```
                %%%%%%%%%%%%%%%%%%%%
                % Analysstrategier %
                %%%%%%%%%%%%%%%%%%%%

% Generellt:
% analys \\- forbjud(_),...,tillat(_),...,
%        sats(Ta,[vert(F1,L1),vert(F2,L2),...,vert(Fn,Ln)]) \- _.
%       med F1,L1,...,Ln instansierade
%       och F2,...,Fn variabler
analys <=
        start(sats(identifiera_ackordton,fordubblingS,harmoA)).

% harmo - analys:
harmoA <=
        harmo(und_pil_false,ackordtoner,narmast,dubb_kravA,fordubblingS,hitta_p3,
         ingen_stamkorsning3K,baston,stamforing_OKA).


                %%%%%%%%%%%%%%%%%%%%%%
                % Kontrollstrategier %
                %%%%%%%%%%%%%%%%%%%%%%

% Generellt:
% kontroll \\- forbjud(_),...,tillat(_),...,
%        sats(Ta,[vert(F1,L1),vert(F2,L2),...,vert(Fn,Ln)]) \- _.
%       med F1,...,Fn,L1,...,Ln instansierade.
kontroll <=
        start(sats(identifiera_ackordtonK,fordubblingK,harmoK)).

% harmo - kontroll (allmanna fallet):
harmoK <=
        format('~nKontroll av ackord nr ~w - ~w~n',[N,N1])
        -> (_ \- harmo(_,vert(_,_,pos(N)),vert(_,_,pos(N1)),_,_)).
harmoK <=
        harmo(und_pil_falseK,ackordtoner,narmastK,dubb_krav,fordubblingK,hitta_parK,
         ingen_stamkorsning3K,bastonK,stamforing_OKK).


                %%%%%%%%%%
                % Regler %
                %%%%%%%%%%

pd_left(T,I,PT) <=
        not(functor(T,sats,2)),
        not(data(T)),
        atom(T),
        definiens(T,Dp,N),
        (PT -> (I@[Dp|Y] \- C))
        -> (I@[T|Y] \- C).

sats_d_left(T,I,PT) <=
        functor(T,sats,2),
        definiens(T,Dp,N),
        (PT -> ([Dp|Y] \- C))
        -> (I@[T|Y] \- C).


                %%%%%%%%%%
                % Proviso %
                %%%%%%%%%%

alltrue([]).
alltrue([true|Xs]) :- alltrue(Xs).
```

```
% Generell inledande strategi for simulering, kontroll och analys.
start(SatsTyp) <=
        pd_left(_,_,start(SatsTyp)),
        v_left(_,_,start(SatsTyp)),
        remadd(SatsTyp).

% Behandlar en sekvens av rem_def och add_def, fortsatter sedan med SatsTyp.
remadd(SatsTyp) <=
        rem_l(_,_,remadd(SatsTyp)),
        add_l(_,_,remadd(SatsTyp)),
        SatsTyp.

% sats \\- sats(Ta,L) \- S.
sats(Id_ack,Dubbl,Harmo) <=
        alltrue(I) -> (I@[sats(_,_)] \- _).
sats(Id_ack,Dubbl,Harmo) <=
        sats_d_left(_,_,a_left(_,_,sats1(Id_ack,Dubbl,Harmo),axiom(_,_,_))).

% Hjalpstrategi for sats:
        sats1(Identifiera_ackordton,Fordubbling,HarmoTyp) <=
                v8_right(_,till_fkn,a_right(_,ackordtoner),Identifiera_ackordton,
                 Fordubbling,Identifiera_ackordton,Identifiera_ackordton,
                 Identifiera_ackordton,a_right(_,ga_igenom(HarmoTyp))).

% ga_igenom \\- ga_igenom(T,L) \- S.
ga_igenom(HarmoTyp) <=
        ga_igenom1(HarmoTyp),
        ga_igenom2(HarmoTyp).

% Hjalpstrategier for ga_igenom:
        % Listan i andra argumentet bestar av fler an tva verts.
        ga_igenom1(HarmoTyp) <=
                (I@[ga_igenom(_,[vert(_,_,_),vert(_,_),vert(_,_)|_])|R] \- _).
        ga_igenom1(HarmoTyp) <=
                d_left(_,_,a_left(_,_,v3_right(_,a_right(_,plus_left(_,_)),
                 HarmoTyp,a_right(_,ga_igenom(HarmoTyp))),axiom(_,_,_))).

        % Listan i andra argumentet bestar av tva verts.
        ga_igenom2(HarmoTyp) <=
                (I@[ga_igenom(_,[vert(_,_,_),vert(_,_)])|R] \- _).
        ga_igenom2(HarmoTyp)<=
                d_left(_,_,a_left(_,_,v_right(_,a_right(_,plus_left(_,_)),HarmoTyp),
                 axiom(_,_,_))).

% harmo \\- A \- harmo(T,V1,V2,V3,R).
harmo(Und_pil_false,Ackordtoner,Narmast,Dubb_krav,Fordubbling,Hitta_par,
                Ingen_stamkorsning3,Baston,Stamforing_OK) <=
        d_right(_,v11_right(_,till_fkn,till_fkn,Und_pil_false,
         a_right(_,Ackordtoner),Narmast,Dubb_krav,Fordubbling,Hitta_par,
         Ingen_stamkorsning3,Baston,Stamforing_OK)).

harmoS <=
        harmo(und_pil_false,ackordtoner,narmast,dubb_krav,fordubblingS,hitta_par,
         ingen_stamkorsning3,baston,stamforing_OK).

% transponera_sats \\- transponera_sats(R,L1,It,I) \- L2.
transponera_sats <=
        d_left(_,_,a_left(_,_,v5_right(_,hoga_tonen,hoga_tonen,hoga_tonen,
         hoga_tonen,a_right(_,transponera_sats)),axiom(_,_,_))),
        d_left(_,_,axiom(_,_,_)).
```

```
tillat(Dubbleringstyp,F)#{F \= fkn(_,_,_)} <=
        till_fkn(F,FKN),
        rem_def(undantag(Dubbleringstyp,FKN),true).
tillat(Dubbleringstyp,fkn(F,D,B)) <=
        rem_def(undantag(Dubbleringstyp,fkn(F,D,B)),true).

% transponera_sats(Riktning,Sats,Ityp,Int) transponerar en Sats, given i form av
% en lista av verts. Transponeringen sker i given Riktning (upp eller ner)
% samt givet intervall, Ityp+Int.
transponera_sats(_,[],_,_) <=
        [].
transponera_sats(upp,[vert(F,[B,T,A,S])|Resten],Ityp,Int) <=
        hoga_tonen(B,Ityp,Int,B1),
        hoga_tonen(T,Ityp,Int,T1),
        hoga_tonen(A,Ityp,Int,A1),
        hoga_tonen(S,Ityp,Int,S1),
        (transponera_sats(upp,Resten,Ityp,Int) -> Trans)
        -> [[B1,T1,A1,S1]|Trans].
transponera_sats(ner,[vert(F,[B,T,A,S])|Resten],Ityp,Int) <=
        laga_tonen(B,Ityp,Int,B1),
        laga_tonen(T,Ityp,Int,T1),
        laga_tonen(A,Ityp,Int,A1),
        laga_tonen(S,Ityp,Int,S1),
        (transponera_sats(ner,Resten,Ityp,Int) -> Trans)
        -> [[B1,T1,A1,S1]|Trans].

% Givet en «melodi» (lista av toner) [L|Ls] resp [H|Hs],
%       en riktning (upp eller ner)
%       samt ett intervall Ityp+Int,
% ges som resultat samma «melodi», transponerad i given riktning
% det antal steg som intervallet anger.
transponera([],_,_,_,[]).
transponera([L|Ls],upp,Ityp,Int,[H|Hs]) <=
        hoga_tonen(L,Ityp,Int,H),
        transponera(Ls,upp,Ityp,Int,Hs).
transponera([H|Hs],ner,Ityp,Int,[L|Ls]) <=
        laga_tonen(H,Ityp,Int,L),
        transponera(Hs,ner,Ityp,Int,Ls).
```

# topp.rul

```
% Nedanstaende filer behovs:
% :- include_rules('bas.rul').
% :- include_rules('bygg.rul').
% :- include_rules('grundregler.rul').
% :- include_rules('kbs.rul').
% :- include_rules('stamforing.rul').
% :- include_rules('ton.rul').

                %%%%%%%%%%%%%%
                % Strategier %
                %%%%%%%%%%%%%%

% Generellt:
% simulering \\- forbjud(_),...,tillat(_),...,
%                sats(Ta,[vert(F1,L1),vert(F2,L2),...,vert(Fn,Ln)]) \- L.
%       med L1,F1,...,Fn instansierade
%       och L2,...,Ln delvis oinstansierade (i regel ar sopranstamman instansierad)
simulering <=
        start(sats(identifiera_ackordton,fordubblingS,harmoS)).
```

```
% Givet Tonart och en lista av tva eller flera verts sa ges en fyrstammig sats enligt
% den definition som ges av listan med verts. Det forsta elementet i listan av verts
% bor vara fullt instansierat. Dessutom infors en inre numrering av de ackord
% som listan av verts representerar.
sats(Tonart,[vert(F,[B,T,A,S])|Resten]) <=
        (till_fkn(F,FKN),
        (ackordtoner(Tonart,FKN) -> Acks),
        identifiera_ackordton(S,Acks,_,pos(1)),
        fordubbling(_,par(FKN,S),Acks,val(bas(B1),[AT1,AT2]),par([B,T,A],pos(1)),R),
        identifiera_ackordton(AT1,[T,A],Ton,pos(1)),
        identifiera_ackordton(AT2,removefirst(Ton,[T,A]),_,pos(1)),
        identifiera_ackordton(B,[B1],B1,pos(0)),                          % Ratt baston?
                        % pos(0) anvands av implementeringstekniska skal (for felutskrift).
        (ga_igenom(Tonart,[vert(F,[B,T,A,S],pos(1))|Resten]) -> Svar)) -> Svar.

% Stegar sig igenom en lista av verts och tillampar harmo pa vart och ett av dem.
ga_igenom(Tonart,[vert(F1,T1,pos(N)),vert(F2,T2)]) <=
        ((N+1 -> N1),
        harmo(Tonart,vert(F1,T1,pos(N)),vert(F2,T2,pos(N1)),vert(tom,tom),Regel))
        -> [T1,T2].
ga_igenom(Tonart,[vert(F1,T1,pos(N)),vert(F2,T2),vert(F3,T3)|Xs]) <=
        ((N+1 -> N1),
        harmo(Tonart,vert(F1,T1,pos(N)),vert(F2,T2,pos(N1)),vert(F3,T3),Regel),
        (ga_igenom(Tonart,[vert(F2,T2,pos(N1)),vert(F3,T3)|Xs]) -> Resten))
        -> [T1|Resten].

% Givet att [B,T,A,S] ar instansierad sa ar [B1,T1,A1,S1] ett ackord
% sadant att det uppfyller de stamforingsregler som finns i programmet.
harmo(Ton,vert(FN1,[B,T,A,S],Pos),vert(FN2,[B1,T1,A1,S1],Pos1),vert(F3,T3),Dr) <=
        till_fkn(FN1,FKN1),
        till_fkn(FN2,FKN2),
        (undantag(otillaten_foljd(FKN1,FKN2)) -> false),
        (ackordtoner(Ton,FKN2) -> Acktoner),
        narmast(S,Acktoner,S1,Pos1),
        dubb_krav(Ton,par(FKN1,[B,T,A,S]),par(FKN2,[B1,T1,A1,S1]),par(F3,T3),Dr),
        fordubbling(FKN1,par(FKN2,S1),Acktoner,val(bas(Bas),Kvar),
         par([B1,T1,A1],Pos1),Dr),
        hitta_par([A,T],Kvar,[[A,A1],[T,T1]]),
        ingen_stamkorsning(T1,A1,S1),  % Behovs for effektiviteten.
        baston(FKN1,FKN2,T1,B,Bas,B1),
        stamforing_OK(Ton,par(FKN1,FKN2),par([B,T,A,S],Pos),par([B1,T1,A1,S1],Pos1)).

% Tar bort och lagger till definitioner for att andra programmets beteende.
tillat(oktavparalleller) <=
        rem_def((oktav_paraller(_,_) <= okp(_,_)),true).
tillat(kvintparalleller) <=
        rem_def((kvint_paraller(_,_) <= kp(_,_)),true).
tillat(medrorelse) <=
        rem_def((medror(X,Y) <= mr(X,Y)),true).
tillat(alla_intervall) <=
        rem_def((tillatna_intervall(X,Y,Z) <= W),
         add_def((tillatna_intervall(X,Y,[[T,I]])),true)).

forbjud(Dubbleringstyp,F)#{F \= fkn(_,_,_), F \= pos(N)} <=
        till_fkn(F,FKN),
        add_def(undantag(Dubbleringstyp,FKN),true).
forbjud(Dubbleringstyp,fkn(F,T,B)) <=
        add_def(undantag(Dubbleringstyp,fkn(F,T,B)),true).
forbjud(ackord([B,T,A,S]),pos(N)) <=
        add_def(undantag(ackord([B,T,A,S]),pos(N)),true).
```

```
% val \\- A \- val(Par1,Par2,T).
val <=
        d_right(_,jfr).


                %%%%%%%%%%%%%%%%%%%%
                % Kontrollstrategi %
                %%%%%%%%%%%%%%%%%%%%

% hoga_tonen, laga_tonen - kontroll:
hoga_tonenK <=
        hoga_tonen,
        hoga_tonenE,
        laga_tonenE.


                %%%%%%%%%%%%%%%%%%%%%%%%%%
                % Regler for felutskrift %
                %%%%%%%%%%%%%%%%%%%%%%%%%%

% hoga_tonen - felutskrift:
hoga_tonenE <=
        (hoga_tonen -> (_ \- hoga_tonen(Lag,Ityp,Int,Ratt))),
        format('~nFelaktigt intervall: ~w - ~w ar ingen ~w ~w.~n',[Lag,Hog,Ityp,Int]),
        format('Den hogre tonen skall vara ~w.~n',[Ratt])
        -> (_ \- hoga_tonen(Lag,Ityp,Int,Hog)).

% laga_tonen - felutskrift:
laga_tonenE <=
        (hoga_tonen -> (_ \- laga_tonen(Hog,Ityp,Int,Ratt))),
        format('~nFelaktigt intervall: ~w - ~w ar ingen ~w ~w.~n',[Lag,Hog,Ityp,Int]),
        format('Den lagre tonen skall vara ~w.~n',[Ratt])
        -> (_ \- laga_tonen(Hog,Ityp,Int,Lag)).
```

# topp.def

```
% Nedanstaende filer behovs:
% :- include_def('bygg.def').
% :- include_def('kbs.def').
% :- include_def('stamforing.def').
% :- include_def('ton.def').

% Strategierna finns i filen 'topp.rul'.

% Definitioner som anvands utat:
%
% simulering \\- forbjud(_),...,tillat(_),...,sats(Ta,L) \- S.
% kontroll \\- forbjud(_),...,tillat(_),...,sats(Ta,L) \- S.
% analys \\- forbjud(_),...,tillat(_),...,sats(Ta,L) \- S.
% sats \\- sats(Ta,L) \- S.
% transponera_sats \\- A,transponera_sats(R,L1,It,I) \- L2.
% transponera \\- A \- transponera(L1,R,It,I,L2).
%
% A star for godtycklig antecedent. Ingen hansyn tas till denna.
```

```
                %%%%%%%%%%%%%%
                % Strategier %
                %%%%%%%%%%%%%%

% ton \\- A \- ton(T,X).
% ton \\- A \- stamton(T1,T2).
ton <=
        d_right(_,dt_right(_)).

% dist \\- A \- dist(T1,T2,X).
dist <=
        d_right(_,v3_right(_,ton(_),ton(_),a_right(_,minus_left(_,_)))).

% distton \\- A \- distton(T1,X,T2).
% distton \\- A \- distton_ner(T1,X,T2).
distton <=
        d_right(_,v3_right(_,ton(_),a_right(_,plus_minus),ton(_))).

% under \\- A \- under(T1,T2).
% under \\- A \- over(T1,T2).
% under \\- A \- str_over(T1,T2).
% under \\- A \- str_under(T1,T2).
% under \\- A \- olika(T1,T2).
under <=
        d_right(_,v3_right(_,ton(_),ton(_),jfr)).

% intervall \\- A \- intervall(T1,T2,It,I).
intervall <=
        d_right(_,v7_right(_,ton(_),ton(_),dt_right(_),dt_right(_),
         stam_intervall,dist,dist_intervall)).

% hoga_tonen \\- A \- hoga_tonen(T1,It,I,T2).
% hoga_tonen \\- A \- laga_tonen(T1,It,I,T2).
hoga_tonen <=
        d_right(_,v9_right(_,dt_right(_),dt_right(_),a_right(_,plus_left(_,_)),
         ton(_),dt_right(_),a_right(_,plus_minus),dt_right(_),distton,ton(_))).

% stam_intervall \\- A \- stam_intervall(N1,Ityp,Int,N2).
stam_intervall <=
        d_right(_,stam_intervall(_)).

% Hjalpstrategi for stam_intervall/0:
        stam_intervall(C)<=
                (_ \- C).
        stam_intervall(true)<=
                true_right.
        stam_intervall(((_ -> _),stam_intervall(_,_,_,_))) <=
                v_right(_,a_right(_,minus_left(_,_)),dt_right(_)).

% dist_intervall \\- A \- dist_intervall(N1,R_S,Ityp,Int).
dist_intervall <=
        d_right(_,dist_intervall(_)).

% Hjalpstrategi for dist_intervall/0:
        dist_intervall(C)<=
                (_ \- C).
        dist_intervall(true)<=
                true_right.
        dist_intervall(((_ -> _),dist_intervall(_,_,_)))<=
                v_right(_,a_right(_,minus_left(_,_)),dt_right(_)).

% mest_nara \\- A \- mest_nara(T1,T2,T3,T4).
mest_nara <=
        d_right(_,v5_right(_,dist,dist,a_right(_,abs_left(_,_)),
         a_right(_,abs_left(_,_)),val)).
```

```
% For varje intervall (prim, sekund, ...) finns
%       - ett internt intervallnummer (0, 1, 2, 3, ...)
%       - grundformen for intervallet (ren eller stor)
%       - ett grundnummer raknat i halvtonsteg (0, 2, 4, 5, ...)
stam_intervall(0,ren,prim,0).
stam_intervall(1,stor,sekund,2).
stam_intervall(2,stor,ters,4).
stam_intervall(3,ren,kvart,5).
stam_intervall(4,ren,kvint,7).
stam_intervall(5,stor,sext,9).
stam_intervall(6,stor,septim,11).
stam_intervall(7,ren,oktav,12).
stam_intervall(8,stor,nona,14).
stam_intervall(9,stor,decima,16).
stam_intervall(10,ren,undecima,17).
stam_intervall(11,ren,duodecima,19).
stam_intervall(X,It,I,N)#{X \= 1, X \= 2, X \= 3, X \= 4, X \= 5,
                X \= 6, X \= 7, X \= 8, X \= 9, X \= 10, X \= 11} <=
        (X -> In),
        stam_intervall(In,It,I,N).

% Namnkonventioner for alteration av tonerna i ett intervall.
% (Forklaring av forsta raden:
%       Ett rent intervall som forminskats tva halvtonsteg
%       kallas dubbelt forminskat.)
dist_intervall(-2,ren,dubbelt_forminskad).
dist_intervall(-2,stor,forminskad).
dist_intervall(-1,ren,forminskad).
dist_intervall(-1,stor,liten).
dist_intervall(0,R_S,R_S).
dist_intervall(1,_,overstigande).
dist_intervall(2,_,dubbelt_overstigande).
dist_intervall(X,It1,It2)#{X \= -2, X \= -1, X \= 0, X \= 1, X \= 2} <=
        (X -> In),
        dist_intervall(In,It1,It2).

% Givet tre toner - Ton, Prov1 och Prov2 - sa ar Val
% den av tonerna Prov1 och Prov2 som ligger narmast Ton.
% Vid backtracking ges aven den andra tonen som Val.
mest_nara(Ton,Prov1,Prov2,Val) <=
        dist(Ton,Prov1,TP1),
        dist(Ton,Prov2,TP2),
        (abso(TP1) -> Atp1),
        (abso(TP2) -> Atp2),
        val(par(Prov1,Atp1),par(Prov2,Atp2),Val).

% Hjalp till mest_nara.
val(par(T1,D1),par(T2,D2),T1) <=
        D1 =< D2.
val(par(T1,D1),par(T2,D2),T2) <=
        D1 > D2.
val(par(T1,D1),par(T2,D2),T1) <=
        D1 > D2.
val(par(T1,D1),par(T2,D2),T2) <=
        D1 =< D2.
```

