempPrime,List,Prime).
init_fixpoint1(SgKey,_,Proj,Sg,_Sv,AbsInt,F,N,_Fs0,Id,List,Prime):-
        current_pp_flag(variants,on),
        current_fact(fixpoint(SgKey,Subg,Proj1,Prime1,Id1,_Fs),_Ref),
        identical_proj_1(AbsInt,Sg,Proj,Subg,Proj1,Prime1,Prime2), !,
        (
            current_fact(fixpoint_variant(Id1,Id,SgKey,Sgv,Projv,Fsv),Refv),
            identical_proj(AbsInt,Sg,Proj,Sgv,Projv) ->
            patch_parents(Refv,fixpoint_variant(Id1,Id,SgKey,Sgv,Projv,Ps),F,N,Ps,Fsv)
        ;
```

```
                (
                    current_pp_flag(reuse_fixp_id,on) ->
                    fixpoint_id_reuse_prev(SgKey,AbsInt,Sg,Proj,Id)
                ;
                    fixpoint_id(Id)
                ),
                asserta_fact(fixpoint_variant(Id1,Id,SgKey,Sg,Proj,[(F,N)]))
            ),
            each_abs_sort(Prime2,AbsInt,Prime),
            current_fact(ch_id(Id1,Num)),
            List = [Id1/Num].
init_fixpoint1(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,Fs,Id,List,Prime):-
        init_fixpoint2(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,Fs,Id,List,Prime).

init_fixpoint2(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,Fs,Id,List,Prime):-
        ( current_pp_flag(reuse_fixp_id,on) ->
            fixpoint_id_reuse_prev(SgKey,AbsInt,Sg,Proj,Id)
        ;
            fixpoint_id(Id)
        ),
        asserta_fact(ch_id(Id,1)),
        proj_to_prime_r(SgKey,Sg,Sv,Call,Proj,AbsInt,TempPrime,Id),
        init_fixpoint_(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,Fs,Id,
                       TempPrime,List,Prime).

init_fixpoint_(SgKey,Call,Proj,Sg,Sv,AbsInt,F,N,Fs,Id,Prime0,List,Prime):-
        normalize_asub0(AbsInt,Prime0,TempPrime),
        asserta_fact(fixpoint(SgKey,Sg,Proj,TempPrime,Id,Fs)),
        bagof(X, X^(trans_clause(SgKey,r,X)),Clauses),!,
        fixpoint_compute(Clauses,SgKey,Sg,Sv,Call,Proj,
                         AbsInt,_LEntry,TempPrime,Prime1,Id,TempList),
        each_apply_trusted(Proj,SgKey,Sg,Sv,AbsInt,Prime1,Prime),
        current_fact(fixpoint(SgKey,Sg,_,_,Id,Fs2),Ref),
        erase(Ref),
        ( current_fact('$depend_list'(Id,SgKey,_),RefDep) ->
          erase(RefDep)
        ; true
        ),
        update_if_member_idlist(TempList,Id,AddList),
        ( member((F,N),Fs2) -> NewFs = Fs2 ; NewFs = [(F,N)|Fs2] ),
        decide_approx(AddList,Id,NewFs,AbsInt,SgKey,Sg,Proj,Prime),
        List = AddList.

widen_call(AbsInt,SgKey,Sg,F1,Id0,Proj1,Proj):-
        ( current_pp_flag(widencall,off) -> fail ; true ),
        widen_call0(AbsInt,SgKey,Sg,F1,Id0,[Id0],Proj1,Proj), !.

widen_call0(AbsInt,SgKey,Sg,F1,Id0,Ids,Proj1,Proj):-
        widen_call1(AbsInt,SgKey,Sg,F1,Id0,Ids,Proj1,Proj).
widen_call0(AbsInt,SgKey,Sg,F1,Id0,Ids,Proj1,Proj):-
        current_pp_flag(widencall,com_child),
        widen_call2(AbsInt,SgKey,Sg,F1,Id0,Ids,Proj1,Proj).