## ton.rul

```
% Nedanstaende filer behovs:
% :- include_rules('bas.rul').
% :- include_rules('grundregler.rul').
```

```
% Givet tva toner, Ton1 resp Ton2, sa ar over(Ton1,Ton2)
% harledbart om Ton1 har en hogre eller lika hog frekvens som Ton2.
over(Ton1,Ton2) <=
        ton(Ton1,X1),
        ton(Ton2,X2),
        X2 =< X1.

% Givet tva toner, Ton1 resp Ton2, sa ar over(Ton1,Ton2)
% harledbart om Ton1 har en hogre frekvens an Ton2.
str_over(Ton1,Ton2) <=
        ton(Ton1,X1),
        ton(Ton2,X2),
        X1 > X2.

% Givet tva toner, Ton1 resp Ton2, sa ar over(Ton1,Ton2)
% harledbart om Ton1 har en lagre frekvens an Ton2.
str_under(Ton1,Ton2) <=
        ton(Ton1,X1),
        ton(Ton2,X2),
        X2 > X1.

% Givet tva toner, Ton1 resp Ton2, sa ar over(Ton1,Ton2)
% harledbart om Ton1 har en annan frekvens an Ton2.
olika(Ton1,Ton2) <=
        ton(Ton1,X1),
        ton(Ton2,X2),
        X1 \= X2.

% Givet tva toner Lag och Hog sa ar Ityp och Int
% intervalltypen resp intervallnamnet for intervallet mellan dessa.
% Forutsattning: Lag ar en lagre ton an Hog
intervall(Lag,Hog,Ityp,Int) <=
        stamton(Lag,S1),
        stamton(Hog,S2),
        vit(S1,VN1),
        vit(S2,VN2),
        stam_intervall(VN2 - VN1,R_S,Int,Stamdist),
        dist(Lag,Hog,D),
        dist_intervall(D - Stamdist,R_S,Ityp).

% Givet en ton Lag och ett intervall Ityp+Int ar Hog den ton
% som ligger givet intervall hogre an Lag
hoga_tonen(Lag,Ityp,Int,Hog) <=
        stam_intervall(VD,R_S,Int,Stamdist),
        dist_intervall(N,R_S,Ityp),
        (N + Stamdist -> D),
        stamton(Lag,S1),
        vit(S1,VN1),
        (VN1 + VD -> VN2),
        vit(S2,VN2),
        distton(Lag,D,Hog),
        stamton(Hog,S2).

% Givet en ton Hog och ett intervall Ityp+Int ar Lag den ton
% som ligger givet intervall lagre an Hog
laga_tonen(Hog,Ityp,Int,Lag) <=
        stam_intervall(VD,R_S,Int,Stamdist),
        dist_intervall(N,R_S,Ityp),
        (N + Stamdist -> D),
        stamton(Hog,S2),
        vit(S2,VN2),
        (VN2 - VD -> VN1),
        vit(S1,VN1),
        distton_ner(Hog,D,Lag),
        stamton(Lag,S1).
```

```
vit(gs,12).
vit(as,13).
vit(hs,14).
vit(c,15).
vit(d,16).
vit(e,17).
vit(f,18).
vit(g,19).
vit(a,20).
vit(h,21).
vit(c1,22).
vit(d1,23).
vit(e1,24).
vit(f1,25).
vit(g1,26).
vit(a1,27).
vit(h1,28).
vit(c2,29).
vit(d2,30).
vit(e2,31).
vit(f2,32).
vit(g2,33).
vit(a2,34).
vit(h2,35).
vit(c3,36).
vit(d3,37).
vit(e3,38).
vit(f3,39).
vit(g3,40).
vit(a3,41).
vit(h3,42).
vit(c4,43).

% Givet tva toner, Ton1 resp Ton2, sa ar Avstand antalet halvtonsteg
% mellan tonerna. Om Ton1 ar en hogre ton an Ton2, fas ett negativt varde.
dist(Ton1,Ton2,Avstand) <=
        ton(Ton1,N1),
        ton(Ton2,N2),
        (N2-N1 -> Avstand).

% Givet en ton Ton1 och ett Avstand sa ar Ton2 den ton
% som ligger Avstand halvtonsteg hogre an Ton1.
distton(Ton1,Avstand,Ton2) <=
        ton(Ton1,N1),
        (Avstand+N1 -> N2),
        ton(Ton2,N2).

% Givet en ton Ton1 och ett Avstand sa ar Ton2 den ton
% som ligger Avstand halvtonsteg lagre an Ton1.
distton_ner(Ton1,Avstand,Ton2) <=
        ton(Ton1,N1),
        (N1-Avstand -> N2),
        ton(Ton2,N2).

% Givet tva toner, Ton1 resp Ton2, sa ar under(Ton1,Ton2)
% harledbart om Ton1 har en lagre eller lika hog frekvens som Ton2.
under(Ton1,Ton2) <=
        ton(Ton1,X1),
        ton(Ton2,X2),
        X1 =< X2.
```

```
alter(c2,'c2#',50).
alter(d2,d2b,50).
alter(d2,d2,51).
alter(d2,'d2#',52).
alter(e2,e2b,52).
alter(e2,e2,53).
alter(e2,'e2#',54).
alter(f2,f2b,53).
alter(f2,f2,54).
alter(f2,'f2#',55).
alter(g2,g2b,55).
alter(g2,g2,56).
alter(g2,'g2#',57).
alter(a2,a2b,57).
alter(a2,a2,58).
alter(a2,'a2#',59).
alter(h2,b2,59).
alter(h2,h2,60).
alter(h2,'h2#',61).
alter(c3,c3b,60).
alter(c3,c3,61).
alter(c3,'c3#',62).
alter(d3,d3b,62).
alter(d3,d3,63).
alter(d3,'d3#',64).
alter(e3,e3b,64).
alter(e3,e3,65).
alter(e3,'e3#',66).
alter(f3,f3b,65).
alter(f3,f3,66).
alter(f3,'f3#',67).
alter(g3,g3b,67).
alter(g3,g3,68).
alter(g3,'g3#',69).
alter(a3,a3b,69).
alter(a3,a3,70).
alter(a3,'a3#',71).
alter(h3,b3,71).
alter(h3,h3,72).
alter(h3,'h3#',73).
alter(c4,c4b,72).
alter(c4,c4,73).

% Tonen Ton har det interna numret N.
ton(Ton,N) <=
        alter(_,Ton,N).

% Givet en ton Ton1 sa ar Ton2 dess stamton.
% Givet stamtonen Ton2, antar Ton1 vardet av dess «hojningar och sankningar».
stamton(Ton1,Ton2) <=
        alter(Ton2,Ton1,_).

% En intern representation av stamtonerna
% i form av en uppraknings-relation.
vit(ck,1).
vit(dk,2).
vit(ek,3).
vit(fk,4).
vit(gk,5).
vit(ak,6).
vit(hk,7).
vit(cs,8).
vit(ds,9).
vit(es,10).
vit(fs,11).
```

```
alter(cs,cs,13).
alter(cs,csb,12).
alter(cs,'cs#',14).
alter(ds,dsb,14).
alter(ds,ds,15).
alter(ds,'ds#',16).
alter(es,esb,16).
alter(es,es,17).
alter(es,'es#',18).
alter(fs,fsb,17).
alter(fs,fs,18).
alter(fs,'fs#',19).
alter(gs,gsb,19).
alter(gs,gs,20).
alter(gs,'gs#',21).
alter(as,asb,21).
alter(as,as,22).
alter(as,'as#',23).
alter(hs,bs,23).
alter(hs,hs,24).
alter(hs,'hs#',25).
alter(c,cb,24).
alter(c,c,25).
alter(c,'c#',26).
alter(d,db,26).
alter(d,d,27).
alter(d,'d#',28).
alter(e,eb,28).
alter(e,e,29).
alter(e,'e#',30).
alter(f,fb,29).
alter(f,f,30).
alter(f,'f#',31).
alter(g,gb,31).
alter(g,g,32).
alter(g,'g#',33).
alter(a,ab,33).
alter(a,a,34).
alter(a,'a#',35).
alter(h,b,35).
alter(h,h,36).
alter(h,'h#',37).
alter(c1,c1b,36).
alter(c1,c1,37).
alter(c1,'c1#',38).
alter(d1,d1b,38).
alter(d1,d1,39).
alter(d1,'d1#',40).
alter(e1,e1b,40).
alter(e1,e1,41).
alter(e1,'e1#',42).
alter(f1,f1b,41).
alter(f1,f1,42).
alter(f1,'f1#',43).
alter(g1,g1b,43).
alter(g1,g1,44).
alter(g1,'g1#',45).
alter(a1,a1b,45).
alter(a1,a1,46).
alter(a1,'a1#',47).
alter(h1,b1,47).
alter(h1,h1,48).
alter(h1,'h1#',49).
alter(c2,c2b,48).
alter(c2,c2,49).
```

```
% Lyckas om det foreligger kvintparalleller
% mellan X1,X2 och Y1,Y2.
ktp(X1,X2,Y1,Y2) :-
        Noll is ((abs(((X2 - X1) mod 12) - 7)) + (abs(((Y2 - Y1) mod 12) - 7))),
        Noll = 0,
        ((X1 \= Y1);(X2 \= Y2)).

% Konstruerare.
constructor(okp,2).
constructor(kp,2).
constructor(mr,2).
```

# ton.def

```
% Strategierna finns i filen ‘ton.rul’.

% Definitioner som anvands utat:
%
% ton \\- A \- ton(T,X).
% ton \\- A \- stamton(T1,T2).
% Istallet for strategin ton kan regeln ton\1 anvandas:
%       ton(_) \\- A \- ton(T,X).
%       ton(_) \\- A \- stamton(T1,T2).
% dist \\- A \- dist(T1,T2,X).
% distton \\- A \- distton(T1,X,T2).
% distton \\- A \- distton_ner(T1,X,T2).
% under \\- A \- under(T1,T2).
% under \\- A \- over(T1,T2).
% under \\- A \- str_over(T1,T2).
% under \\- A \- str_under(T1,T2).
% under \\- A \- olika(T1,T2).
% intervall \\- A \- intervall(T1,T2,It,I).
% hoga_tonen \\- A \- hoga_tonen(T1,It,I,T2).
% hoga_tonen \\- A \- laga_tonen(T1,It,I,T2).
% mest_nara \\- A \- mest_nara(T1,T2,T3,T4).
% val \\- A \- val(Par1,Par2,T).
%
% A star for godtycklig antecedent. Ingen hansyn tas till denna.


% For varje ton (cs, ‘cs#’, dsb, ...) finns
%       en stamton (cs, cs, ds, ...)
%       och ett internt nummer raknat i halvtonsteg (1, 2, 3, ...)
alter(ck,ck,1).
alter(ck,’ck#’,2).
alter(dk,dkb,2).
alter(dk,dk,3).
alter(dk,’dk#’,4).
alter(ek,ekb,4).
alter(ek,ek,5).
alter(ek,’ek#’,6).
alter(fk,fkb,5).
alter(fk,fk,6).
alter(fk,’fk#’,7).
alter(gk,gkb,7).
alter(gk,gk,8).
alter(gk,’gk#’,9).
alter(ak,akb,9).
alter(ak,ak,10).
alter(ak,’ak#’,11).
alter(hk,bk,11).
alter(hk,hk,12).
alter(hk,’hk#’,13).
```

```
                %%%%%%%%%%
                % Regler %
                %%%%%%%%%%

% Regel for att avgora om oktavparalleller forekommer.
par_eights_left(okp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]),PT) <=
        number(Bn),number(Tn),number(An),number(Sn),
        number(B1n),number(T1n),number(A1n),number(S1n),
        (not(oktp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n])),(PT -> (I@[false|R] \- C));
        oktp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]),(PT -> (I@[true|R] \- C)))
        -> (I@[okp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n])|R] \- C).

% Regel for att avgora om kvintparalleller forekommer.
par_fifths_left(kp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]),PT) <=
        number(Bn),number(Tn),number(An),number(Sn),
        number(B1n),number(T1n),number(A1n),number(S1n),
        (not(ktp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n])),(PT -> (I@[false|R] \- C));
        ktp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]),(PT -> (I@[true|R] \- C)))
        -> (I@[kp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n])|R] \- C).

% Regel for att avgora om medrorelse foreligger.
par_move_left(mr([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]),PT) <=
        number(Bn),number(Tn),number(An),number(Sn),
        number(B1n),number(T1n),number(A1n),number(S1n),
        ((((Bn < B1n,Tn < T1n,An < A1n ,Sn < S1n);
        (Bn > B1n,Tn > T1n,An > A1n ,Sn > S1n)),
        (PT -> (I@[true|R] \- C)));
        (not(((Bn < B1n,Tn < T1n,An < A1n ,Sn < S1n);
        (Bn > B1n,Tn > T1n,An > A1n ,Sn > S1n))),
        (PT -> (I@[false|R] \- C))))
        -> (I@[mr([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n])|R] \- C).


                %%%%%%%%%%%
                % Provison %
                %%%%%%%%%%%

% Lyckas om oktavparalleller forekommer
% mellan ackorden [Bn,Tn,An,Sn] och [B1n,T1n,A1n,S1n].
oktp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) :-
        oktp(Bn,Tn,B1n,T1n);
        oktp(Bn,An,B1n,A1n);
        oktp(Bn,Sn,B1n,S1n);
        oktp(Tn,An,T1n,A1n);
        oktp(Tn,Sn,T1n,S1n);
        oktp(An,Sn,A1n,S1n).

% Lyckas om det foreligger oktavparalleller
% mellan X1,X2 och Y1,Y2.
oktp(X1,X2,Y1,Y2) :-
        Noll is ((abs((X2 - X1) mod 12)) + (abs((Y2 - Y1) mod 12))),
        Noll = 0,
        ((X1 \= Y1);( X2 \= Y2)).

% Lyckas om kvintparalleller forekommer
% mellan ackorden [Bn,Tn,An,Sn] och [B1n,T1n,A1n,S1n].
ktp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) :-
        ktp(Bn,Tn,B1n,T1n);
        ktp(Bn,An,B1n,A1n);
        ktp(Bn,Sn,B1n,S1n);
        ktp(Tn,An,T1n,A1n);
        ktp(Tn,Sn,T1n,S1n);
        ktp(An,Sn,A1n,S1n).
```

```
% ej_medrorelse - felutskrift:
ej_medrorelseE <=
        (((ton(_) -> (_ \- ton(S,_))),
        (ton(_) -> (_ \- ton(A,_))),
        (ton(_) -> (_ \- ton(T,_))),
        (ton(_) -> (_ \- ton(B,_))),
        (ton(_) -> (_ \- ton(S1,_))),
        (ton(_) -> (_ \- ton(A1,_))),
        (ton(_) -> (_ \- ton(T1,_))),
        (ton(_) -> (_ \- ton(B1,_))),
        format('Medrorelse mellan ackord nr ~w och ~w:~n',[N,N1]),
        format(' Sopran ~w - ~w~n Alt ~w - ~w~n',[S,S1,A,A1]),
        format(' Tenor ~w - ~w~n Bas ~w - ~w~n',[T,T1,B,B1]))
        -> (_ \- ej_medrorelse(par([B,T,A,S],pos(N)),
                        par([B1,T1,A1,S1],pos(N1))))).

% inga_oktpar - felutskrift:
inga_oktparE <=
        (((ton(_) -> (_ \- ton(S,_))),
        (ton(_) -> (_ \- ton(A,_))),
        (ton(_) -> (_ \- ton(T,_))),
        (ton(_) -> (_ \- ton(B,_))),
        (ton(_) -> (_ \- ton(S1,_))),
        (ton(_) -> (_ \- ton(A1,_))),
        (ton(_) -> (_ \- ton(T1,_))),
        (ton(_) -> (_ \- ton(B1,_))),
        format('Oktavparallell mellan ackord nr ~w och ~w:~n',[N,N1]),
        format(' Sopran ~w - ~w~n Alt ~w - ~w~n',[S,S1,A,A1]),
        format(' Tenor ~w - ~w~n Bas ~w - ~w~n',[T,T1,B,B1]))
        -> (_ \- inga_oktpar(par([B,T,A,S],pos(N)),
                        par([B1,T1,A1,S1],pos(N1))))).

% inga_kvintpar - felutskrift:
inga_kvintparE <=
        (((ton(_) -> (_ \- ton(S,_))),
        (ton(_) -> (_ \- ton(A,_))),
        (ton(_) -> (_ \- ton(T,_))),
        (ton(_) -> (_ \- ton(B,_))),
        (ton(_) -> (_ \- ton(S1,_))),
        (ton(_) -> (_ \- ton(A1,_))),
        (ton(_) -> (_ \- ton(T1,_))),
        (ton(_) -> (_ \- ton(B1,_))),
        format('Kvintparallell mellan ackord nr ~w och ~w:~n',[N,N1]),
        format(' Sopran ~w - ~w~n Alt ~w - ~w~n',[S,S1,A,A1]),
        format(' Tenor ~w - ~w~n Bas ~w - ~w~n',[T,T1,B,B1]))
        -> (_ \- inga_kvintpar(par([B,T,A,S],pos(N)),
                        par([B1,T1,A1,S1],pos(N1))))).

% godkant_intervall - felutskrift:
godkant_intervallE <=
        (((ton(_) -> (_ \- ton(Ton1,_))),
        (ton(_) -> (_ \- ton(Ton2,_))),
        format('Otillatet intervall i ~wstamman mellan ackord nr ',[S]),
        format('~w och ~w: ~w - ~w~n',[N,N1,Ton1,Ton2]))
        -> (_ \- godkant_intervall(S,par(Ton1,pos(N)),par(Ton2,pos(N1)),_))).
```

```
% inga_oktpar - kontroll:
inga_oktparK <=
        inga_oktpar,
        inga_oktparE,
        skip.

% inga_kvintpar - kontroll:
inga_kvintparK <=
        inga_kvintpar,
        inga_kvintparE,
        skip.

% intervall_OK - kontroll:
intervall_OKK <=
        d_right(_,v4_right(_,godkant_intervallK,godkant_intervallK,
         godkant_intervallK,godkant_intervallK)).

% godkant_intervall - kontroll:
godkant_intervallK <=
        godkant_intervall,
        godkant_intervallE,
        skip.


              %%%%%%%%%%%%%%%%%%%%%%%%%
              % Regler for felutskrift %
              %%%%%%%%%%%%%%%%%%%%%%%%%

% spec_sf_krav - felutskrift:
spec_sf_kravE <=
        (format('Speciellt stamforingskrav ej uppfyllt~n',[]),
        format('mellan ackord nr ~w och ~w:~n',[N,N1]),
        format(' Sopran ~w - ~w~n Alt ~w - ~w~n',[S,S1,A,A1]),
        format(' Tenor ~w - ~w~n Bas ~w - ~w~n',[T,T1,B,B1]))
        -> (_ \- spec_sf_krav(_,_,par([B,T,A,S],pos(N)),
                 par([B1,T1,A1,S1],pos(N1)))).

% ingen_stamkorsning - felutskrift:
ingen_stamkorsningE <=
        (((ton(_) -> (_ \- ton(S,_))),
        (ton(_) -> (_ \- ton(A,_))),
        (ton(_) -> (_ \- ton(T,_))),
        (ton(_) -> (_ \- ton(B,_))),
        format('Stamkorsning i ackord nr ~w:~n',[N]),
        format(' Sopran ~w~n Alt ~w~n',[S,A]),
        format(' Tenor ~w~n Bas ~w~n',[T,B]))
        -> (_ \- ingen_stamkorsning(par([B,T,A,S],pos(N))))).

% omfang_OK - felutskrift:
omfang_OKE <=
        (((ton(_) -> (_ \- ton(Ton,_))),
        (dt_right(_) -> (_ \- tillatet_omfang(S,Lag,Hog))),
        format('Tonen ~w i ackord nr ~w ligger utanfor ~wstammans omfang.~n',
         [Ton,N,S]),
        format(' Tillatet omfang: ~w - ~w~n',[Lag,Hog]))
        -> (_ \- omfang_OK(S,Ton,pos(N)))).

% stamavst_OK - felutskrift:
stamavst_OKE <=
        (((ton(_) -> (_ \- ton(Ton1,_))),
        (ton(_) -> (_ \- ton(Ton2,_))),
        format('Otillatet intervall mellan ~w- och ~wstamman ',[S1,S2]),
        format('i ackord nr ~w: ~w - ~w~n',[N,Ton1,Ton2]))
        -> (_ \- stamavst_OK(S1,S2,Ton1,Ton2,pos(N)))).
```

```
                %%%%%%%%%%%%%%%%%%%%
                % Analysstrategier %
                %%%%%%%%%%%%%%%%%%%%

stamforing_OKA <=
        d_right(_,v9_right(_,und_pil_false,spec_sf_kravA,ingen_stamkorsningK,omfang_OKK,
         stamavstand_OKK,ej_medrorelseK,inga_oktparK,inga_kvintparK,intervall_OKK)).

spec_sf_kravA <=
        skip.


                %%%%%%%%%%%%%%%%%%%%%%
                % Kontrollstrategier %
                %%%%%%%%%%%%%%%%%%%%%%

% stamforing_OK - kontroll:
stamforing_OKK <=
        d_right(_,v9_right(_,und_pil_false,spec_sf_kravK,ingen_stamkorsningK,
         omfang_OKK,stamavstand_OKK,ej_medrorelseK,inga_oktparK,
         inga_kvintparK,intervall_OKK)).

% spec_sf_krav - kontroll:
spec_sf_kravK <=
        spec_sf_krav,
        spec_sf_kravE.

% ingen_stamkorsning/1 - kontroll:
ingen_stamkorsningK <=
        ingen_stamkorsning,
        ingen_stamkorsningE,
        skip.

% ingen_stamkorsning/3 - kontroll:
ingen_stamkorsning3K <=
        skip.

% omfang_OK/1 - kontroll:
omfang_OKK <=
        d_right(_,v4_right(_,omfang_OK3K,omfang_OK3K,omfang_OK3K,omfang_OK3K)).

% omfang_OK/3 - kontroll:
omfang_OK3K <=
        omfang_OK3,
        omfang_OKE,
        skip.

% stamavstand_OK - kontroll:
stamavstand_OKK <=
        d_right(_,v_right(_,stamavst_OKK,stamavst_OKK)).

% stamavst_OK - kontroll:
stamavst_OKK <=
        stamavst_OK,
        stamavst_OKE,
        skip.

% ej_medrorelse - kontroll:
ej_medrorelseK <=
        ej_medrorelse,
        ej_medrorelseE,
        skip.
```

```
% ingen_stamkorsning \\- A \- ingen_stamkorsning(Par).
ingen_stamkorsning <=
        d_right(_,v3_right(_,under,under,under)).

% ingen_stamkorsning3 \\- A \- ingen_stamkorsning(T1,T2,T3).
ingen_stamkorsning3 <=
        d_right(_,v_right(_,under,under)).

% omfang_OK \\- A \- omfang_OK(Par).
omfang_OK <=
        d_right(_,v4_right(_,omfang_OK3,omfang_OK3,omfang_OK3,omfang_OK3)).

% omfang_OK3 \\- A \- omfang_OK(S,T,P).
omfang_OK3 <=
        d_right(_,v3_right(_,tillatet_omfang,under,under)).

% stamavstand_OK \\- A \- stamavstand_OK(Par).
stamavstand_OK <=
        d_right(_,v_right(_,stamavst_OK,stamavst_OK)).

% stamavst_OK \\- A \- stamavst_OK(S1,S2,T1,T2,P).
stamavst_OK <=
        d_right(_,v3_right(_,intervall,tillatna_avstand,member)).

% ej_medrorelse \\- A \- ej_medrorelse(Par1,Par2).
ej_medrorelse <=
        d_right(_,v9_right(_,ton(_),ton(_),ton(_),ton(_),ton(_),ton(_),ton(_),
         ton(_),a_right(_,d_left(_,_,ej_medrorelse1)))).

% Hjalpstrategi for ej_medrorelse (med resp utan undantag):
        ej_medrorelse1 <=
                par_move_left(_,false_left(_)),
                false_left(_).

% inga_oktpar \\- A \- inga_oktpar(Par1,Par2).
inga_oktpar <=
        d_right(_,v9_right(_,ton(_),ton(_),ton(_),ton(_),ton(_),ton(_),ton(_),
         ton(_),a_right(_,d_left(_,_,inga_oktpar1)))).

% Hjalpstrategi for inga_oktpar (med resp utan undantag):
        inga_oktpar1 <=
                par_eights_left(_,false_left(_)),
                false_left(_).

% inga_kvintpar \\- A \- inga_kvintpar(Par1,Par2).
inga_kvintpar <=
        d_right(_,v9_right(_,ton(_),ton(_),ton(_),ton(_),ton(_),ton(_),ton(_),
         ton(_),a_right(_,d_left(_,_,inga_kvintpar1)))).

% Hjalpstrategi for inga_oktpar (med resp utan undantag):
        inga_kvintpar1 <=
                par_fifths_left(_,false_left(_)),
                false_left(_).

% intervall_OK \\- A \- intervall_OK(FKNS,Par1,Par2).
intervall_OK <=
        d_right(_,v4_right(_,godkant_intervall,godkant_intervall,
         godkant_intervall,godkant_intervall)).

% godkant_intervall \\- A \- godkant_intervall(S,Par1,Par2,FKNS).
godkant_intervall <=
        d_right(_,v4_right(_,under,intervall,tillatna_intervall,member)).
```

```
% Harledbart om inga kvintparalleller forekommer mellan ackorden X och Y
% kp ar en sa kallad konstruerare.
kvint_paraller(X,Y) <=
        kp(X,Y).

% Harledbart om forflyttningarna for respektive stamma mellan ackorden
% [B,T,A,S] och [B1,T1,A1,S1] ar tillatna med hansyn till givna funktioner FKNS.
intervall_OK(FKNS,par([B,T,A,S],Pos),par([B1,T1,A1,S1],Pos1)) <=
        godkant_intervall(sopran,par(S,Pos),par(S1,Pos1),FKNS),
        godkant_intervall(alt,par(A,Pos),par(A1,Pos1),FKNS),
        godkant_intervall(tenor,par(T,Pos),par(T1,Pos1),FKNS),
        godkant_intervall(bas,par(B,Pos),par(B1,Pos1),FKNS).

% Harledbart om forflyttningen fran tonen T1 till tonen T2 ar
% tillaten for stamman S med hansyn till givna funktioner FKNS.
godkant_intervall(S,par(T1,_),par(T2,_),FKNS) <=
        under(T1,T2),
        intervall(T1,T2,A,B),
        tillatna_intervall(S,FKNS,Intervall),
        member([A,B],Intervall).
godkant_intervall(S,par(T1,_),par(T2,_),FKNS) <=
        str_over(T1,T2),
        intervall(T2,T1,A,B),
        tillatna_intervall(S,FKNS,Intervall),
        member([A,B],Intervall).
```

# stamforing.rul

```
% Nedanstaende filer behovs:
% :- include_rules('bas.rul').
% :- include_rules('bygg.rul').
% :- include_rules('grundregler.rul').
% :- include_rules('kbs.rul').
% :- include_rules('ton.rul').

                %%%%%%%%%%%%%%
                % Strategier %
                %%%%%%%%%%%%%%

% stamforing_OK \\- A \- stamforing_OK(Ta,FKNS,L1,L2).
stamforing_OK <=
        d_right(_,v9_right(_,und_pil_false,spec_sf_krav,ingen_stamkorsning,
         omfang_OK,stamavstand_OK,ej_medrorelse,inga_oktpar,inga_kvintpar,
         intervall_OK)).

% spec_sf_krav \\- A \- spec_sf_krav(Ta,Par1,Par2,Par3).
spec_sf_krav <=
        d_right(_,spec_sf_krav(_)).

% Hjalpstrategi for spec_sf_krav/0:
        spec_sf_krav(C) <=
                (_ \- C).
        spec_sf_krav(((ackordtoner(_,_) -> [_,_,_]),identifiera_ackordton(_,_,_,_),
                        stamma(_,_,_,_),laga_tonen(_,_,_,_))) <=
                v4_right(_,a_right(_,ackordtoner),identifiera_ackordton,dt_right(_),
                 hoga_tonen).
        spec_sf_krav((member(_,_),(ackordtoner(_,_) -> [_,_,_,_]),(ackordtoner(_,_) -> _),
                        identifiera_ackordton(_,_,_,_),stamma(_,_,_,_),
                        identifiera_ackordton(_,_,_,_))) <=
                v6_right(_,member,a_right(_,ackordtoner),a_right(_,ackordtoner),
                 identifiera_ackordton,dt_right(_),identifiera_ackordton).
        spec_sf_krav((spec_krav(_,_) -> false)) <=
                a_right(_,d_left(_,_,false_left(_))).
```

```
% Harledbart om tonerna i ackordet [B,T,A,S] ligger inom givna omfang
% for respektive stammor.
omfang_OK(par([B,T,A,S],Pos)) <=
        omfang_OK(sopran,S,Pos),
        omfang_OK(alt,A,Pos),
        omfang_OK(tenor,T,Pos),
        omfang_OK(bas,B,Pos).

% Harledbart om tonen Ton ligger inom den givna stammans S omfang.
omfang_OK(S,Ton,_) <=
        tillatet_omfang(S,Lag,Hog),
        under(Lag,Ton),
        under(Ton,Hog).

% Harledbart om tonerna i ackordet [B,T,A,S] ar placerade med tillatna
% intervall sinsemellan. (Basstamman ar i detta avseende fri.)
stamavstand_OK(par([B,T,A,S],Pos)) <=
        stamavst_OK(alt,sopran,A,S,Pos),
        stamavst_OK(tenor,alt,T,A,Pos).

% Harledbart om intervallet mellan Lag och Hog ar tillatet.
stamavst_OK(_,_,Lag,Hog,_) <=
        intervall(Lag,Hog,Ityp,Intervall),
        tillatna_avstand(I_lista),
        member(Intervall,I_lista).

% Harledbart om medrorelse inte foreligger mellan ackorden [B,T,A,S] och [B1,T1,A1,S1].
ej_medrorelse(par([B,T,A,S],_),par([B1,T1,A1,S1],_)) <=
        ton(B,Bn), ton(T,Tn),
        ton(A,An), ton(S,Sn),
        ton(B1,B1n), ton(T1,T1n),
        ton(A1,A1n), ton(S1,S1n),
        (medror([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) -> false).

% Harledbart om medrorelse foreligger mellan ackorden X och Y
% mr ar en sa kallad konstruerare.
medror(X,Y) <=
        mr(X,Y).

% Harledbart om inga oktavparalleller forekommer mellan ackorden
% [B,T,A,S] och [B1,T1,A1,S1].
inga_oktpar(par([B,T,A,S],_),par([B1,T1,A1,S1],_)) <=
        ton(B,Bn),ton(T,Tn),
        ton(A,An),ton(S,Sn),
        ton(B1,B1n),ton(T1,T1n),
        ton(A1,A1n),ton(S1,S1n),
        (oktav_paraller([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) -> false).

% Harledbart om oktavparalleller forekommer mellan ackorden X och Y
% okp ar en sa kallad konstruerare.
oktav_paraller(X,Y) <=
        okp(X,Y).

% Harledbart om kvintparalleller forekommer mellan ackorden
% [B,T,A,S] och [B1,T1,A1,S1].
inga_kvintpar(par([B,T,A,S],_),par([B1,T1,A1,S1],_)) <=
        ton(B,Bn),ton(T,Tn),
        ton(A,An),ton(S,Sn),
        ton(B1,B1n),ton(T1,T1n),
        ton(A1,A1n),ton(S1,S1n),
        (kvint_paraller([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) -> false).
```

```
% Definition som anvands utat.
%
% stamforing_OK \\- A \- stamforing_OK(Ta,FKNS,L1,L2).
%
% A star for godtycklig antecedent. Ingen hansyn tas till denna.


:- dynamic_term(medror/2).
:- dynamic_term(oktav_paraller/2).
:- dynamic_term(kvint_paraller/2).


% Harledbart om alla stamforingsregler ar uppfyllda for ackorden X och Y
% i given Tonart, med hansyn till givna funktioner FKNS.
stamforing_OK(Tonart,FKNS,X,par(Ack,Pos)) <=
        (undantag(ackord(Ack),Pos) -> false),
        spec_sf_krav(Tonart,FKNS,X,par(Ack,Pos)),
        ingen_stamkorsning(par(Ack,Pos)),
        omfang_OK(par(Ack,Pos)),
        stamavstand_OK(par(Ack,Pos)),
        ej_medrorelse(X,par(Ack,Pos)),
        inga_oktpar(X,par(Ack,Pos)),
        inga_kvintpar(X,par(Ack,Pos)),
        intervall_OK(FKNS,X,par(Ack,Pos)).

% Harledbart om vissa speciella eller anvandardefinierade stamforingskrav ar uppfyllda.
spec_sf_krav(Tonart,par(fkn(fD,4,1),fkn(fD,Typ,Bt)),
                par([B,T,A,S],Pos),par([B1,T1,A1,S1],_)) <=
        (ackordtoner(Tonart,fkn(fD,4,1)) -> [Grt,Kvrt,Kvnt]),
        identifiera_ackordton(Kvrt,[B,T,A,S],Kvartstamma,Pos),
        stamma(Kvartstamma,[B,T,A,S],[B1,T1,A1,S1],Tersstamma),
        laga_tonen(Kvartstamma,liten,sekund,Tersstamma).
spec_sf_krav(Tonart,par(fkn(fS,S65,Sb),fkn(fD,Typ,Bt)),
                par([B,T,A,S],Pos),par([B1,T1,A1,S1],Pos1)) <=
        member(S65,[65,m65]),
        (ackordtoner(Tonart,fkn(fS,S65,Sb)) -> [Grt,Trs,Kvnt,Sxt]),
        (ackordtoner(Tonart,fkn(fD,Typ,Bt)) -> [Grt1,Trs1|Xs]),
        identifiera_ackordton(Sxt,[B,T,A,S],Sxtstamma,Pos),
        stamma(Sxtstamma,[B,T,A,S],[B1,T1,A1,S1],Till_ton),
        identifiera_ackordton(Till_ton,[Grt1|Xs],Ackton,Pos1).              % Inte tersen
spec_sf_krav(_,par(fkn(F1,T1,B1),fkn(F2,T2,B2)),par([_,_,_,_],_),par([_,_,_,_],_)) <=
        (spec_krav(fkn(F1,T1,B1),fkn(F2,T2,B2)) -> false).

% Definition av speciella stamforingskrav.
spec_krav(fkn(fD,4,1),fkn(fD,Typ,Bt)).
spec_krav(fkn(fS,65,Sb),fkn(fD,Typ,Bt)).
spec_krav(fkn(fS,m65,Sb),fkn(fD,Typ,Bt)).

% Korresponderande toner (toner i samma stamma).
stamma(X,[X,_,_,_],[Y,_,_,_],Y).
stamma(X,[_,X,_,_],[_,Y,_,_],Y).
stamma(X,[_,_,X,_],[_,_,Y,_],Y).
stamma(X,[_,_,_,X],[_,_,_,Y],Y).

% Harledbart om ingen stamkorsning forekommer i givet ackord [B,T,A,S].
ingen_stamkorsning(par([B,T,A,S],_)) <=
        under(B,T),
        under(T,A),
        under(A,S).

% Harledbart om ingen stamkorsning forekommer mellan tonerna (T,A,S).
ingen_stamkorsning(T,A,S) <=
        under(T,A),
        under(A,S).
```

# skala.rul

```
% Nedanstaende filer behovs:
% :- include_rules('grundregler.rul').
% :- include_rules('ton.rul').


                %%%%%%%%%%%%%
                % Strategier %
                %%%%%%%%%%%%%

% skala \\- A,skala(T,St) \- L.
skala <=
        d_left(_,_,a_left(_,_,dt_right(_),iskala)).

% iskala \\- A,iskala(L1,T) \- L2.
iskala <=
        d_left(_,_,a1_left(_,_,v7_right(_,hoga_tonen,hoga_tonen,
         hoga_tonen,hoga_tonen,hoga_tonen,hoga_tonen,hoga_tonen),
         axiom(_,_,_))).

% Da den melodiska mollskalan brukar
% betecknas med bade upp- och nedgang,
% fungerar inte strategin skala pa denna.
% mskala \\- A,skala(T,mel_moll) \- L.
mskala <=
        d_left(_,_,a_left(_,_,v4_right(_,dt_right(_),dt_right(_),iskala,
         iskala),axiom(_,_,_))).

% r_skala \\- A,r_skala(T,St) \- L.
r_skala <=
        d_left(_,_,a_left(_,_,v_right(_,dt_right(_),a_right(_,r_iskala)),
         axiom(_,_,_))).

% r_iskala \\- A,r_iskala(L1,T) \- L2.
r_iskala <=
        d_left(_,_,a_left(_,_,v_right(_,hoga_tonen,a_right(_,r_iskala)),
         axiom(_,_,_))),
        d_left(_,_,axiom(_,_,_)).


                %%%%%%%%%%%%%%%%%%%%%%
                % Kontrollstrategier %
                %%%%%%%%%%%%%%%%%%%%%%

% skala - kontroll:
skalaK <=
        d_left(_,_,a_left(_,_,dt_right(_),iskalaK)).

iskalaK <=
        d_left(_,_,a1_left(_,_,v7_right(_,hoga_tonenK,hoga_tonenK,
         hoga_tonenK,hoga_tonenK,hoga_tonenK,hoga_tonenK,hoga_tonenK),
         axiom(_,_,_))).
```