widen_call1(AbsInt,SgKey,Sg,F1,Id0,Ids,Proj1,Proj):-
        current_fact(fixpoint(SgKey0,Sg0,Proj0,_Prime0,Id0,Fs0)),
        ( SgKey=SgKey0,
          % same program point:
          member((F1,_NewId0),Fs0)
        -> Sg0=Sg,
           abs_sort(AbsInt,Proj0,Proj0_s),
           abs_sort(AbsInt,Proj1,Proj1_s),
           widencall(AbsInt,Proj0_s,Proj1_s,Proj)
         ; % continue with the parents:
           member((_F1,NewId0),Fs0),
           \+ member(NewId0,Ids),
           widen_call1(AbsInt,SgKey,Sg,F1,NewId0,[NewId0|Ids],Proj1,Proj)
        ).

widen_call2(AbsInt,SgKey,Sg,F1,_Id,_Ids,Proj1,Proj):-
        current_fact(complete(SgKey,AbsInt,Sg0,Proj0,_Prime0,_Id0,Fs0)),
        member((F1,Id0),Fs0),
        Sg0=Sg,
        same_fixpoint_ancestor(Id0,[Id0],AbsInt),
        abs_sort(AbsInt,Proj0,Proj0_s),
        abs_sort(AbsInt,Proj1,Proj1_s),
        widencall(AbsInt,Proj0_s,Proj1_s,Proj).

same_fixpoint_ancestor(Id0,_Ids,_AbsInt):-
        current_fact(fixpoint(_SgKey0,_Sg0,_Proj0,_Prime0,Id0,_Fs0)), !.
same_fixpoint_ancestor(Id0,_Ids,_AbsInt):-
        current_fact(approx(_SgKey0,_Sg0,_Proj0,_Prime0,Id0,_Fs0)), !.
same_fixpoint_ancestor(Id0,Ids,AbsInt):-
        current_fact(complete(_SgKey0,AbsInt,_Sg0,_Proj0,_Prime0,Id0,Fs0)),
        member((_F1,Id),Fs0),
        \+ member(Id,Ids),
        same_fixpoint_ancestor(Id,[Id|Ids],AbsInt).

fixpoint_variants_update(Id,AbsInt,Sg,Prime):-
        current_fact(fixpoint_variant(Id,Idv,SgKey,Sgv,Projv,Fs),Ref),!,
        erase(Ref),
        varset(Sg,Hv),
        varset(Sgv,Hvv),
        each_exit_to_prime(Prime,AbsInt,Sgv,Hv,Sg,Hvv,(no,Projv),Prime2),
        asserta_fact(complete(SgKey,AbsInt,Sgv,Projv,Prime2,Idv,Fs)),
        fixpoint_variants_update(Id,AbsInt,Sg,Prime).
fixpoint_variants_update(_,_,_,_).

approx_variants_update(Id,AbsInt,Sg,Prime):-
        current_fact(fixpoint_variant(Id,Idv,SgKey,Sgv,Projv,Fs),Ref),!,
        erase(Ref),
        varset(Sg,Hv),
        varset(Sgv,Hvv),
        each_exit_to_prime(Prime,AbsInt,Sgv,Hv,Sg,Hvv,(no,Projv),Prime2),
        asserta_fact(approx_variant(Id,Idv,SgKey,Sgv,Projv,Prime2,Fs)),
        approx_variants_update(Id,AbsInt,Sg,Prime).
approx_variants_update(_,_,_,_).

decide_approx([],Id,Fs,AbsInt,SgKey,Sg,Proj,Prime):- !,
        current_fact(ch_id(Id,_),Ref3),
        erase(Ref3),
        % Not needed for correctness: only book-keeping
        % update_depend_list_approx(Id,AbsInt),
        asserta_fact(complete(SgKey,AbsInt,Sg,Proj,Prime,Id,Fs)),
        (
            current_pp_flag(variants,on) ->
            each_abs_sort(Prime,AbsInt,Prime_s),
            fixpoint_variants_update(Id,AbsInt,Sg,Prime_s)
        ;
            true
        ).
decide_approx(AddList,Id,Fs,_AbsInt,SgKey,Sg,Proj,Prime):-
        asserta_fact('$depend_list'(Id,SgKey,AddList)),
        asserta_fact(approx(SgKey,Sg,Proj,Prime,Id,Fs),_),
        (
            current_pp_flag(variants,on) ->
            each_abs_sort(Prime,AbsInt,Prime_s),
            approx_variants_update(Id,AbsInt,Sg,Prime_s)
        ;
            true
        ).