# stamforing.def

```
% Nedanstaende filer behovs:
% :- include_def('bas.def').
% :- include_def('bygg.def').
% :- include_def('kbs.def').
% :- include_def('ton.def').

% Strategierna finns i filen 'stamforing.rul'.
```

```
% Givet en lista av intervalltyper [Ityp1,Ityp2,...] samt en grundton Grt
% ges som resultat en skala med given grundton, dar intervalltyperna
% bestams av den givna listan.
iskala([Ityp1,Ityp2,Ityp3,Ityp5,Ityp6],Grt) <=
        (hoga_tonen(Grt,Ityp1,sekund,Ton2),
        hoga_tonen(Grt,Ityp2,ters,Ton3),
        hoga_tonen(Grt,Ityp3,kvart,Ton4),
        hoga_tonen(Grt,ren,kvint,Ton5),
        hoga_tonen(Grt,Ityp5,sext,Ton6),
        hoga_tonen(Grt,Ityp6,septim,Ton7),
        hoga_tonen(Grt,ren,oktav,Ton8))
        -> [Grt,Ton2,Ton3,Ton4,Ton5,Ton6,Ton7,Ton8].

% For varje typ av skala anges har intervalltyperna for
% [sekund,ters,kvart,sext,septim]. Alla skaltyper har
% ren prim, kvint och oktav, varfor dessa intervalltyper
% inte behover anges.
ityper(dur,[stor,stor,ren,stor,stor]).
ityper(ren_moll,[stor,liten,ren,liten,liten]).
ityper(harm_moll,[stor,liten,ren,liten,stor]).
ityper(dorisk,[stor,liten,ren,stor,liten]).
ityper(frygisk,[liten,liten,ren,liten,liten]).
ityper(lydisk,[stor,stor,overstigande,stor,stor]).
ityper(mixolydisk,[stor,stor,ren,stor,liten]).
ityper(mel_moll,[stor,liten,ren,stor,stor]).


% Rekursiv definition av skala. Fungerar som den ovanstaende, med skillnaden att
% den melodiska mollskalan tas omhand av samma definition som ovriga skalor.
% Utbyggbar for godtyckliga langder av skalor.
r_skala(Grundton,Skaltyp) <=
        r_intervall(Skaltyp,Inter),
        (r_iskala(Inter,Grundton) -> Skala)
        -> [Grundton|Skala].

% Rekursiv variant av iskala.
r_iskala([],_) <=
        [].
r_iskala([Ityp,Int|Inter],Grt) <=
        hoga_tonen(Grt,Ityp,Int,Ton),
        (r_iskala(Inter,Grt) -> Restskala)
        -> [Ton|Restskala].

% Har anges alla intervalltyper och intervall, alltsa aven kvint och oktav.
r_intervall(dur,[stor,sekund,stor,ters,ren,kvart,ren,kvint,
                stor,sext,stor,septim,ren,oktav]).
r_intervall(ren_moll,[stor,sekund,liten,ters,ren,kvart,ren,kvint,
                liten,sext,liten,septim,ren,oktav]).
r_intervall(harm_moll,[stor,sekund,liten,ters,ren,kvart,ren,kvint,
                liten,sext,stor,septim,ren,oktav]).
r_intervall(dorisk,[stor,sekund,liten,ters,ren,kvart,ren,kvint,
                stor,sext,liten,septim,ren,oktav]).
r_intervall(frygisk,[liten,sekund,liten,ters,ren,kvart,ren,kvint,
                liten,sext,liten,septim,ren,oktav]).
r_intervall(lydisk,[stor,sekund,stor,ters,overstigande,kvart,ren,kvint,
                stor,sext,stor,septim,ren,oktav]).
r_intervall(mixolydisk,[stor,sekund,stor,ters,ren,kvart,ren,kvint,
                stor,sext,liten,septim,ren,oktav]).
r_intervall(mel_moll,[stor,sekund,liten,ters,ren,kvart,ren,kvint,
                stor,sext,stor,septim,ren,oktav,liten,septim,liten,sext,
                ren,kvint,ren,kvart,liten,ters,stor,sekund,ren,prim]).
```

```
                %%%%%%%%%%%%%%%%%%%%%%%%
                % Kontrollstrategier %
                %%%%%%%%%%%%%%%%%%%%%%%%

% fordubbling - kontroll:
fordubblingK <=
        fordubbling(dubblering,identifiera_ackordton),
        dubbleringE,
        skip.

% baston - kontroll (lyckas alltid):
bastonK <=
        skip.

% undantag - kontroll (lyckas alltid):
und_pil_falseK <=
        skip.


                %%%%%%%%%%%%%%%%%%%%%%%%%%
                % Regel for felutskrift %
                %%%%%%%%%%%%%%%%%%%%%%%%%%

% dubblering - felutskrift:
dubbleringE <=
        (((ton(_) -> (_ \- ton(S,_))),
        (ton(_) -> (_ \- ton(A,_))),
        (ton(_) -> (_ \- ton(T,_))),
        (ton(_) -> (_ \- ton(B,_))),
        format('Otillaten fordubbling i ackord nr ~w.~n',[N]))
        -> (_ \- fordubbling(_,par(_,S),_,_,par([B,T,A],pos(N)),_))).
```

# skala.def

```
% Nedanstaende fil behovs:
% :- include_def('intervall.def').

% Strategierna finns i filen 'skala.rul'.

% Definitioner som anvands utat:
%
% skala \\- A,skala(T,St) \- L.
% mskala \\- A,skala(T,mel_moll) \- L.
% r_skala \\- A,r_skala(T,St) \- L.
%
% A star for godtycklig antecedent. Ingen hansyn tas till denna.


% Givet skalans Grundton och typen av skala, Skaltyp,
% ges som resultat en lista av de ingaende tonerna i angiven skala.
%        (Den melodiska mollskalan, som ar olika i uppat- och
%        nedatgaende, kraver en sarskild berakningsmetod.)
% Kan med betydande effektivitetsforlust anvandas aven omvant,
% dvs man kan ange vissa toner i skalan och fa fram Grundton och Skaltyp.
skala(Grundton,Skaltyp)#{Skaltyp \= mel_moll} <=
        ityper(Skaltyp,Ityper)
        -> iskala(Ityper,Grundton).
skala(Grundton,mel_moll) <=
        ityper(mel_moll,Ityper1),
        ityper(ren_moll,Ityper2),
        (iskala(Ityper1,Grundton) -> [U1,U2,U3,U4,U5,U6,U7,U8]),
        (iskala(Ityper2,Grundton) -> [N1,N2,N3,N4,N5,N6,N7,N8])
        -> [U1,U2,U3,U4,U5,U6,U7,U8,N7,N6,N5,N4,N3,N2,N1].
```

```
% tillatna_avstand \\- A \- tillatna_avstand(L).
tillatna_avstand <=
        dt_right(_).

% dubb_krav \\- A \- dubb_krav(Ta,Par1,Par2,Par3,X).
dubb_krav <=
        d_right(_,dubb_krav(_)).

% Hjalpstrategi for dubb_krav/0:
        dubb_krav(C) <=
                (_ \- C).
        dubb_krav(true) <=
                true_right.
        dubb_krav((till_fkn(_,_),member(_,_),hoga_tonen(_,_,_,_),hoga_tonen(_,_,_,_),
                        hoga_tonen(_,_,_,_),identifiera_ackordton(_,_,_,_),
                        (identifiera_ackordton(_,_,_,_);
                        identifiera_ackordton(_,_,_,_)))) <=
                v7_right(_,dt_right(_),member,hoga_tonen,hoga_tonen,hoga_tonen,
                 identifiera_ackordton,o_right(_,_,identifiera_ackordton)).

% fordubbling \\- A \- fordubbling(FKN,Par,L,Val,P,X).
fordubbling(Dubblering,Identifiera_ackordton) <=
        d_right(_,v4_right(_,Dubblering,Identifiera_ackordton,Identifiera_ackordton,
         Identifiera_ackordton)).

% fordubbling - simulering:
fordubblingS <=
        fordubbling(dubblering,skip).

% dubblering \\- A \- dubblering(FKN,Par,L,Val,P,X).
dubblering <=
        d_right(_,v_right(_,und_pil_false,dubblering(_))).

% Hjalpstrategi for dubblering/0:
        dubblering(C) <=
                (_ \- C).
        dubblering(member(_,_)) <=                                          % nr 1
                member.
        dubblering((member(_,_),identifiera_ackordton(_,_,_,_))) <=             % nr 2-6
                v_right(_,member,identifiera_ackordton).
        dubblering((member(_,_),identifiera_ackordton(_,_,_,_),
                        (removefirst(_,_) -> _))) <=                      % nr 7-8, 16-24
                v3_right(_,member,identifiera_ackordton,a_right(_,removefirst)).
        dubblering((identifiera_ackordton(_,_,_,_),(removefirst(_,_) -> _))) <= % nr 9-15
                v_right(_,identifiera_ackordton,a_right(_,removefirst)).
        dubblering(basval(_,_,_)) <=                                        % nr 25
                dt_right(_).

% und_pil_false \\- A \- undantag(X) -> false.
% und_pil_false \\- A \- undantag(X,Y) -> false.
und_pil_false <=
        a_right(_,d_left(_,_,false_left(_))).


                %%%%%%%%%%%%%%%%%%
                % Analysstrategi %
                %%%%%%%%%%%%%%%%%%

dubb_kravA <=
        skip.
```