not_modified([]).
not_modified([Id/N|List]):-
        current_fact(ch_id(Id,N)), !,
        not_modified(List).

proj_to_prime_nr(SgKey,Sg,Sv,Call,Proj,AbsInt,_ClId,LPrime,Id) :-
        bagof(X, X^(trans_clause(SgKey,nr,X)),Clauses), !,
        proj_to_prime(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,LPrime1,Id),
        compute_clauses_lub(AbsInt,Proj,LPrime1,LPrime).
proj_to_prime_nr(SgKey,Sg,Sv,_Call,Proj,AbsInt,ClId,LPrime,_Id) :-
        apply_trusted0(Proj,SgKey,Sg,Sv,AbsInt,ClId,Prime), !,
        singleton(Prime,LPrime).
proj_to_prime_nr(_SgKey,Sg,Sv,Call,_Proj,AbsInt,_ClId,LSucc,_Id) :-
        % In Java programs, mode and type information is known for any method.
        % Therefore, in case of a method with unavailable code we can still
        % infer useful information.
        ( current_pp_flag(prog_lang,java) ->
          unknown_call(AbsInt,Sg,Sv,Call,Succ),
          singleton(Succ,LSucc)
        ;
          fail
        ).
proj_to_prime_nr(SgKey,_Sg,_Sv,_Call,_Proj,_AbsInt,ClId,Bot,_Id) :-
        bottom(Bot),
        inexistent(SgKey,ClId).

proj_to_prime_r(SgKey,Sg,Sv,Call,Proj,AbsInt,Prime,Id) :-
        bagof(X, X^(trans_clause(SgKey,nr,X)),Clauses), !,
        proj_to_prime(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,Prime,Id).
proj_to_prime_r(_SgKey,_Sg,_Sv,_Call,_Proj,_AbsInt,Bot,_Id):-
        bottom(Bot).

proj_to_prime(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,Prime,Id) :-
        proj_to_prime_loop(Clauses,Sg,Sv,Call,Proj,AbsInt,ListPrime0,Id),
        reduce_equivalent(ListPrime0,AbsInt,ListPrime1),
        each_apply_trusted(Proj,SgKey,Sg,Sv,AbsInt,ListPrime1,Prime).

proj_to_prime_loop([],_,_,_,_,_,[],_).
proj_to_prime_loop([Clause|Rest],Sg,Sv,Call,Proj,AbsInt,Primes,Id):-
        do_nr_cl(Clause,Sg,Sv,Call,Proj,AbsInt,Primes,TailPrimes,Id),!,
        proj_to_prime_loop(Rest,Sg,Sv,Call,Proj,AbsInt,TailPrimes,Id).

do_nr_cl(Clause,Sg,Sv,Call,Proj,AbsInt,Primes,TailPrimes,Id):-
        Clause = clause(Head,Vars_u,K,Body),
        clause_applies(Head,Sg), !,
        varset(Head,Hv),
        sort(Vars_u,Vars),
        ord_subtract(Vars,Hv,Fv),
        process_body(Body,K,AbsInt,Sg,Hv,Fv,Vars_u,Head,Sv,Call,
                                                        Proj,LPrime,Id),
        append_(LPrime,TailPrimes,Primes).
do_nr_cl(_Clause,_Sg,_Sv,_Call,_Proj,_AbsInt,Primes,Primes,_Id).

append_([Prime],TailPrimes,Primes):- !, Primes=[Prime|TailPrimes].
append_(LPrime,TailPrimes,Primes):- append(LPrime,TailPrimes,Primes).

process_body(Body,K,AbsInt,Sg,Hv,Fv,_,Head,Sv,Call,Proj,LPrime,Id):-
        Body = g(_,[],'$built'(_,true,_),'true/0',true), !,
        Help=(Sv,Sg,Hv,Fv,AbsInt),
        singleton(Prime,LPrime),
        call_to_success_fact(AbsInt,Sg,Hv,Head,K,Sv,Call,Proj,Prime,_Succ),
        ( current_pp_flag(fact_info,on) ->
          call_to_entry(AbsInt,Sv,Sg,Hv,Head,K,[],Prime,Exit,_),
          decide_memo(AbsInt,K,Id,no,Hv,[Exit])
        ;
          true
        ).
```

```
process_body(Body,K,AbsInt,Sg,Hv,Fv,Vars_u,Head,Sv,_,Proj,Prime,Id):-
        call_to_entry(AbsInt,Sv,Sg,Hv,Head,K,Fv,Proj,Entry,ExtraInfo),
        singleton(Entry,LEntry),
        entry_to_exit(Body,K,LEntry,Exit,[],_,Vars_u,AbsInt,Id),
        each_exit_to_prime(Exit,AbsInt,Sg,Hv,Head,Sv,ExtraInfo,Prime).

fixpoint_compute(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,LEntryInf,
                 Prime0,Prime,Id,List) :-
        fixpoint_compute_(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,LEntryInf,
                          Prime0,Prime1,Id,List),
        compute_clauses_lub(AbsInt,Proj,Prime1,Prime).

fixpoint_compute_(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,LEntryInf,
                 TempPrime,Prime,Id,List) :-
        compute(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,LEntryInf,
                 TempPrime,Prime1,Id,[],NewList,Flag),
        fixpoint(NewList,Flag,Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,LEntryInf,
                 Prime1,Prime,Id,List), !.

fixpoint([],_,_,_,_,_,_,_,_,_,Prime1,Prime,_,List):- !,
        Prime = Prime1,
        List = [].
fixpoint(NewList,Flag,_,_,_,_,_,_,_,_,Prime1,Prime,_,List):-
        var(Flag),!,
        Prime = Prime1,
        List = NewList.
fixpoint(_,_,Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,LEntryInf,Prime1,Prime,Id,List):-
        fixpoint_compute_(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,LEntryInf,
                 Prime1,Prime,Id,List).

% some domains need normalization to perform the widening:
normalize_asub0(AbsInt,Prime0,Prime):-
        current_pp_flag(widen,on), !,
        normalize_asub(AbsInt,Prime0,Prime).
normalize_asub0(_AbsInt,Prime,Prime).

compute([],_,_,_,_,_,_,[],Prime,Prime,_,List,List,_).
compute([Clause|Rest],SgKey,Sg,Sv,Call,Proj,AbsInt,[EntryInf|LEntryInf],
                          TempPrime,Prime,Id,List,NewList,Flag) :-
        do_r_cl(Clause,SgKey,Sg,Sv,Proj,AbsInt,EntryInf,Id,List,IntList,
                                 TempPrime,NewPrime,Flag),
        compute(Rest,SgKey,Sg,Sv,Call,Proj,AbsInt,LEntryInf,NewPrime,Prime,
                                 Id,IntList,NewList,Flag).

do_r_cl(Clause,SgKey,Sg,Sv,Proj,AbsInt,EntryInf,Id,OldL,List,TempPrime,
                                             NewPrime,Flag):-
        Clause=clause(Head,Vars_u,K,Body),
        clause_applies(Head,Sg), !,
        erase_previous_memo_tables_and_parents(Body,AbsInt,K,Id),
        varset(Head,Hv),
        reuse_entry(EntryInf,Vars_u,AbsInt,Sv,Sg,Hv,Head,K,Proj,Entry,ExtraInfo),
        singleton(Entry,LEntry),
        entry_to_exit(Body,K,LEntry,Exit,OldL,List,Vars_u,AbsInt,Id),
        each_exit_to_prime(Exit,AbsInt,Sg,Hv,Head,Sv,ExtraInfo,Prime1),
        widen_succ(AbsInt,TempPrime,Prime1,NewPrime),
        decide_flag(AbsInt,TempPrime,NewPrime,SgKey,Sg,Id,Proj,Flag).

do_r_cl(_,_,_,_,_,_,_,_,List,List,Prime,Prime,_).

widen_succ_off(AbsInt,Prime0,Prime1,LPrime):-
        current_pp_flag(multi_success,on), !,
        reduce_equivalent([Prime0,Prime1],AbsInt,LPrime).
widen_succ_off(AbsInt,Prime0,Prime1,Prime):-
        singleton(P0,Prime0),
        singleton(P1,Prime1),
        singleton(P,Prime),
        compute_lub(AbsInt,[P0,P1],P).