# kbs.rul

```
% Nedanstaende filer behovs:
% :- include_rules('bas.rul').
% :- include_rules('bygg.rul').
% :- include_rules('grundregler.rul').
% :- include_rules('ton.rul').

                %%%%%%%%%%%%%%
                % Strategier %
                %%%%%%%%%%%%%%

% till_fkn \\- A \- till_fkn(F,FKN).
till_fkn <=
        dt_right(_).

% intervall_tonart_funktions_grundton \\- A \-intervall_tonart_funktions_grundton(F,It,I).
intervall_tonart_funktions_grundton <=
        dt_right(_).

% baston \\- A \- baston(FKN1,FKN2,T1,T2,T3,T4).
baston <=
        d_right(_,baston(_)).

% Hjalpstrategi for baston/0:
        baston(C) <=
                (_\- C).
        baston((laga_tonen(_,_,_,_),kanske_upp(_,_),under(_,_))) <=
                v3_right(_,hoga_tonen,kanske,under).
        baston((hoga_tonen(_,_,_,_),kanske_ner(_,_),under(_,_))) <=
                v3_right(_,hoga_tonen,kanske,under).
        baston((hoga_tonen(_,_,_,_),under(_,_))) <=
                v_right(_,hoga_tonen,under).
        baston(hoga_tonen(_,_,_,_)) <=
                hoga_tonen.
        baston(laga_tonen(_,_,_,_)) <=
                hoga_tonen.
        baston(((undantag(bas(_,_))-> false),lamplig_oktav(_,_,_))) <=
                v_right(_,und_pil_false,lamplig_oktav).

% kanske \\- A \- kanske_upp(T1,T2).
% kanske \\- A \- kanske_ner(T1,T2).
kanske <=
        dt_right(_),
        d_right(_,hoga_tonen).

% tillatet_omfang \\- A \- tillatet_omfang(S,T1,T2).
tillatet_omfang <=
        dt_right(_).

% tillatna_intervall \\- A \- tillatna_intervall(S,FKNS,L).
tillatna_intervall <=
        d_right(_,tillatna_intervall(_)).

% Hjalpstrategi for tillatna_intervall/0:
        tillatna_intervall(C) <=
                (_\- C).
        tillatna_intervall(((undantag(intervall(_)) -> false),member(_,_))) <=
                v_right(_,und_pil_false,member).
        tillatna_intervall((undantag(intervall(_)) -> false)) <=
                und_pil_false.
        tillatna_intervall(member(_,_)) <=
                member.
        tillatna_intervall(true) <=             % Anvands om alla intervall ar tillatna.
                true_right.
```

```
% Allmant tretons-ackord, utan kvint
dubblering(FKN,par(fkn(F,DM,1),S1),[X1,X2,X3],val(bas(X1),Kvar),_,18) <=
        (undantag(utan_kvint,fkn(F,DM,1)) -> false),                % tillat bara tonika
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[X1,X2],S1_ackton,_),
        (removefirst(S1_ackton,[X1,X1,X2]) -> Kvar).
% Allmant tretons-ackord, dubbel ters
dubblering(FKN,par(fkn(F,DM,1),S1),[X1,X2,X3],val(bas(X1),Kvar),_,19)
                #{FKN \= fkn(fD,4,1)} <=
        (undantag(dubbel_ters,fkn(F,DM,1)) -> false),
        member(DM,[dur,moll,6,m6]),
        identifiera_ackordton(S1,[X2,X3],S1_ackton,_),
        (removefirst(S1_ackton,[X2,X2,X3]) -> Kvar).
% Allmant tretons-ackord, dubbel sext
dubblering(FKN,par(fkn(F,DM,1),S1),[X1,X2,X3],val(bas(X1),Kvar),_,20) <=
        (undantag(dubbel_sext,fkn(F,DM,1)) -> false),
        member(DM,[6,m6,d64]),
        identifiera_ackordton(S1,[X2,X3],S1_ackton,_),
        (removefirst(S1_ackton,[X3,X2,X3]) -> Kvar).
% Allmant tretons-ackord, dubbel kvint
dubblering(FKN,par(fkn(F,DM,1),S1),[X1,X2,X3],val(bas(X1),Kvar),_,21) <=
        (undantag(dubbel_kvint,fkn(F,DM,1)) -> false),
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[X2,X3],S1_ackton,_),
        (removefirst(S1_ackton,[X3,X2,X3]) -> Kvar).
% Kvartsext-ackord, dubbel baston
dubblering(FKN,par(fkn(F,DM,5),S1),[X1,X2,X3],val(bas(X3),Kvar),_,22) <=
        (undantag(dubbel_baston,fkn(F,DM,5)) -> false),
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[X1,X2,X3],S1_ackton,_),
        (removefirst(S1_ackton,[X1,X2,X3]) -> Kvar).
% Kvartsext-ackord, dubbel grundton
dubblering(FKN,par(fkn(F,DM,5),S1),[X1,X2,X3],val(bas(X3),Kvar),_,23) <=
        (undantag(dubbel_grundton,fkn(F,DM,5)) -> false),
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[X1,X2],S1_ackton,_),
        (removefirst(S1_ackton,[X1,X1,X2]) -> Kvar).
% Kvartsext-ackord, dubbel ters
dubblering(FKN,par(fkn(F,DM,5),S1),[X1,X2,X3],val(bas(X1),Kvar),_,24)
                #{FKN \= fkn(fD,4,1)} <=
        (undantag(dubbel_ters,fkn(F,DM,5)) -> false),
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[X1,X2],S1_ackton,_),
        (removefirst(S1_ackton,[X1,X2,X2]) -> Kvar).
% Fri dubblering
dubblering(FKN,par(fkn(F,DM,Bt),S1),Acks,val(bas(B),Acks),_,25) <=
        (undantag(fri_dubblering,fkn(F,DM,Bt)) -> false),
        basval(Bt,Acks,B).

% Val av baston vid fri dubblering.
basval(1,[X1,X2,X3],X1).
basval(3,[X1,X2,X3],X2).
basval(5,[X1,X2,X3],X3).
basval(1,[X1,X2,X3,X4],X1).
basval(3,[X1,X2,X3,X4],X2).
basval(5,[X1,X2,X3,X4],X3).
basval(6,[X1,X2,X3,X4],X4).
basval(7,[X1,X2,X3,X4],X4).
basval(s,[X1,X2,X3,X4],X3).
```

```
% Sextackord, dubbel ters 2 (Soderholm sid 35 - 45).
dubblering(FKN,par(fkn(F,DM,3),S1),[Grt,Trs,Kvnt],val(bas(Trs),[Grt,Trs]),_,6) <=
        (undantag(dubbel_ters,fkn(F,DM,3)) -> false),
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[Kvnt],Kvnt,_).
% Ofullkomlig dominant, dubbel baston (Soderholm sid 52 - 53).
dubblering(FKN,par(fkn(F,7,s),S1),[Grt,Trs,Kvnt,Spt],val(bas(Kvnt),Kvar),_,7)<=
        (undantag(dubbel_baston,fkn(F,7,s)) -> false),
        member(F,[fD,fS]),
        identifiera_ackordton(S1,[Kvnt,Trs,Spt],S1_ackton,_),
        (removefirst(S1_ackton,[Kvnt,Trs,Spt]) -> Kvar).
% Ofullkomlig dominant, dubbel septim (Soderholm sid 52 - 53).
dubblering(FKN,par(fkn(F,7,s),S1),[Grt,Trs,Kvnt,Spt],val(bas(Kvnt),Kvar),_,8)<=
        (undantag(dubbel_septim,fkn(F,7,s)) -> false),
        member(F,[fD,fS]),
        identifiera_ackordton(S1,[Trs,Spt],S1_ackton,_),
        (removefirst(S1_ackton,[Spt,Trs,Spt]) -> Kvar).
% D - D7
dubblering(fkn(fD,_,1),par(fkn(fD,7,1),S1),[Grt,Trs,Kvnt,Spt],val(bas(Grt),Kvar),_,9) <=
        (undantag(odubblerat,fkn(fD,7,1)) -> false),
        identifiera_ackordton(S1,[Trs,Kvnt,Spt],S1_ackton,_),
        (removefirst(S1_ackton,[Spt,Trs,Kvnt]) -> Kvar).
% T - D7 och S - D7 utan kvint i D7
dubblering(fkn(_,_,1),par(fkn(fD,7,1),S1),[Grt,Trs,Kvnt,Spt],val(bas(Grt),Kvar),_,10) <=
        (undantag(utan_kvint,fkn(fD,7,1)) -> false),
        identifiera_ackordton(S1,[Grt,Trs,Spt],S1_ackton,_),
        (removefirst(S1_ackton,[Grt,Trs,Spt]) -> Kvar).
% Allmant fyratons-ackord, grundton i basen
dubblering(FKN,par(fkn(F,Typ,1),S1),[X1,X2,X3,X4],val(bas(X1),Kvar),_,11) <=
        (undantag(odubblerat,fkn(F,Typ,1)) -> false),
        identifiera_ackordton(S1,[X2,X3,X4],S1_ackton,_),
        (removefirst(S1_ackton,[X2,X3,X4]) -> Kvar).
% Allmant fyratons-ackord, ters i basen
dubblering(FKN,par(fkn(F,Typ,3),S1),[X1,X2,X3,X4],val(bas(X2),Kvar),_,12) <=
        (undantag(odubblerat,fkn(F,Typ,3)) -> false),
        identifiera_ackordton(S1,[X1,X3,X4],S1_ackton,_),
        (removefirst(S1_ackton,[X1,X3,X4]) -> Kvar).
% Allmant fyratons-ackord, kvint i basen
dubblering(FKN,par(fkn(F,Typ,5),S1),[X1,X2,X3,X4],val(bas(X3),Kvar),_,13) <=
        (undantag(odubblerat,fkn(F,Typ,5)) -> false),
        identifiera_ackordton(S1,[X1,X2,X4],S1_ackton,_),
        (removefirst(S1_ackton,[X1,X2,X4]) -> Kvar).
% Allmant fyratons-ackord, sext i basen
dubblering(FKN,par(fkn(F,Typ,6),S1),[X1,X2,X3,X4],val(bas(X4),Kvar),_,14) <=
        (undantag(odubblerat,fkn(F,Typ,6)) -> false),
        identifiera_ackordton(S1,[X1,X3,X2],S1_ackton,_),
        (removefirst(S1_ackton,[X1,X3,X2]) -> Kvar).
% Allmant fyratons-ackord, septim i basen
dubblering(FKN,par(fkn(F,Typ,7),S1),[X1,X2,X3,X4],val(bas(X4),Kvar),_,15) <=
        (undantag(odubblerat,fkn(F,Typ,7)) -> false),
        identifiera_ackordton(S1,[X1,X3,X2],S1_ackton,_),
        (removefirst(S1_ackton,[X1,X3,X2]) -> Kvar).
% S65 utan ters, dubbel grundton och grundton i basen
dubblering(FKN,par(fkn(fS,Typ,1),S1),[G,T,K,S],val(bas(G),Kvar),_,16) <=
        (undantag(dubbel_grundton,fkn(fS,Typ,1)) -> false),
        member(Typ,[65,m65]),
        identifiera_ackordton(S1,[G,K,S],S1_ackton,_),
        (removefirst(S1_ackton,[G,K,S])-> Kvar).
% Allmant tretons-ackord, dubbel grundton
dubblering(FKN,par(fkn(F,DM,1),S1),[X1,X2,X3],val(bas(X1),Kvar),_,17) <=
        (undantag(dubbel_grundton,fkn(F,DM,1)) -> false),
        member(DM,[dur,moll,6,m6,d64,md64,4]),
        identifiera_ackordton(S1,[X1,X2,X3],S1_ackton,_),
        (removefirst(S1_ackton,[X1,X2,X3]) -> Kvar).
```

```
% f_T6, f_D6 - bara grundtonsfordubbling
undantag(dubbel_sext,fkn(fT,6,1)).
undantag(dubbel_sext,fkn(fT,m6,1)).
undantag(dubbel_sext,fkn(fD,6,1)).
undantag(dubbel_sext,fkn(fD,m6,1)).
undantag(dubbel_ters,fkn(fT,6,1)).
undantag(dubbel_ters,fkn(fT,m6,1)).
undantag(dubbel_ters,fkn(fD,6,1)).
undantag(dubbel_ters,fkn(fD,m6,1)).


% Vid forbindelsen S3 - D vill man ibland i forsta hand dubblera kvinten.
% Har kontrolleras om villkoren for detta ar uppfyllda.
% I sa fall instansieras sista argumentet till 3, vilket ar numret pa
% en dubblerings-regel (se dubblering/6). I annat fall sker ingenting.
dubb_krav(Tonart,par(_,_),par(fkn(fS,DM,3),[_,_,_,S]),par(F3,[_,_,_,S1]),4) <=
        till_fkn(F3,fkn(fD,T,B)),
        member(DM,[dur,moll]),
        hoga_tonen(Tonart,ren,kvart,Grt_i_S),
        hoga_tonen(Tonart,ren,kvint,Grt_i_D),
        hoga_tonen(Tonart,stor,septim,Trs_i_D),
        identifiera_ackordton(S,[Grt_i_S],X,_),
        (identifiera_ackordton(S1,[Grt_i_D],Y,_);
        identifiera_ackordton(S1,[Trs_i_D],Y,_)).
dubb_krav(Tonart,par(_,_),par(_,_),par(_,_),_).


% fordubbling(F1,par(F2,Meloditon),Ackordtoner,val(bas(Baston,Alt_Tenor)),Pos,Nr).
% Valjer vilken av ett ackords toner (Ackordtoner) som skall fordubblas.
% Givet Meloditon och Ackordtoner sa ges i Baston respektive Alt_Tenor
% vilka toner som finns att valja pa for basen, alten och tenoren.
% For manga kombinationer finns flera tillampbara dubblerings-regler som ges vid
% backtracking.
fordubbling(FKN,par(F,S1),Acktoner,val(bas(B1),Kvar),par([B,T,A],pos(N)),R) <=
        dubblering(FKN,par(F,S1),Acktoner,val(bas(B1),Kvar),par([B,T,A],pos(N)),R),
        identifiera_ackordton(B,[B1],_,_),                                  % Utfores
        identifiera_ackordton(T,Kvar,At,_),                               % endast vid
        identifiera_ackordton(A,removefirst(At,Kvar),_,_).               % satskontroll

% Hjalp till fordubbling
% Sextackord, dubbel meloditon (Soderholm sid 35 - 45).
dubblering(FKN,par(fkn(F,DM,3),S1),[Grt,Trs,Kvnt],val(bas(Trs),[Grt,Kvnt]),_,1) <=
        (undantag(dubbel_meloditon,fkn(F,DM,3)) -> false),
        member(DM,[dur,moll]).
% f_D3 (Soderholm sid 35 - 45).
dubblering(FKN,par(fkn(fD,DM,3),S1),[Grt,Trs,Kvnt],val(bas(Trs),[Grt,Kvnt]),_,2) <=
        (undantag(dubbel_melodiD3,fkn(F,DM,3)) -> false),
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[Grt,Kvnt],S1_ackton,_).
% Sextackord, dubbel grundton (Soderholm sid 35 - 45).
dubblering(FKN,par(fkn(_,DM,3),S1),[Grt,Trs,Kvnt],val(bas(Trs),[Grt,Grt]),_,3) <=
        (undantag(dubbel_grundton,fkn(F,DM,3)) -> false),
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[Grt,Trs,Kvnt],Kvnt,_).
% Sextackord, dubbel kvint (Soderholm sid 35 - 45).
dubblering(FKN,par(fkn(_,DM,3),S1),[Grt,Trs,Kvnt],val(bas(Trs),[Kvnt,Kvnt]),_,4) <=
        (undantag(dubbel_kvint,fkn(F,DM,3)) -> false),
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[Grt,Trs,Kvnt],Grt,_).
% Sextackord, dubbel ters 1 (Soderholm sid 35 - 45).
dubblering(FKN,par(fkn(F,DM,3),S1),[Grt,Trs,Kvnt],val(bas(Trs),[Kvnt,Trs]),_,5) <=
        (undantag(dubbel_ters,fkn(F,DM,3)) -> false),
        member(DM,[dur,moll]),
        identifiera_ackordton(S1,[Grt],Grt,_).
```

```
% f_D7, f_D7_3, f_D7_5, f_D7_7, f_Dst, f_D4, f_D64, f_oD64
undantag(fri_dubblering,fkn(fD,7,1)).
undantag(fri_dubblering,fkn(fD,7,3)).
undantag(fri_dubblering,fkn(fD,7,5)).
undantag(fri_dubblering,fkn(fD,7,7)).
undantag(fri_dubblering,fkn(fD,7,s)).
undantag(fri_dubblering,fkn(fD,4,1)).
undantag(fri_dubblering,fkn(fD,d64,1)).
undantag(fri_dubblering,fkn(fD,md64,1)).
% f_S65, f_oS65, f_S65_3, f_S65_5, f_S65_6, f_o65_3, f_oS65_5, f_oS65
undantag(fri_dubblering,fkn(fS,65,1)).
undantag(fri_dubblering,fkn(fS,65,3)).
undantag(fri_dubblering,fkn(fS,65,5)).
undantag(fri_dubblering,fkn(fS,65,6)).
undantag(fri_dubblering,fkn(fS,m65,1)).
undantag(fri_dubblering,fkn(fS,m65,3)).
undantag(fri_dubblering,fkn(fD,m65,5)).
undantag(fri_dubblering,fkn(fD,m65,6)).
% f_T6, f_S6, f_D6, f_oT6, f_oS6, f_oD6
undantag(fri_dubblering,fkn(fT,6,1)).
undantag(fri_dubblering,fkn(fT,m6,1)).
undantag(fri_dubblering,fkn(fS,6,1)).
undantag(fri_dubblering,fkn(fS,m6,1)).
undantag(fri_dubblering,fkn(fD,6,1)).
undantag(fri_dubblering,fkn(fD,m6,1)).
% f_S7, f_S7_3, f_S7_5, f_S7_7, f_Sst
undantag(fri_dubblering,fkn(fS,7,1)).
undantag(fri_dubblering,fkn(fS,7,3)).
undantag(fri_dubblering,fkn(fS,7,5)).
undantag(fri_dubblering,fkn(fS,7,7)).
undantag(fri_dubblering,fkn(fS,7,s)).

% Utan kvint, OK endast for tonika
undantag(utan_kvint,fkn(fS,moll,1)).
undantag(utan_kvint,fkn(fS,dur,1)).
undantag(utan_kvint,fkn(fD,dur,1)).
undantag(utan_kvint,fkn(fD,moll,1)).

% f_T5, f_oT5, f_S5, f_oS5, f_D5, f_oD5 - ej dubbel grundton
undantag(dubbel_grundton,fkn(fT,dur,5)).
undantag(dubbel_grundton,fkn(fT,moll,5)).
undantag(dubbel_grundton,fkn(fS,dur,5)).
undantag(dubbel_grundton,fkn(fS,moll,5)).
undantag(dubbel_grundton,fkn(fD,dur,5)).
undantag(dubbel_grundton,fkn(fD,moll,5)).
% f_T5, f_oT5, f_S5, f_oS5, f_D5, f_oD5 - ej dubbel ters
undantag(dubbel_ters,fkn(fT,dur,5)).
undantag(dubbel_ters,fkn(fT,moll,5)).
undantag(dubbel_ters,fkn(fS,dur,5)).
undantag(dubbel_ters,fkn(fS,moll,5)).
undantag(dubbel_ters,fkn(fD,dur,5)).
undantag(dubbel_ters,fkn(fD,moll,5)).
% f_D4 - ej dubbel kvint (dubbel kvart kan ej astadkommas, strangt forbjudet)
undantag(dubbel_kvint,fkn(fD,4,1)).
% f_D3 - basen (= tersen) far ej dubblas
undantag(dubbel_ters,fkn(fD,dur,3)).
undantag(dubbel_ters,fkn(fD,moll,3)).
undantag(dubbel_meloditon,fkn(fD,dur,3)).
undantag(dubbel_meloditon,fkn(fD,moll,3)).
% f_S6
undantag(dubbel_sext,fkn(fS,6,1)).
undantag(dubbel_ters,fkn(fS,m6,1)).
```

```
% Tillatna intervall mellan tenor-alt och alt-sopran
tillatna_avstand([prim,sekund,ters,kvart,kvint,sext,septim,oktav]).


% Ett undantag specificerar en foreteelse som inte foljer monstret.
% Det typiska testet ar
%       undantag(X) -> false,
% vilket misslyckas om det finns en definition
%       undantag(X).

% Undantag ackordfoljd: otillatna ackordfoljder.
undantag(otillaten_foljd(fkn(F,dur,5),fkn(F1,dur,5))).                % Soderholm sid 61
undantag(otillaten_foljd(fkn(F,dur,5),fkn(F1,moll,5))).               % Soderholm sid 61
undantag(otillaten_foljd(fkn(F,moll,5),fkn(F1,dur,5))).               % Soderholm sid 61
undantag(otillaten_foljd(fkn(F,moll,5),fkn(F1,moll,5))).              % Soderholm sid 61
undantag(otillaten_foljd(fkn(fD,7,1),fkn(fT,dur,3))).                 % Soderholm sid 47
undantag(otillaten_foljd(fkn(fD,7,1),fkn(fT,moll,3))).                % Soderholm sid 47

% Undantag bas: basgangar som har speciella krav
%               och inte far anvanda den sista generella klausulen i baston.
undantag(bas(fkn(F,dur,3),fkn(F,dur,1))).
undantag(bas(fkn(F,moll,3),fkn(F,moll,1))).
undantag(bas(fkn(fS,Atyp,1),fkn(fD,Atyp1,1))).
undantag(bas(fkn(fD,Atyp,1),fkn(fT,Atyp1,1))).
undantag(bas(fkn(fT,Atyp,1),fkn(fS,Atyp1,1))).
undantag(bas(fkn(fT,Atyp,1),fkn(fD,Atyp1,1))).
undantag(bas(fkn(fS,Atyp,1),fkn(fT,Atyp1,1))).
undantag(bas(fkn(fT,dur,3),fkn(fS,Typ,1))).
undantag(bas(fkn(fT,moll,3),fkn(fS,Typ,1))).
undantag(bas(fkn(fT,dur,1),fkn(fD,dur,3))).
undantag(bas(fkn(fT,moll,1),fkn(fD,dur,3))).
undantag(bas(fkn(fT,6,1),fkn(fD,dur,3))).

% Undantag tillatna_intervall: vid denna ackordfoljd tillats andra intervall
%               vid stamforflyttning (se tillatna_intervall/3).
undantag(intervall(par(fkn(fD,T,B),fkn(fD,T1,B1)))).

% Undantag fordubbling: Upprakning av de fordubblingar
%               som normalt inte ar tillatna (se dubblering/6).
% f_T, f_oT, f_T3, f_oT3, f_T5, f_oT5
undantag(fri_dubblering,fkn(fT,dur,1)).
undantag(fri_dubblering,fkn(fT,moll,1)).
undantag(fri_dubblering,fkn(fT,dur,3)).
undantag(fri_dubblering,fkn(fT,moll,3)).
undantag(fri_dubblering,fkn(fT,moll,5)).
undantag(fri_dubblering,fkn(fT,dur,5)).
% f_S, f_oS, f_S3, f_oS3, f_S5, f_oS5
undantag(fri_dubblering,fkn(fS,moll,1)).
undantag(fri_dubblering,fkn(fS,dur,1)).
undantag(fri_dubblering,fkn(fS,moll,3)).
undantag(fri_dubblering,fkn(fS,dur,3)).
undantag(fri_dubblering,fkn(fS,dur,5)).
undantag(fri_dubblering,fkn(fS,moll,5)).
% f_D, f_oD, f_D3, f_oD3, f_D5, f_oD5
undantag(fri_dubblering,fkn(fD,dur,1)).
undantag(fri_dubblering,fkn(fD,moll,1)).
undantag(fri_dubblering,fkn(fD,dur,3)).
undantag(fri_dubblering,fkn(fD,moll,3)).
undantag(fri_dubblering,fkn(fD,dur,5)).
undantag(fri_dubblering,fkn(fD,moll,5)).
```

```
baston(fkn(fT,dur,3),fkn(fS,Typ,1),T1,B,_,B1) <=                                % T3 - S
        hoga_tonen(B,liten,sekund,B1).
baston(fkn(fT,moll,3),fkn(fS,Typ,1),T1,B,_,B1) <=                              % oT3 - S
        hoga_tonen(B,stor,sekund,B1).
baston(fkn(fT,_,1),fkn(fD,dur,3),_,B,_,B1) <=                               % alla T - D3
        laga_tonen(B,liten,sekund,B1).
baston(FKN1,FKN2,_,B,Bas,B1) <=                                               % Alla ovriga
        (undantag(bas(FKN1,FKN2)) -> false),                    % om inget undantag finns
        lamplig_oktav(Bas,B,B1).                        % mot att anvanda allmanna fallet.

% Hjalp till baston.
kanske_upp(Ton,Ton).
kanske_upp(T1,T2) <=
        hoga_tonen(T1,ren,oktav,T2).

% Hjalp till baston.
kanske_ner(Ton,Ton).
kanske_ner(T1,T2) <=
        laga_tonen(T1,ren,oktav,T2).

% Deklaration av tillatna omfang for de olika stammorna (Arbetsbok sid 4).
tillatet_omfang(bas,es,d1).
tillatet_omfang(tenor,c,d2).
tillatet_omfang(alt,g,d2).
tillatet_omfang(sopran,c1,a2).

% Deklaration av tillatna intervall vid forflyttning av en stamma.
% S: stamma. FKNS: de tva funktioner som forflyttningen sker emellan.
tillatna_intervall(S,FKNS,
                [[ren,prim],[overstigande,prim],[liten,sekund],[stor,sekund],[liten,ters],
                [stor,ters],[ren,kvart],[ren,kvint]]) <=
        (undantag(intervall(FKNS)) -> false),
        member(S,[alt,tenor]).
% Generellt sopranstamman
tillatna_intervall(sopran,FKNS,
                [[ren,prim],[overstigande,prim],[liten,sekund],[stor,sekund],[liten,ters],
                [stor,ters],[ren,kvart],[ren,kvint],[liten,sext],[stor,sext],
                [liten,septim],[stor,septim],[ren,oktav],[liten,nona],[stor,nona]]) <=
        (undantag(intervall(FKNS)) -> false).
% Basstamman, andra ej dominant
tillatna_intervall(bas,par(fkn(_,_,_),fkn(F,_,_)),
                [[ren,prim],[overstigande,prim],[liten,sekund],[stor,sekund],[liten,ters],
                [stor,ters],[ren,kvart],[ren,kvint],[liten,sext],[stor,sext],
                [liten,septim],[stor,septim],[ren,oktav],[liten,nona],[stor,nona]])
                #{F \= fD} <=
        (undantag(intervall(par(fkn(_,_,_),fkn(F,_,_)))) -> false).
% Basstamman, andra dominant
tillatna_intervall(bas,par(fkn(F,_,_),fkn(fD,_,_)),
                [[ren,prim],[overstigande,prim],[liten,sekund],[stor,sekund],[liten,ters],
                [stor,ters],[forminskad,kvart],[forminskad,kvint],[ren,kvart],[ren,kvint],
                [liten,sext],[stor,sext],[liten,septim],[stor,septim],[ren,oktav],
                [liten,nona],[stor,nona]]) <=
        member(F,[fT,fS]).
% Alt-tenor, dominant till dominant
tillatna_intervall(S,par(fkn(fD,_,_),fkn(fD,_,_)),
                [[ren,prim],[overstigande,prim],[liten,sekund],[stor,sekund],[liten,ters],
                [stor,ters],[ren,kvart],[forminskad,kvint],[ren,kvint]]) <=
        member(S,[alt,tenor]).
% Bas-sopran, dominant till dominant
tillatna_intervall(S,par(fkn(fD,_,_),fkn(fD,_,_)),
                [[ren,prim],[overstigande,prim],[liten,sekund],[stor,sekund],[liten,ters],
                [stor,ters],[ren,kvart],[forminskad,kvint],[ren,kvint],[liten,sext],
                [stor,sext],[liten,septim],[stor,septim],[ren,oktav],[liten,nona],
                [stor,nona]]) <=
        member(S,[bas,sopran]).
```

```
till_fkn(f_D7_7,fkn(fD,7,7)).
till_fkn(f_D4,fkn(fD,4,1)).
till_fkn(f_D64,fkn(fD,d64,1)).
till_fkn(f_oD64,fkn(fD,md64,1)).
till_fkn(f_Dst,fkn(fD,7,s)).
till_fkn(f_T5,fkn(fT,dur,5)).
till_fkn(f_S5,fkn(fS,dur,5)).
till_fkn(f_D5,fkn(fD,dur,5)).
till_fkn(f_oT5,fkn(fT,moll,5)).
till_fkn(f_oS5,fkn(fS,moll,5)).
till_fkn(f_oD5,fkn(fD,moll,5)).
till_fkn(f_S65,fkn(fS,65,1)).
till_fkn(f_oS65,fkn(fS,m65,1)).
till_fkn(f_S65_3,fkn(fS,65,3)).
till_fkn(f_S65_5,fkn(fS,65,5)).
till_fkn(f_S65_6,fkn(fS,65,6)).
till_fkn(f_oS65_3,fkn(fS,m65,3)).
till_fkn(f_oS65_5,fkn(fS,m65,5)).
till_fkn(f_oS65_6,fkn(fS,m65,6)).
till_fkn(f_T6,fkn(fT,6,1)).
till_fkn(f_S6,fkn(fS,6,1)).
till_fkn(f_D6,fkn(fD,6,1)).
till_fkn(f_S7,fkn(fS,7,1)).
till_fkn(f_S7_3,fkn(fS,7,3)).
till_fkn(f_S7_5,fkn(fS,7,5)).
till_fkn(f_S7_7,fkn(fS,7,7)).
till_fkn(f_Sst,fkn(fS,7,s)).
till_fkn(tom,fkn(tom,tom,tom)).

% Upprakning av avstandet (arg 2 och 3) fran tonart till funktionens (arg 1) grundton.
intervall_tonart_funktions_grundton(fT,ren,prim).
intervall_tonart_funktions_grundton(fS,ren,kvart).
intervall_tonart_funktions_grundton(fD,ren,kvint).

% Valjer en lamplig baston B1 till det ackord som har den funktion som ges av andra
% argumentet. Forsta argumentet ar foregaende ackords funktion.
% B ar foregaende ackords baston.
baston(fkn(fD,_,1),fkn(fT,_,1),T1,B,_,B1) <=              % fD - fT med grundton i basen
        laga_tonen(B,ren,kvint,B11),
        kanske_upp(B11,B1),
        under(B1,T1).
baston(fkn(fT,_,1),fkn(fS,_,1),T1,B,_,B1) <=              % fT - fS med grundton i basen
        hoga_tonen(B,ren,kvart,B11),
        kanske_ner(B11,B1),
        under(B1,T1).
baston(fkn(fT,_,1),fkn(fD,_,1),T1,B,_,B1) <=              % fT - fD med grundton i basen
        hoga_tonen(B,ren,kvint,B11),
        kanske_ner(B11,B1),
        under(B1,T1).
baston(fkn(fS,_,1),fkn(fT,_,1),T1,B,_,B1) <=              % fS - fT med grundton i basen
        laga_tonen(B,ren,kvart,B11),
        kanske_upp(B11,B1),
        under(B1,T1).
baston(fkn(fS,_,1),fkn(fD,_,1),T1,B,_,B1) <=              % fS - fD med grundton i basen
        hoga_tonen(B,stor,sekund,B1),
        under(B1,T1).
baston(fkn(F,dur,3),fkn(F,dur,1),T1,B,_,B1) <=               % Sextackord till grundlage
        laga_tonen(B,stor,ters,B1).                             % (Soderholm sid 41:138)
baston(fkn(F,moll,3),fkn(F,moll,1),T1,B,_,B1) <=             % Sextackord till grundlage
        laga_tonen(B,liten,ters,B1).                            % (Soderholm sid 41:138)
baston(fkn(F,_,Bt),fkn(F,_,Bt),T1,B,_,B1) <=                 % Samma funktion och baston
        hoga_tonen(B,ren,prim,B11),
        kanske_upp(B11,B1),
        under(B1,T1).
```