reuse_entry(EntryInf,Vars_u,AbsInt,Sv,Sg,Hv,Head,K,Proj,Entry,ExtraInfo):-
        var(EntryInf), !,
        sort(Vars_u,Vars),
        ord_subtract(Vars,Hv,Fv),
        call_to_entry(AbsInt,Sv,Sg,Hv,Head,K,Fv,Proj,Entry,ExtraInfo),
        EntryInf = (Entry,ExtraInfo).
reuse_entry(EntryInf,_Vars_u,_AbsInt,_Sv,_Sg,_Hv,_Head,_K,_Proj,Entry,ExtraInfo):-
        EntryInf = (Entry,ExtraInfo).

decide_flag(AbsInt,TempPrime,NewPrime,_SgKey,_Sg,_Id,_Proj,_Flag):-
        abs_subset_(NewPrime,AbsInt,TempPrime), !.
decide_flag(_AbsInt,TempPrime,NewPrime,SgKey,Sg,Id,Proj,Flag):-
        Flag = notend,
        merge_(NewPrime,TempPrime,LPrime),
        current_fact(fixpoint(SgKey,Sg,_,_,Id,Fs),Ref),
        erase(Ref),
        asserta_fact(fixpoint(SgKey,Sg,Proj,LPrime,Id,Fs)),
        current_fact(ch_id(Id,Num),Ref3),
        erase(Ref3),
        Num1 is Num+1,
        asserta_fact(ch_id(Id,Num1)).

merge_([NewPrime],_TempPrime,LPrime):- !, LPrime=[NewPrime].
merge_(NewPrime,TempPrime,LPrime):-
        merge(NewPrime,TempPrime,LPrime).

%-------------------------------------------------------------------------%
% entry_to_exit(+,+,+,-,+,-,+,+,+)                                        %
%-------------------------------------------------------------------------%

entry_to_exit((Sg,Rest),K,Call,Exit,OldList,NewList,Vars_u,AbsInt,NewN):- !,
        body_succ(Call,Sg,Succ,OldList,IntList,Vars_u,AbsInt,K,NewN,_),
        entry_to_exit(Rest,K,Succ,Exit,IntList,NewList,Vars_u,AbsInt,NewN).
entry_to_exit(true,_,Call,Call,List,List,_,_,_):- !.
entry_to_exit(Sg,Key,Call,Exit,OldList,NewList,Vars_u,AbsInt,NewN):-
        body_succ(Call,Sg,Exit,OldList,NewList,Vars_u,AbsInt,Key,NewN,_),
        decide_memo(AbsInt,Key,NewN,no,Vars_u,Exit),!.

body_succ(Call,Atom,Succ,List,List,HvFv_u,AbsInt,_ClId,ParentId,no):-
        bottom(Call), !,
        Succ = Call,
        Atom=g(Key,_Av,_I,_SgKey,_Sg),
        asserta_fact(memo_table(Key,AbsInt,ParentId,no,HvFv_u,Succ)).
body_succ(Call,Atom,Succ,List,NewList,HvFv_u,AbsInt,ClId,ParentId,Id):-
        Atom=g(Key,Sv,Info,SgKey,Sg),
        body_succ_(Info,SgKey,Sg,Sv,HvFv_u,Call,Succ,List,NewList,AbsInt,
                   ClId,Key,ParentId,Id),
        decide_memo(AbsInt,Key,ParentId,Id,HvFv_u,Call).

body_succ_(Info,SgKey,Sg,Sv,HFv,Call,Succ,L,NewL,AbsInt,ClId,Key,PId,Id):-
        Info = [_|_], !,
        split_combined_domain(AbsInt,Call,Calls,Domains),
        map_body_succ(Info,SgKey,Sg,Sv,HFv,Calls,Succs,L,NewL,Domains,
                      ClId,Key,PId,Id),
        split_combined_domain(AbsInt,Succ,Succs,Domains).
body_succ_(Info,SgKey,Sg,Sv,HFv,Call,Succ,L,NewL,AbsInt,ClId,Key,PId,Id):-
        body_succ0(Info,SgKey,Sg,Sv,HFv,Call,Succ,L,NewL,AbsInt,
                   ClId,Key,PId,Id).