# kbs.def

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Sidhanvisningar galler foljande bocker:
% Soderholm - Valdemar Soderholm:
%               Harmonilara,
%               Trettonde oversedda upplagan,
%               Nordiska Musikforlaget 1959.
% Arbetsbok - Valdemar Soderholm:
%               Arbetsbok i elementar harmonilara,
%               Nordiska Musikforlaget 1951.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Nedanstaende filer behovs:
% :- include_def('bas.def').
% :- include_def('bygg.def').
% :- include_def('ton.def').

% Strategierna finns i filen 'kbs.rul'.

% Definitioner som anvands utat:
%
% till_fkn \\- A \- till_fkn(F,FKN).
% intervall_tonart_funktions_grundton \\- A \-intervall_tonart_funktions_grundton(F,It,I).
% baston \\- A \- baston(F1,F2,T1,T2,T3,T4).
% tillatet_omfang \\- A \- tillatet_omfang(S,T1,T2).
% tillatna_intervall \\- A \- tillatna_intervall(S,FKNS,L).
% tillatna_avstand \\- A \- tillatna_avstand(L).
% dubb_krav \\- A \- dubb_krav(T,P1,P2,P3,R).
% dubblering \\- A \- dubblering(F,P,A,V,Pos,R).
%
% A star for godtycklig antecedent. Ingen hansyn tas till denna.


:- dynamic_term(undantag/1).
:- dynamic_term(undantag/2).
:- dynamic_term(tillatna_intervall/3).
:- dynamic_term(tillatna_avstand/1).


% Extern till intern representation av funktioner.
% Betydelse av fkn(F,T,B):     F star for musikalisk funktion: fT - tonika,
%                                      fS - subdominant, fD - dominant.
%                              T star for ackordtyp : dur, moll, 7 - septimackord
%                                      4 - kvartackord, o.s.v.
%                              B star for vilken ackordton som ar i basen: 1 - grundton
%                                      3 - ters, 5 - kvint, 6 - sext, 7 - septim,
%                                      s - ofullkomlig dominant (Soderholm sid 53).
till_fkn(f_T,fkn(fT,dur,1)).
till_fkn(f_oT,fkn(fT,moll,1)).
till_fkn(f_T3,fkn(fT,dur,3)).
till_fkn(f_oT3,fkn(fT,moll,3)).
till_fkn(f_S,fkn(fS,dur,1)).
till_fkn(f_oS,fkn(fS,moll,1)).
till_fkn(f_S3,fkn(fS,dur,3)).
till_fkn(f_oS3,fkn(fS,moll,3)).
till_fkn(f_D,fkn(fD,dur,1)).
till_fkn(f_oD,fkn(fD,moll,1)).
till_fkn(f_D3,fkn(fD,dur,3)).
till_fkn(f_oD3,fkn(fD,moll,3)).
till_fkn(f_D7,fkn(fD,7,1)).
till_fkn(f_D7_3,fkn(fD,7,3)).
till_fkn(f_D7_5,fkn(fD,7,5)).
```

```
v9_right((C1,C2,C3,C4,C5,C6,C7,C8,C9),PT1,PT2,PT3,PT4,PT5,PT6,PT7,PT8,PT9) <=
        (PT1 -> (A \- C1)),
        (PT2 -> (A \- C2)),
        (PT3 -> (A \- C3)),
        (PT4 -> (A \- C4)),
        (PT5 -> (A \- C5)),
        (PT6 -> (A \- C6)),
        (PT7 -> (A \- C7)),
        (PT8 -> (A \- C8)),
        (PT9 -> (A \- C9))
        -> (A \- (C1,C2,C3,C4,C5,C6,C7,C8,C9)).

v11_right((C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11),
         PT1,PT2,PT3,PT4,PT5,PT6,PT7,PT8,PT9,PT10,PT11) <=
        (PT1 -> (A \- C1)),
        (PT2 -> (A \- C2)),
        (PT3 -> (A \- C3)),
        (PT4 -> (A \- C4)),
        (PT5 -> (A \- C5)),
        (PT6 -> (A \- C6)),
        (PT7 -> (A \- C7)),
        (PT8 -> (A \- C8)),
        (PT9 -> (A \- C9)),
        (PT10 -> (A \- C10)),
        (PT11 -> (A \- C11))
        -> (A \- (C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11)).

v12_right((C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12),
         PT1,PT2,PT3,PT4,PT5,PT6,PT7,PT8,PT9,PT10,PT11,PT12) <=
        (PT1 -> (A \- C1)),
        (PT2 -> (A \- C2)),
        (PT3 -> (A \- C3)),
        (PT4 -> (A \- C4)),
        (PT5 -> (A \- C5)),
        (PT6 -> (A \- C6)),
        (PT7 -> (A \- C7)),
        (PT8 -> (A \- C8)),
        (PT9 -> (A \- C9)),
        (PT10 -> (A \- C10)),
        (PT11 -> (A \- C11)),
        (PT12 -> (A \- C12))
        -> (A \- (C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12)).


                %%%%%%%%%%%
                % Provison %
                %%%%%%%%%%%

% Data (tal, variabler och listor).
data(X) :- number(X).
data(X) :- var(X).
data(X) :- X == [].
data(X) :- nonvar(X), functor(X,'.',2).

% Konstruerare.
constructor('.',2).
```

```
% a_left/4 med malen i omvand ordning.
a1_left((A -> C1),I,PT,PT1) <=
        (PT1 -> (I@[C1|Y] \- C)),
        (PT -> (I@Y \- A))
        -> (I@[(A -> C1)|Y] \- C).

% axiom/3 med restriktioner.
axiom(T,C,I) <=
        data(T),
        data(C),
        unify(T,C)
        -> (I@[T|_] \- C).

v3_right((C1,C2,C3),PT1,PT2,PT3) <=
        (PT1 -> (A \- C1)),
        (PT2 -> (A \- C2)),
        (PT3 -> (A \- C3))
        -> (A \- (C1,C2,C3)).

v4_right((C1,C2,C3,C4),PT1,PT2,PT3,PT4) <=
        (PT1 -> (A \- C1)),
        (PT2 -> (A \- C2)),
        (PT3 -> (A \- C3)),
        (PT4 -> (A \- C4))
        -> (A \- (C1,C2,C3,C4)).

v5_right((C1,C2,C3,C4,C5),PT1,PT2,PT3,PT4,PT5) <=
        (PT1 -> (A \- C1)),
        (PT2 -> (A \- C2)),
        (PT3 -> (A \- C3)),
        (PT4 -> (A \- C4)),
        (PT5 -> (A \- C5))
        -> (A \- (C1,C2,C3,C4,C5)).

v6_right((C1,C2,C3,C4,C5,C6),PT1,PT2,PT3,PT4,PT5,PT6) <=
        (PT1 -> (A \- C1)),
        (PT2 -> (A \- C2)),
        (PT3 -> (A \- C3)),
        (PT4 -> (A \- C4)),
        (PT5 -> (A \- C5)),
        (PT6 -> (A \- C6))
        -> (A \- (C1,C2,C3,C4,C5,C6)).

v7_right((C1,C2,C3,C4,C5,C6,C7),PT1,PT2,PT3,PT4,PT5,PT6,PT7) <=
        (PT1 -> (A \- C1)),
        (PT2 -> (A \- C2)),
        (PT3 -> (A \- C3)),
        (PT4 -> (A \- C4)),
        (PT5 -> (A \- C5)),
        (PT6 -> (A \- C6)),
        (PT7 -> (A \- C7))
        -> (A \- (C1,C2,C3,C4,C5,C6,C7)).

v8_right((C1,C2,C3,C4,C5,C6,C7,C8),PT1,PT2,PT3,PT4,PT5,PT6,PT7,PT8) <=
        (PT1 -> (A \- C1)),
        (PT2 -> (A \- C2)),
        (PT3 -> (A \- C3)),
        (PT4 -> (A \- C4)),
        (PT5 -> (A \- C5)),
        (PT6 -> (A \- C6)),
        (PT7 -> (A \- C7)),
        (PT8 -> (A \- C8))
        -> (A \- (C1,C2,C3,C4,C5,C6,C7,C8)).
```

```
                %%%%%%%%%%%%%%
                % Strategier %
                %%%%%%%%%%%%%%

% Anvands av standardstrategierna (arl, ...)
user_add_right(_,PT) <=
        gt_right(_),
        le_right(_),
        eq_right(_),
        ne_right(_),
        eights(_),
        no_par_move(_).

% Anvands av standardstrategierna (arl, ...)
user_add_left(_,_,PT) <=
        plus_left(_,_),
        minus_left(_,_),
        abs_left(_,_),
        par_move_left(_,PT),
        par_eights_left(_,PT),
        par_fifths_left(_,PT).

                %%%%%%%%%%
                % Regler %
                %%%%%%%%%%

% Lyckas utan att gora nagot.
skip <=
        (_ \- _).

% d_right(_,d_right(_,true_right(_)))
ton(C) <=
        not(data(C)),
        atom(C),
        clause(C,B),
        not(data(B)),
        atom(B),
        clause(B,true)
        -> (_ \- C).

% d_right(_,true_right(_))
dt_right(C) <=
        not(data(C)),
        atom(C),
        clause(C,true)
        -> (_ \- C).

% d_right/2 med restriktioner.
d_right(C,PT) <=
        not(data(C)),
        atom(C),
        clause(C,B),
        (PT -> (P \- B))
        -> (P \- C).

% d_left/3 med restriktioner.
d_left(T,I,PT) <=
        not(functor(T,kp,2)),
        not(functor(T,okp,2)),
        not(functor(T,mr,2)),
        not(data(T)),
        atom(T),
        definiens(T,Dp,N),
        (PT -> (I@[Dp|Y] \- C))
        -> (I@[T|Y] \- C).
```

```
        % Anvands nar andra argumentet innehaller tva element.
        id_acktonE_tvaton <=
                ((ton(_) -> (_ \- ton(Ton,_))),
                format('Ackord nr ~w: Tonen ~w ingar inte i givet ackord.~n',[N,Ton]),
                format(' Tillatna toner: ~w och ~w i alla mojliga oktavlagen.~n',
                 [A,B]))
                -> (_ \- identifiera_ackordton(Ton,[A,B],_,pos(N))).

        % Anvands nar andra argumentet innehaller tre element.
        id_acktonE_treton <=
                ((ton(_) -> (_ \- ton(Ton,_))),
                format('Ackord nr ~w: Tonen ~w ingar inte i givet ackord.~n',[N,Ton]),
                format(' Tillatna toner: ~w - ~w - ~w i alla mojliga oktavlagen.~n',
                 [A,B,C]))
                -> (_ \- identifiera_ackordton(Ton,[A,B,C],_,pos(N))).

        % Anvands nar andra argumentet innehaller fyra element.
        id_acktonE_fyrton <=
                ((ton(_) -> (_ \- ton(Ton,_))),
                format('Ackord nr ~w: Tonen ~w ingar inte i givet ackord.~n',[N,Ton]),
                format(' Tillatna toner: ~w - ~w - ~w - ~w i alla mojliga oktavlagen.~n',
                 [A,B,C,D]))
                -> (_ \- identifiera_ackordton(Ton,[A,B,C,D],_,pos(N))),

        % Anvands nar forsta argumentet ar en ton som inte existerar
        id_acktonE_felton <=
                (not((ton(_) -> (_ \- ton(Ton,_)))),
                format('Ackord nr ~w: Tonen ~w existerar inte.~n',[N,Ton]))
                -> (_ \- identifiera_ackordton(Ton,_,_,pos(N))).


                %%%%%%%%%%
                % Regler %
                %%%%%%%%%%

oktav_avst_right(oktav_avst(X1,X2)) <=
        number(X1),
        number(X2),
        Noll is abs((X2 - X1) mod 12),
        Noll = 0
        -> (_ \- oktav_avst(X1,X2)).

oktav_avst_left(oktav_avst(X1,X2),PT) <=
        number(X1),
        number(X2),
        Noll is abs((X2 - X1) mod 12),
        (Noll = 0,(PT -> (I@[true|R] \- C));
        Noll \= 0,(PT -> (I@[false|R] \- C)))
        -> (I @[oktav_avst(X1,X2)|R] \- C).


                %%%%%%%%%%%
                % Proviso %
                %%%%%%%%%%%

% Konstruerare.
constructor(oktav_avst,2).
```

# grundregler.rul

```
% Nedanstaende fil behovs:
% :- include_rules(lib('rules.rul')).
```

```
% l_o \\- A,l_o(T,L1) \- L2.
l_o <=
        d_left(_,_,a_left(_,_,l_o1(_),axiom(_,_,_))).

% Hjalpstrategi for l_o:
        l_o1(C) <=
                (_ \- C).
        l_o1((lamplig_oktav_hp(_,_,_),(l_o(_,_) -> _))) <=
                v_right(_,lamplig_oktav,a_right(_,l_o)).
        l_o1(lamplig_oktav_hp(_,_,_)) <=
                lamplig_oktav.


                %%%%%%%%%%%%%%%%%%%%%%
                % Kontrollstrategier %
                %%%%%%%%%%%%%%%%%%%%%%

% identifiera_ackordton - kontroll:
identifiera_ackordtonK <=
        d_right(_,v4_right(_,ton(_),member,ton(_),oktav_avst_right(_))),
        d_right(_,v_right(_,a_right(_,removefirst),identifiera_ackordtonK)),
        identifiera_ackordtonE.

% hitta_par - kontroll:
hitta_parK <=
        skip.

% narmast - kontroll:
narmastK <=
        narmast1K,
        narmast2.

% narmast1 - kontroll:
narmast1K <=
        atom(X) -> (_ \- narmast(_,_,X,_)).
narmast1K <=
        d_right(_,identifiera_ackordtonK).


                %%%%%%%%%%%%%%%%%%%%%%%%%%
                % Regler for felutskrift %
                %%%%%%%%%%%%%%%%%%%%%%%%%%

% identifiera_ackordton - felutskrift:
identifiera_ackordtonE <=
        id_acktonE_treton,
        id_acktonE_fyrton,
        id_acktonE_tvaton,
        id_acktonE_baston,
        id_acktonE_enton,
        id_acktonE_felton.

% Hjalpregler for identifiera_ackordtonE

        % Anvands vid kontroll av bastonen.
        id_acktonE_baston <=
                ((ton(_) -> (_ \- ton(Ton,_))),
                format('Ackord nr 1: Bastonen skall vara ~w, inte ~w.~n',[A,Ton]))
                -> (_ \- identifiera_ackordton(Ton,[A],A,pos(0))).

        id_acktonE_enton <=
                ((ton(_) -> (_ \- ton(Ton,_))),
                format('Ackord nr 1: Bastonen skall vara ~w, inte ~w.~n',[A,Ton]))
                -> (_ \- identifiera_ackordton(Ton,[A],A,pos(1))).
```

```
% hitta_par1(_) \\- A,hitta_par1(L1) \- L2.
hitta_par1(A) <=
        (I@[A|R] \- _).
hitta_par1(hitta_par1([])) <=
        d_left(_,_,axiom(_,_,_)).
hitta_par1(hitta_par1([_|_])) <=
        d_left(_,_,a_left(_,_,v_right(_,a_right(_,narm),a_right(_,hitta_par1(_))),
         axiom(_,_,_))).
hitta_par1(hitta_par1(tag_bort(_,_))) <=
        d_left(_,_,a_left(_,_,a_right(_,tag_bort),hitta_par1(_))).
hitta_par1(hitta_par1(lampliga_oktaver(_,_))) <=
        d_left(_,_,a_left(_,_,a_right(_,lampliga_oktaver),hitta_par1(_))).

% narm \\- A,narm(T,L1) \- L2.
narm <=
        d_left(_,_,narm1).

% Hjalpstrategi for narm:
        narm1 <=
                a_left(_,_,v_right(_,a_right(_,narm),mest_nara),axiom(_,_,_)),
                axiom(_,_,_).

% narmast \\- A \- narmast(T1,L,T2,P).
narmast <=
        narmast1,
        narmast2.

% Hjalpstrategier for narmast:
        % Anvands nar T2 ar instansierad
        narmast1 <=
                atom(T2) -> (_ \- narmast(_,_,T2,_)).
        narmast1 <=
                d_right(_,identifiera_ackordton).

        % Anvands nar T2 ar en variabel
        narmast2 <=
                var(T2) -> (_ \- narmast(_,_,T2,_)).
        narmast2 <=
                d_right(_, v_right(_,a_right(_,l_o),a_right(_,narm))).

% tag_bort \\- A,tag_bort(T,L1) \- L2.
tag_bort <=
        tag_bort(_).

% Hjalpstrategi for tag_bort/0:
        tag_bort(A) <=
                (I@[A|R] \- _).
        tag_bort(tag_bort(_,[])) <=
                d_left(_,_,axiom(_,_,_)).
        tag_bort(tag_bort(_,[_|_]))<=
                d_left(_,_,a_left(_,_,v3_right(_,identifiera_ackordton,
                 a_right(_,removefirst),a_right(_,tag_bort(_))),axiom(_,_,_))).

% lampliga_oktaver \\- A,lampliga_oktaver(L1,L2) \- L3.
lampliga_oktaver <=
        lampliga_oktaver(_).

lampliga_oktaver(A) <=
        (I@[A|R] \- _).
lampliga_oktaver(lampliga_oktaver([],_)) <=
        d_left(_,_,axiom(_,_,_)).
lampliga_oktaver(lampliga_oktaver([_|_],_)) <=
        d_left(_,_,a_left(_,_,v_right(_,a_right(_,l_o),
         a_right(_,lampliga_oktaver(_))),axiom(_,_,_))).
```

```
% omringa \\- A \- omringa_ner(T1,T2,T3,T4,T5).
% omringa \\- A \- omringa_upp(T1,T2,T3,T4,T5).
omringa <=
        d_right(_,omringa(_)).

% Hjalpstrategi for omringa/0:
        omringa(C) <=
                (_ \- C).
        omringa(under(_,_)) <=
                under.
        omringa((str_over(_,_),_,_)) <=
                v3_right(_,under,hoga_tonen,omringa).

% ackordtoner \\- A,ackordtoner(Ta,F) \- At.
ackordtoner <=
        d_left(_,_,a_left(_,_,v_right(_,funktions_grundton,ackord),axiom(_,_,_))).

% funktions_grundton \\- A \- funktions_grundton(Ta,F,T).
funktions_grundton <=
 d_right(_,v_right(_,intervall_tonart_funktions_grundton,hoga_tonen)).

% identifiera_ackordton \\- A \- identifiera_ackordton(T1,L,T2).
identifiera_ackordton <=
        d_right(_,v4_right(_,ton(_),member,ton(_),oktav_avst_right(_))),
        d_right(_,v_right(_,a_right(_,ia_reduce),identifiera_ackordton)).

% Hjalpstrategi for identifiera_ackordton:
        ia_reduce <=
                removefirst.

% hitta_par \\- A \- hitta_par(L1,L2,L3).
hitta_par <=
        hitta_p1,
        hitta_p2,
        hitta_p3.

% Hjalpstrategier for hitta_par:
        % Anvands nar X ar en variabel
        hitta_p1 <=
                var(X) -> (_ \- hitta_par(_,_,[[_,X]|_])).
        hitta_p1 <=
                d_right(_,a_right(_,hitta_par1(_))).

        % Anvands nar forsta argumentet ar tomma listan
        hitta_p2 <=
                (_ \- hitta_par([],_,_)).
        hitta_p2 <=
                dt_right(_).

        % Anvands nar X och Y ar instansierade
        hitta_p3 <=
                (atom(X),atom(Y)) -> (_ \- hitta_par(_,_,[[_,X],[_,Y]])).
        hitta_p3 <=
                d_right(_,v_right(_,identifiera_ackordton,identifiera_ackordton)).
```

```
% Givet en lista av toner (forsta argumentet) och Ackordtoner sa ges som svar en lista
% dar till varje ton i forsta argumentet har kopplats en lista dar Ackordtoner befinner
% sig i narmaste lage.
% Ex. arg1=[f1,a], arg2=[e,g] ger svaret [[f1,[e1,g1]],[a,[e,g]]]
% Vid backtracking genereras aven svar dar Ackordtoner befinner sig i nast narmaste lage.
lampliga_oktaver([],_) <=
        [].
lampliga_oktaver([X|Xs],Ackordtoner) <=
        ((l_o(X,Ackordtoner) -> X_At),
        (lampliga_oktaver(Xs,Ackordtoner) -> Resten))
        -> [[X,X_At]|Resten].

% Hjalp till lampliga_oktaver. Givet en ton X och en lista av toner, sa ges som svar
% en lista dar tonerna i andra argumentet har placerats i lamplig oktav i forhallande
% till X (se lamplig_oktav).
% Genererar flera svar vid backtracking.
l_o(X,[Y]) <=
        lamplig_oktav_hp(Y,X,Y1)
        -> [Y1].
l_o(X,[Y|Ys])#{Ys \= []} <=
        (lamplig_oktav_hp(Y,X,Y1),
        (l_o(X,Ys) -> Resten))
        -> [Y1|Resten].
```

# bygg.rul

```
% Nedanstaende filer behovs:
% :- include_rules('ackord.rul').
% :- include_rules('bas.rul').
% :- include_rules('grundregler.rul').
% :- include_rules('ton.rul').


                %%%%%%%%%%%%%%
                % Strategier %
                %%%%%%%%%%%%%%

% lamplig_oktav \\- A \- lamplig_oktav(T1,T2,T3).
lamplig_oktav <=
        d_right(_,lamplig_oktav(_)).

% Hjalpstrategier for lamplig_oktav/0:
        lamplig_oktav(C) <=
                (_\- C).
        lamplig_oktav((ton(_,_),ton(_,_),(oktav_avst(_,_) -> false),
                        lamplig_oktav1(_,_,_))) <=
                v4_right(_,ton(_),ton(_),a_right(_,oktav_avst_left(_,false_left(_))),
                 lamplig_oktav1).
        lamplig_oktav((ton(_,_),ton(_,_),oktav_avst(_,_),_)) <=
                v4_right(_,ton(_),ton(_),oktav_avst_right(_),lamplig_oktav2).

        lamplig_oktav1 <=
                d_right(_,v4_right(_,under,hoga_tonen,omringa,mest_nara)).

        lamplig_oktav2 <=
                d_right(_,lamplig_oktav2(_)).

        lamplig_oktav2(C) <=
                (_ \- C).
        lamplig_oktav2(true) <=
                true_right.
        lamplig_oktav2((hoga_tonen(_,_,_,_);laga_tonen(_,_,_,_))) <=
                o_right(_,(hoga_tonen(_,_,_,_);laga_tonen(_,_,_,_)),hoga_tonen).
```

```
hitta_par([X|Xs],Ackordtoner,Svar) <=
        hitta_par1(lampliga_oktaver([X|Xs],Ackordtoner)) -> Svar.
hitta_par([A,T],Tonlista,[[A,A1],[T,T1]]) <= % Denna klausul anvans endast da alla
        identifiera_ackordton(A1,Tonlista,_,_), % variabler ar instansierade. Finns med
        identifiera_ackordton(T1,Tonlista,_,_). % for effektivare funktionsanalys.

% Hjalpfunktion till hitta_par. Givet en lista pa formen [[Ton,Lista_av_valtoner]|_]
% sa valjs till varje ton ut en efterfoljare och resultatet ar en lista av tonpar.
% Genererar olika forslag vid backtracking.
hitta_par1([]) <=
        [].
hitta_par1([[T,Valtoner]|Xs]) <=
        ((narm(T,Valtoner) -> [Narmast_T]), % hitta den ton som ar narmast
        (hitta_par1(tag_bort(Narmast_T,Xs)) -> Resten)) % tag bort forekomster av denna
        -> [[T,Narmast_T]|Resten]. % och anropa rekursivt
hitta_par1(X)#{X\=[], X\=[_|_]} <= % anvands da argumentet inte
        (X -> Y) % ar en lista
        -> hitta_par1(Y).

% Hjalp till hitta_par for att minska antalet svar
% lamplig_oktav_hp ar en begransning av lamplig_oktav som utnyttjar det faktum att
% alt- och tenorstammorna inte far flyttas en hel oktav.
lamplig_oktav_hp(Ton,Jmfr,Val) <=
        ton(Ton,Nton),
        ton(Jmfr,Njmfr),
        (oktav_avst(Nton,Njmfr) -> false),
        lamplig_oktav1(Ton,Jmfr,Val).
lamplig_oktav_hp(Ton,Jmfr,Val) <=
        ton(Ton,Nton),
        ton(Jmfr,Njmfr),
        oktav_avst(Nton,Njmfr),
        lamplig_oktav2_hp(Jmfr,Val).

lamplig_oktav2_hp(Ton,Ton).

% Givet en ton X och en lista ges som svar den ton i listan som
% ligger narmast X. Vid backtracking ges andra svar (ur listan).
narm(X,[Y]) <=
        [], [Y].
narm(X,[Y|Ys])#{Ys \= []} <=
        ((narm(X,Ys) -> [Z]),
        mest_nara(X,Y,Z,Val))
        -> [Val].

% Givet en ton T och en lista av toner ges som Val den ton i listan som ligger
% narmast T. Vid backtracking ges ovriga toner i Tonlista som Val.
narmast(T,Tonlista,Val,Pos) <= % anvands endast da
        identifiera_ackordton(Val,Tonlista,_,Pos). % Val ar instansierad.
narmast(T,[X|Xs],Val,_) <=
        (l_o(T,[X|Xs]) -> Ol),
        (narm(T,Ol) -> [Val]).

% Givet en ton T1 och en lista pa formen [[_,L1],[_,L2]|_] sa ges som svar en lista
% dar forsta forekomsten av en ton pa oktavavstand till T1 har plockats bort ur
% listorna Li (i=1,2,...,langden pa listan i andra argumentet till tag_bort).
tag_bort(_,[]) <=
        [].
tag_bort(T1,[[T2,Valtoner]|Resten]) <=
        (identifiera_ackordton(T1,Valtoner,Ton_att_ta_bort,_),
        (removefirst(Ton_att_ta_bort,Valtoner) -> Kvar_av_valtoner),
        (tag_bort(T1,Resten) -> Kvar_av_resten))
        -> [[T2,Kvar_av_valtoner]|Kvar_av_resten].
```

```
lamplig_oktav1(Ton,Jmfr,Val) <=
        str_over(Jmfr,Ton),
        hoga_tonen(Ton,ren,oktav,Tonupp),
        omringa_upp(Ton,Jmfr,Tonupp,Lag,Hog),
        mest_nara(Jmfr,Lag,Hog,Val).

% Hjalp till lamplig_oktav.
lamplig_oktav2(Ton,Ton).
lamplig_oktav2(Ton,Val) <=
        hoga_tonen(Ton,ren,oktav,Val);
        laga_tonen(Ton,ren,oktav,Val).

% Givet en ton Lag, sa ar Laga och Hoga oktavforekomster av denna
% sadana att de omringar Jmfr och intervallet mellan dem ar en ren
% oktav.
omringa_ner(Lag,Jmfr,Hog,Lag,Hog) <=
        under(Lag,Jmfr).
omringa_ner(Lag,Jmfr,Hog,Laga,Hoga) <=
        str_over(Lag,Jmfr),
        laga_tonen(Lag,ren,oktav,Lagre),
        omringa_ner(Lagre,Jmfr,Lag,Laga,Hoga).

% Givet en ton Lag, sa ar Laga och Hoga oktavforekomster av denna
% sadana att de omringar Jmfr och intervallet mellan dem ar en ren
% oktav.
omringa_upp(Lag,Jmfr,Hog,Lag,Hog) <=
        under(Jmfr,Hog).
omringa_upp(Lag,Jmfr,Hog,Laga,Hoga) <=
        str_over(Jmfr,Hog),
        hoga_tonen(Hog,ren,oktav,Hogre),
        omringa_upp(Hog,Jmfr,Hogre,Laga,Hoga).

% Givet Tonart och en fkn beskrivande funktion och ackordtyp
% ges som resultat en lista, Ackordtoner, med tonerna i ackordet.
ackordtoner(Tonart,fkn(Funktion,Typ,_)) <=
        (funktions_grundton(Tonart,Funktion,Grundton),
        ackord(Grundton,Typ, Ackordtoner))
        -> Ackordtoner.

% Givet Tonart och en funktion F, sa ges funktionens Grundton.
funktions_grundton(Tonart,F,Grundton) <=
        intervall_tonart_funktions_grundton(F,Ityp,Int),
        hoga_tonen(Tonart,Ityp,Int,Grundton).

% Identitet.
identiska(Ton,Ton).

% Givet en Ton och en lista av toner, Tonlista,
% kontrolleras att Ton befinner sig pa oktavavstand till nagon ton i Tonlista.
% Denna ton ges som Svar. Fjarde argumentet ar en intern positionsangivelse.
identifiera_ackordton(Ton,Tonlista,Svar,_) <=
        ton(Ton,Nton),
        member(Svar,Tonlista),
        ton(Svar,Nsvar),
        oktav_avst(Nton,Nsvar).
identifiera_ackordton(Ton,T,Svar,Pos)#{T \= [], T \= [_|_]} <=     % T = removefirst(A,B)
        (T -> Tonlista),
        identifiera_ackordton(Ton,Tonlista,Svar,Pos).

% Givet toner ingaende i ett ackord (forsta argumentet) och de ackordtoner
% som ar mojliga i nasta (andra argumentet) sa ges mojliga fortsattningar i tredje
% argumentet (se andra klausulen). Som forsta svar ges det alternativ som innebar minsta
% mojliga forflyttning av stammorna. Vid backtracking genereras alla alternativ som
% inte innebar att stammorna flyttas mer an en oktav.
hitta_par([],_,[]).
```

```
ne_right(X\=Y) <=
        number(X),
        number(Y),
        X \= Y
        -> (_ \- X\=Y).


                %%%%%%%%%%%%
                % Provison %
                %%%%%%%%%%%%

% Konstruerare.
constructor('+',2).
constructor('-',2).
constructor('>',2).
constructor('=<',2).
constructor('=',2).
constructor('\=',2).
```

# bygg.def

```
% Nedanstaende filer behovs:
% :- include_def('ackord.def').
% :- include_def('bas.def').
% :- include_def('ton.def').

% Strategierna finns i filen 'bygg.rul'.

% Definitioner som anvands utat:
%
% lamplig_oktav \\- A \- lamplig_oktav(T1,T2,T3).
% omringa \\- A \- omringa_ner(T1,T2,T3,T4,T5).
% omringa \\- A \- omringa_upp(T1,T2,T3,T4,T5).
% ackordtoner \\- A,ackordtoner(Ta,F) \- At.
% funktions_grundton \\- A \- funktionsgrundton(Ta,F,T).
% identiska \\- A \- identiska(X1,X2).
% identifiera_ackordton \\- A \- identifiera_ackordton(T1,L,T2).
% hitta_par \\- A \- hitta_par(L1,L2,L3).
% narmast \\- A \- narmast(T1,L,T2,P).
%
% A star for godtycklig antecedent. Ingen hansyn tas till denna.


% Givet en ton Ton, sa ar Val den oktavforekomst av Ton som ligger
% narmast Jmfr. Vid backtracking ges aven den nast narmaste forekomsten.
lamplig_oktav(Ton,Jmfr,Val) <=
        ton(Ton,Nton),
        ton(Jmfr,Njmfr),
        (oktav_avst(Nton,Njmfr) -> false),
        lamplig_oktav1(Ton,Jmfr,Val).
lamplig_oktav(Ton,Jmfr,Val) <=
        ton(Ton,Nton),
        ton(Jmfr,Njmfr),
        oktav_avst(Nton,Njmfr),
        lamplig_oktav2(Jmfr,Val).

% Hjalp till lamplig_oktav.
lamplig_oktav1(Ton,Jmfr,Val) <=
        str_over(Ton,Jmfr),
        laga_tonen(Ton,ren,oktav,Tonner),
        omringa_ner(Tonner,Jmfr,Ton,Lag,Hog),
        mest_nara(Jmfr,Lag,Hog,Val).
```

```
% plus_minus \\- A, X1 + X2 \- X3.
% plus_minus \\- A, X1 - X2 \- X3.
plus_minus <=
        plus_left(_,_),
        minus_left(_,_).

% member \\- A \- member(X,L).
member <=
        d_right(_,member(_)).

% Hjalpstrategi for member/0:
        member(C) <=
                (_ \- C).
        member(true) <=
                true_right.
        member(member(_,_)) <=
                member.

% removefirst \\- A,removefirst(X,L1) \- L2.
removefirst <=
        d_left(_,_,axiom(_,_,_)),
        d_left(_,_,a_left(_,_,a_right(_,removefirst),axiom(_,_,_))).


                %%%%%%%%%%
                % Regler %
                %%%%%%%%%%

plus_left(I,X+Y) <=
        number(X),
        number(Y),
        Z is X+Y
        -> (I@[X+Y|_] \- Z).

minus_left(I,X-Y) <=
        number(X),
        number(Y),
        Z is X-Y
        -> (I@[X-Y|_] \- Z).

abs_left(I,abso(X)) <=
        number(X),
        Y is abs(X)
        -> (I@[abso(X)|_] \- Y).

gt_right(X>Y) <=
        number(X),
        number(Y),
        X > Y
        -> (_ \- X>Y).

le_right(X=<Y) <=
        number(X),
        number(Y),
        X =< Y
        -> (_ \- X=<Y).

eq_right(X=Y) <=
        number(X),
        number(Y),
        X = Y
        -> (_ \- X=Y).
```

```
% intlista - felutskrift:
ackord2E <=
        format('~nFelaktig ackordtyp: ~w.~n',[Atyp])
        -> (_ \- intlista(Atyp,_,_)).
```

# bas.def

```
% Strategierna finns i filen 'bas.rul'.

% Definitioner som anvands utat:
%
% jfr \\- A \- X1 =< X2.
% jfr \\- A \- X1 > X2.
% jfr \\- A \- X1 = X2.
% jfr \\- A \- X1 \= X2.
% plus_minus \\- A, X1 + X2 \- X3.
% plus_minus \\- A, X1 - X2 \- X3.
% member \\- A \- member(X,L).
% removefirst \\- A,removefirst(X,L1) \- L2.
%
% A star for godtycklig antecedent. Ingen hansyn tas till denna.


% Givet ett tal och en lista av tal
% avgors om talet ingar i listan.
member(X,[X|_]).
member(Y,[X|Xs])#{f(Y,X)\=f(Z,Z)} <=
        member(Y,Xs).
member(Y,Xs)#{Xs\=[], Xs\=[_|_]} <=
        (Xs -> Ws)
        -> member(Y,Ws).

% Givet ett tal och en lista av tal
% ges som resultat en lista dar forsta forekomsten av talet tagits bort.
removefirst(_,[]) <=
        [].
removefirst(X,[X|Xs]) <=
        Xs.
removefirst(Y,[X|Xs])#{f(Y,X)\=f(Z,Z)} <=
        (removefirst(Y,Xs) -> Ws)
        -> [X|Ws].
```

# bas.rul

```
% Nedanstaende fil behovs:
% :- include_rules('grundregler.rul').