map_body_succ([],_SgKey,_Sg,_Sv,_HFv,[],[],L,L,[],_ClId,_Key,_PId,no).
map_body_succ([I|Info],SgKey,Sg,Sv,HFv,[Call|Calls],[Succ|Succs],L,NewL,
              [AbsInt|Domains],ClId,Key,PId,Id):-
        body_succ0(I,SgKey,Sg,Sv,HFv,Call,Succ,L,_NewL,AbsInt,
                   ClId,Key,PId,_Id), !,
        map_body_succ(Info,SgKey,Sg,Sv,HFv,Calls,Succs,L,NewL,Domains,
                   ClId,Key,PId,Id).

body_succ0('$var',SgKey,Sg,_Sv_u,HvFv_u,Calls,Succs,List0,List,AbsInt,
            ClId,F,_N,Id):-
        !,
        ( Calls=[Call],
          concrete(AbsInt,Sg,Call,Concretes),
          concretes_to_body(Concretes,SgKey,AbsInt,B)
        -> fixpoint_id(Id),
           meta_call(B,HvFv_u,Calls,[],Succs,List0,List,AbsInt,ClId,Id,Ids),
           assertz_fact(memo_call(F,Id,AbsInt,Concretes,Ids))
         ; Id=no,
           List=List0,
           variable(F,ClId),
           each_unknown_call(Calls,AbsInt,[Sg],Succs) % Sg is a variable
        ).
body_succ0('$meta'(T,B,_),SgKey,Sg,Sv_u,HvFv_u,Call,Succ,List0,List,AbsInt,
            ClId,F,N,Id):-
        !,
        ( current_pp_flag(reuse_fixp_id,on) ->
           ( Call=[C]
             -> sort(Sv_u,Sv),
                project(AbsInt,Sg,Sv,HvFv_u,C,Proj),
                fixpoint_id_reuse_prev(SgKey,AbsInt,Sg,Proj,Id)
              ; true
           )
        ;
            fixpoint_id(Id)
        ),
        meta_call(B,HvFv_u,Call,[],Exits,List0,List,AbsInt,ClId,Id,_Ids),
        ( body_succ_meta(T,AbsInt,Sv_u,HvFv_u,Call,Exits,Succ) ->
           ( Call=[C] ->
               sort(Sv_u,Sv),
               project(AbsInt,Sg,Sv,HvFv_u,C,Proj),
               each_project(Exits,AbsInt,Sg,Sv,HvFv_u,Prime),
               asserta_fact(complete(SgKey,AbsInt,Sg,Proj,Prime,Id,[(F,N)]))
             ; true
           )
        ; % for the trusts, if any:
          varset(Sg,Sv_r), % Sv_u contains extra vars (from meta-term)
                           % which will confuse apply_trusted
          body_succ0(nr,SgKey,Sg,Sv_r,HvFv_u,Call,Succ,[],_List,AbsInt,
                     ClId,F,N,Id0),
          retract_fact(complete(SgKey,AbsInt,Sg,Proj,Prime,Id0,Ps)),
          asserta_fact(complete(SgKey,AbsInt,Sg,Proj,Prime,Id,Ps))
        ).
body_succ0('$built'(T,Tg,Vs),SgKey,Sg,Sv_u,HvFv_u,Call,Succ,List0,List,AbsInt,
            _ClId,F,N,Id):-
        !,
        Id=no,
        List=List0,
        sort(Sv_u,Sv),
        each_body_succ_builtin(Call,AbsInt,T,Tg,Vs,SgKey,Sg,Sv,HvFv_u,F,N,Succ).
body_succ0(RFlag,SgKey,Sg,Sv_u,HvFv_u,Call,Succ,List0,List,AbsInt,
           ClId,F,N,Id):-
        sort(Sv_u,Sv),
        each_call_to_success(Call,RFlag,SgKey,Sg,Sv,HvFv_u,AbsInt,ClId,
                             Succ,List0,List,F,N,Id).

each_call_to_success([Call],RFlag,SgKey,Sg,Sv,HvFv_u,AbsInt,ClId,Succ,L0,L,
                     F,N,Id):-
        !,
        project(AbsInt,Sg,Sv,HvFv_u,Call,Proj),
        call_to_success(RFlag,SgKey,Call,Proj,Sg,Sv,AbsInt,ClId,Succ,L1,F,N,Id),
        merge(L1,L0,L).
each_call_to_success(LCall,RFlag,SgKey,Sg,Sv,HvFv_u,AbsInt,ClId,LSucc,L0,L,
                     F,N,Id):-
        each_call_to_success0(LCall,RFlag,SgKey,Sg,Sv,HvFv_u,AbsInt,ClId,
                              LSucc,L0,L,F,N,Id).

each_call_to_success0([],_Flag,_SgK,_Sg,_Sv,_HvFv,_AbsInt,_,[],L,L,_F,_N,_NN).
each_call_to_success0([Call|LCall],RFlag,SgKey,Sg,Sv,HvFv_u,AbsInt,ClId,
                      LSucc,L0,L,F,N,NewN):-
        project(AbsInt,Sg,Sv,HvFv_u,Call,Proj),
        call_to_success(RFlag,SgKey,Call,Proj,Sg,Sv,AbsInt,ClId,LSucc0,L1,F,N,_),
        merge(L0,L1,L2),
        append(LSucc0,LSucc1,LSucc),
        each_call_to_success0(LCall,RFlag,SgKey,Sg,Sv,HvFv_u,AbsInt,ClId,
                              LSucc1,L2,L,F,N,NewN).

meta_call([],_HvFv_u,Call,[],Call,List,List,_AbsInt,_ClId,_Id,[]).
meta_call([Body|Bodies],HvFv_u,Call,Succ0,Succ,L0,List,AbsInt,ClId,Id,Ids):-
        meta_call_([Body|Bodies],HvFv_u,Call,Succ0,Succ,L0,List,AbsInt,ClId,Id,Ids).

meta_call_([Body|Bodies],HvFv_u,Call,Succ0,Succ,L0,List,AbsInt,ClId,Id,Ids):-
        meta_call_body(Body,ClId,Call,Succ1,L0,L1,HvFv_u,AbsInt,Id,Ids0),
        widen_succ(AbsInt,Succ0,Succ1,Succ2),
        append(Succ0,Succ1,Succ2),
        append(Ids0,Ids1,Ids),
        meta_call_(Bodies,HvFv_u,Call,Succ2,Succ,L1,List,AbsInt,ClId,Id,Ids1).
meta_call_([],_HvFv_u,_Call,Succ,Succ,List,List,_AbsInt,_ClId,_Id,[]).

meta_call_body((Sg,Rest),K,Call,Exit,OldList,NewList,Vars_u,AbsInt,PId,CIds):-
        !,
        CIds=[Id|Ids],
        body_succ(Call,Sg,Succ,OldList,IntList,Vars_u,AbsInt,K,PId,Id),
        meta_call_body(Rest,K,Succ,Exit,IntList,NewList,Vars_u,AbsInt,PId,Ids).
meta_call_body(true,_,Call,Call,List,List,_,_,_,[no]):- !.
meta_call_body(Sg,Key,Call,Exit,OldList,NewList,Vars_u,AbsInt,PId,[Id]):-
        body_succ(Call,Sg,Exit,OldList,NewList,Vars_u,AbsInt,Key,PId,Id).

concretes_to_body([],_SgKey,_AbsInt,[]).
concretes_to_body([Sg|Sgs],SgKey,AbsInt,[B|Bs]):-
        body_info0(Sg:SgKey,[],AbsInt,B),
        concretes_to_body(Sgs,SgKey,AbsInt,Bs).

%------------------------------------------------------------------------%
% query(+,+,+,+,+,+,+,-)                                                 %
%------------------------------------------------------------------------%

:- doc(query(AbsInt,QKey,Query,Qv,RFlag,N,Call,Succ),
        "The success pattern of @var{Query} with @var{Call} is
         @var{Succ} in the analysis domain @var{AbsInt}. The predicate
         called is identified by @var{QKey}, and @var{RFlag} says if it
         is recursive or not. The goal @var{Query} has variables @var{Qv},
         and the call pattern is uniquely identified by @var{N}.").

query(AbsInt,QKey,Query,Qv,RFlag,N,Call,Succ) :-
        project(AbsInt,Query,Qv,Qv,Call,Proj),
        call_to_success(RFlag,QKey,Call,Proj,Query,Qv,AbsInt,0,Succ,_,N,0,Id), !,
        approx_to_completes(AbsInt).

query(_AbsInt,_QKey,_Query,_Qv,_RFlag,_N,_Call,_Succ):-
        % should never happen, but...
        error_message("SOMETHING HAS FAILED!").
```