                %%%%%%%%%%%%%%
                % Strategier %
                %%%%%%%%%%%%%%

% jfr \\- A \- X1 =< X2.
% jfr \\- A \- X1 > X2.
% jfr \\- A \- X1 = X2.
% jfr \\- A \- X1 \= X2.
jfr <=
        le_right(_),
        gt_right(_),
        eq_right(_),
        ne_right(_).
```

```
        % Anvands for fyratonsackord
        iackord2 <=
                (_ \- iackord([_,_,_,_,_,_],_,_)).
        iackord2 <=
                d_right(_,v3_right(_,hoga_tonen,hoga_tonen,hoga_tonen)).


                %%%%%%%%%%%%%%%%%%%%%%
                % Kontrollstrategier %
                %%%%%%%%%%%%%%%%%%%%%%

% ackord - kontroll:
ackordK <=
        (_ \- ackord(_,_,_)).
ackordK <=
        d_right(_,v_right(_,ackord1K,iackordK)),
        ackord1E.

% Hjalpstrategi for ackordK:
        ackord1K <=
                dt_right(_),
                ackord2E.

% iackord - kontroll:
iackordK <=
        iackordVarK,
        iackordNoVarK.

% Hjalpstrategier for iackordK:
        % Misslyckas om forsta argumentet ar en variabel
        iackordVarK <=
                var(List) -> (_ \- iackord(List,_,_,_)).

        % Anvands om forsta argumentet ar instansierat
        iackordNoVarK <=
                not(var(List)) -> (_ \- iackord(List,_,_,_)).
        iackordNoVarK <=
                iackord1K,
                iackord2K.

        % Anvands for tretonsackord
        iackord1K <=
                (_ \- iackord([_,_,_,_],_,_,_)).
        iackord1K <=
                d_right(_,v_right(_,hoga_tonenK,hoga_tonenK)).

        % Anvands for fyratonsackord
        iackord2K <=
                (_ \- iackord([_,_,_,_,_,_],_,_,_)).
        iackord2K <=
                d_right(_,v3_right(_,hoga_tonenK,hoga_tonenK,hoga_tonenK)).


                %%%%%%%%%%%%%%%%%%%%%%%%%%
                % Regler for felutskrift %
                %%%%%%%%%%%%%%%%%%%%%%%%%%

% ackord - felutskrift:
ackord1E <=
        format('~nFelaktig ton: grundtonen skall vara ~w - inte ~w.~n',
        [Ack,Grt])
        -> (_ \- ackord(Ack,_,[Grt|_])).
```

```
% Givet en lista av intervalltyper och intervallnamn, [Ityp1,Int1,Ityp2,Int2,...]
% samt en grundton Grt, ges som resultat en lista med toner
% pa givet intervall-avstand fran grundtonen.
iackord([Ityp1,Int1,Ityp2,Int2],Grt,[Ton2,Ton3]) <=
        hoga_tonen(Grt,Ityp1,Int1,Ton2),
        hoga_tonen(Grt,Ityp2,Int2,Ton3).
iackord([Ityp1,Int1,Ityp2,Int2,Ityp3,Int3],Grt,[T,K,S]) <=
        hoga_tonen(Grt,Ityp1,Int1,T),
        hoga_tonen(Grt,Ityp2,Int2,K),
        hoga_tonen(Grt,Ityp3,Int3,S).

% For varje typ av ackord anges nedan intervalltyper och
% intervallnamn for de ingaende tonerna, allt raknat fran grundtonen
% ---------- Tretonsackord ----------
intlista(dur,[stor,ters,ren,kvint]).
intlista(moll,[liten,ters,ren,kvint]).
intlista(forminskat,[liten,ters,forminskad,kvint]).
intlista(6,[stor,ters,stor,sext]).
intlista(mT6,[liten,ters,liten,sext]).
intlista(mS6,[liten,ters,stor,sext]).
intlista(mD6,[stor,ters,liten,sext]).
intlista(4,[ren,kvart,ren,kvint]).
intlista(d64,[ren,kvart,stor,sext]).
intlista(md64,[ren,kvart,liten,sext]).
% ---------- Fyratonsackord ----------
intlista(7,[stor,ters,ren,kvint,liten,septim]).
intlista(m7,[liten,ters,ren,kvint,liten,septim]).
intlista(maj7,[stor,ters,ren,kvint,stor,septim]).
intlista(dim,[liten,ters,forminskad,kvint,forminskad,septim]).
intlista(65,[stor,ters,ren,kvint,stor,sext]).
intlista(m65,[liten,ters,ren,kvint,stor,sext]).
```

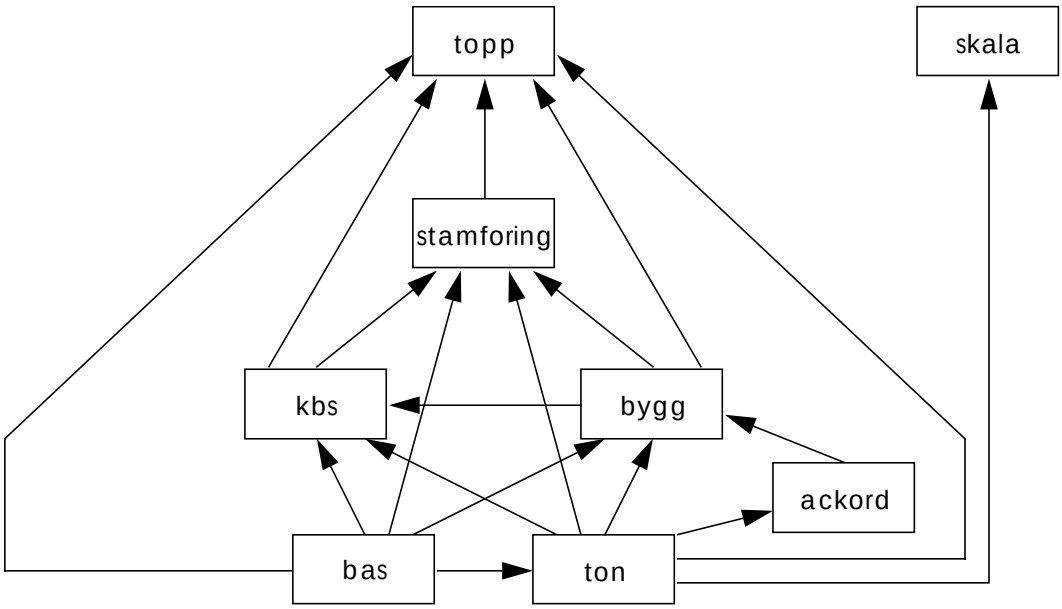
# ackord.rul

```
% Nedanstaende filer behovs:
% :- include_rules('ton.rul').
% :- include_rules('grundregler.rul').


                %%%%%%%%%%%%%%
                % Strategier %
                %%%%%%%%%%%%%%

% ackord \\- A \- ackord(T,At,L).
ackord <=
        d_right(_,v_right(_,dt_right(_),iackord)).

% iackord \\- A \- iackord(L1,T,L2).
iackord <=
        iackord1,
        iackord2.

% Hjalpstrategier for iackord:
        % Anvands for tretonsackord
        iackord1 <=
                (_ \- iackord([_,_,_,_],_,_)).
        iackord1 <=
                d_right(_,v_right(_,hoga_tonen,hoga_tonen)).
```

# Appendix A

## Programlistning

I programlistningen är filerna placerade i alfabetisk ordning. Programmets struktur framgår av figuren nedan.




### ackord.def

```
% Nedanstaende fil behovs:
% :- include_def('ton.def').

% Strategierna finns i filen 'ackord.rul'.

% Definitioner som anvands utat:
%
% ackord \\- A \- ackord(T,At,L).
%
% A star for godtycklig antecedent. Ingen hansyn tas till denna.


% Givet ackordets Grundton och typen av ackord A
% ges som resultat en lista med de ingaende tonerna i angivet ackord.
ackord(Grundton,A,[Grundton,Ton2,Ton3])#{A\=7, A\=m7, A\=maj7, A\=dim, A\=65, A\=m65} <=
        intlista(A,Ints),
        iackord(Ints,Grundton,[Ton2,Ton3]).
ackord(Grundton,A,[Grundton,T,K,S])#{A\=dur, A\=moll, A\=forminskat, A\=4} <=
        intlista(A,Ints),
        iackord(Ints,Grundton,[T,K,S]).
```

# 9 Referenser


[Aro89] M. Aronsson, L.-H. Eriksson, A. Gäredal, L. Hallnäs, P. Olin, *The Programming Language GCLA: A Definitional Approach to Logic Programming*. SICS Research Report R89005B, 1989.

[Aro91] M. Aronsson, *A Definitional Approach to the Combination of Functional and Relational Programming*. SICS Research Report R91:10, 1991.

[Aro92a] M. Aronsson, *GCLA User's Manual*. SICS Technical Report T91:21, 1992.

[Aro92b] M. Aronsson, *Methodology and Programming Techniques in GCLA II*. SICS Research Report R92:05, 1992. Även publicerad i [Eri92].

[Eri88] L.-H. Eriksson, L. Hallnäs, *A Programming Calculus Based on Partial Inductive Definitions*. SICS Research Report R88013, 1988.

[Eri91] L.-H. Eriksson, *A Finitary Version of the Calculus of Partial Inductive Definitions*. Publicerad i [Eri92].

[Eri92] L.-H. Eriksson, L. Hallnäs, P. Schroeder-Heister, *Extensions of Logic Programming*. Proceedings of a workshop held at SICS, February 1991. Springer Lecture Notes in Artificial Intelligence, 1992.

[Fal93] G. Falkman, J. Warnby, *Implementing a small KBS using GCLA II*. Master thesis, Department of Computer Sciences, University of Göteborg, 1993.

[Hal91] L. Hallnäs, *Partial Inductive Definitions*. Theoretical Computer Science, vol. 87, 1991.

[Kre91] P. Kreuger, *GCLA II, A Definitional Approach to Control*. Ph L thesis, Department of Computer Sciences, University of Göteborg, 1991. Även publicerad i [Eri92].

[Lin] H. Lindroth, *Musikalisk satslära*. Allhems Förlag, Malmö.

[Söd51] V. Söderholm, *Arbetsbok i elementär harmonilära*. AB Nordiska Musikförlaget / Edition Wilhelm Hansen, Stockholm, © 1951.

[Söd59] V. Söderholm, *Harmonilära*, trettonde översedda upplagan. AB Nordiska Musikförlaget, Stockholm, © 1959.

- Hur komplicerad en härledningsregel än må vara, talar GCLA-systemets tracer bara om huruvida den lyckas eller ej. Det finns ingen möjlighet till *spårning inuti regler.* Om denna möjlighet fanns, skulle det vara enklare att upptäcka eventuella implementeringsfel på regelnivå.
- I Prolog finns möjlighet att stoppa exekveringen med Ctrl-C och starta spårningen från det läge, i vilket körningen avbröts. På så sätt kommer man snabbare ”in i händelsernas centrum”. Det vore inte så dumt om man i GCLA hade samma möjlighet.

2. Performance Tool.

   Statistikpaketet är till stor hjälp vid effektivisering av program. Även här har vi några förslag till förbättringar.

   - Varje försök till effektivisering av statistikpaketet självt är ett gott initiativ. I den version som vi använt, tar en exekvering med insamlande av statistik mångdubbelt längre tid än en normal exekvering av samma fråga. Detta gör Performance Tool i princip oanvändbart i större sammanhang.
   - När statistikpaketet används inträffar det i vissa sammanhang att en exekvering, som bevisligen förlöper problemfritt utan statistikpaketet, plötsligt avbryts. Man får ett felmeddelande av följande utseende:

     ```
     {ERROR: term too large in assert or record}
     { Execution aborted }
     ```

     Orsaken är att Performance Tool inte klarar av att hantera sekventer av godtycklig storlek. Det är naturligtvis önskvärt att *sekventernas storlek* inte är avgörande för om insamlandet av statistik skall lyckas eller ej.
   - Den nya funktionen `path` är mycket användbar. För att göra den mer lättläst skulle man kunna presentera sökvägen *i trädform* på ett eller annat sätt. Med hjälp av denna funktion torde det inte heller vara någon omöjlig uppgift att automatiskt generera sökstrategier enligt den så kallade *optimalitetsmetoden* (se avsnitt **4.3 Utveckling av sökstrategier**).

## 8.3 Kommentar

Att skriva ett så komplext program som ett musikaliskt expertsystem i ett programmeringsspråk som man aldrig tidigare varit i kontakt med, och samtidigt försöka finna en lämplig formalisering av Söderholms harmonilära, är en stor utmaning. Givetvis kan resultatet inte bli fulländat. Vi tror därför att vi med vår nuvarande kunskap om GCLA II skulle skriva ett bättre program som i större utsträckning utnyttjar fördelarna med programspråket. Därmed inte sagt att vi är helt missnöjda med resultatet.

```
% oktav_paraller är en så kallad konstruerare, som beräknas av
% en egen regel och lyckas om oktavparalleller föreligger
% mellan dess argument.
inga_oktpar(par([B,T,A,S],_),par([B1,T1,A1,S1],_)) <=
       (oktav_paraller([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) ->
         false).
```

Detta är en lika obegriplig definition av oktavparalleller som den föregående. Den innebär i praktiken att definitionen av toner måste flyttas till regelnivån eller fördubblas. Det är som synes inte klart vilket som är det bästa valet.

4. Negation.

   En av anledningarna till att vi utför kontrollen av oktavparalleller på regelnivå är att det är svårt att negera komplicerade definitioner. Vi har inte ägnat särskilt mycket tid åt att finna strategier för negation. Istället har vi ”löst” problemet genom att antingen skriva om definitionen utan negation eller flytta beräkningen till regelnivå.

5. Konserverande strategier.

   Vi har också gjort erfarenheten att sökstrategierna har en *konserverande effekt*. Man gör inte gärna småjusteringar i programmet om det inte är absolut nödvändigt, eftersom detta ofta medför ändringsarbete även på strateginivå. Detta innebär en risk för sämre och mindre genomarbetade program än nödvändigt. Man måste dock framhålla att användningen av sökstrategier har stora fördelar - inte minst det faktum att samma definition kan användas för flera olika ändamål.

### 8.2.2 Programmeringsmiljö

1. Tracer.

   GCLA-systemets tracer är ett oumbärligt verktyg vid utveckling och felsökning av program. I stort fungerar den tillfredsställande, men vi har likväl vissa förslag till förbättringar:

   - Kommandot `r xxx` kan användas för att återgå till ett tidigare anrop med nummer `xxx`. Vi har märkt tydliga behov av ett motsvarande kommando `l xxx` för att hoppa över exekveringen *fram till ett givet anrop*. Detta är användbart när man redan vet hur början ser ut och inte är intresserad av att studera densamma.
   - Spy-points kan placeras på regler och strategier för att avbryta exekveringen vid anrop av dessa. Det vore önskvärt att kunna markera även *definitioner* på detta sätt. Det förekommer ju att samma strategi eller regel används för olika definitioner, som inte alla är lika intressanta att studera. Denna förändring skulle alltså leda till förbättrad precision i spårningen, jämfört med version 0.2.

ningsregler gör, hur man ändrar härledningsregler för att få dem att göra vad man vill o.s.v. Vi tror inte att standardstrategierna `gcla`, `arl` med flera är användbara till något annat än att prova vissa mindre definitioner. Vid större program blir sökrymden alltför stor. Under de tidigare faserna av vårt arbete hade vi svårt att se någon vits med ett programspråk där man är tvungen att ägna så mycket tid åt hur definitionen skall utnyttjas. Med tiden har vi dock funnit även fördelar med detta.

3. Att dra gränsen mellan objektnivå och metanivå.

   Ett annat problem är att avgöra var de olika programdelarna hör hemma. Hur mycket skall ligga på regel- respektive definitionsnivå? I vilka fall bör man skriva nya regler? Vad är den principiella skillnaden mellan en regel och en strategi?

   Vi tittar på ett exempel ur vårt program.

```
% Härledbart om inga oktavparalleller förekommer mellan
% ackorden [B,T,A,S] och [B1,T1,A1,S1].
inga_oktpar(par([B,T,A,S],_),par([B1,T1,A1,S1],_)) <=
       ton(B,Bn),ton(T,Tn),
       ton(A,An),ton(S,Sn),
       ton(B1,B1n),ton(T1,T1n),
       ton(A1,A1n),ton(S1,S1n),
       (oktav_paraller([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) ->
         false).

% Härledbart om oktavparalleller förekommer mellan ackorden X
% och Y. okp är en så kallad konstruerare.
oktav_paraller(X,Y) <=
       okp(X,Y).
```

   Själva kontrollen av oktavparallell-förekomster ligger på regelnivån. Det närmaste man kommer en *definition* av oktavparalleller, är att oktavparalleller föreligger om `okp` lyckas. Sett som en definition av oktavparalleller är detta sällsynt obegripligt, men lägger vi hela definitionen på definitionsnivån kommer vi att få en definition som blir hopplöst ineffektiv att utföra till vänster om deduktionssymbolen ‘`\-`’.

```
oktav_paraller([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) <=
       (oktav_avst(Bn,Tn),oktav_avst(B1n,T1n),
       ((Bn \= B1n);(Tn \= T1n)));
       (oktav_avst(Bn,An),oktav_avst(B1n,A1n),
       ((Bn \= B1n);(An \= A1n)));
       (oktav_avst(Bn,Sn),oktav_avst(B1n,S1n),
       ((Bn \= B1n);(Sn \= S1n)));
       (oktav_avst(Tn,An),oktav_avst(T1n,A1n),
       ((Tn \= T1n);(An \= A1n)));
       (oktav_avst(Tn,Sn),oktav_avst(T1n,S1n),
       ((Tn \= T1n);(Sn \= S1n)));
       (oktav_avst(An,Sn),oktav_avst(A1n,S1n),
       ((Tn \= T1n);(Sn \= S1n))).
```

   Man kan också tänka sig att även regelnivån har tillgång till tonernas interna representationsnummer. Om så var fallet hade man kunnat ha följande definition:

2. Default-resonemang.

   Det är mycket enkelt att utföra default-resonemang av den typ vi beskrivit i avsnitt **6.2.6.2**. Detta underlättar naturligtvis skrivandet av system som använder sig av default-resonemang.

3. Använda samma definition till olika ändamål.

   Vårt program har en definition som utnyttjas på tre olika sätt som ger tre olika procedurella beteenden. Att på detta sätt kunna beskriva systemets *kunskap* en gång och sedan beskriva hur den skall användas på olika sätt är naturligtvis en stor fördel. Det behöver dock inte nödvändigtvis innebära att programmet blir kortare. I vårt fall hade man i exempelvis Prolog säkerligen kunnat ha stora delar av programmet i en upplaga och använt sig av olika varianter bara för vissa delar av programmet. På så sätt hade Prologprogrammets totala omfattning knappast blivit större än GCLA-programmets, där regeldefinitionen tillkommer.

4. Löst språk.

   I GCLA ges möjlighet att göra nästan vad som helst. Man kan ersätta de fördefinierade härledningsreglerna med sina egna, flytta delar av definitionen till regelnivå, blanda funktionell och relationell programmering o.s.v. Det mesta man kan tänkas vilja göra borde vara möjligt.

#### 8.2.1.2 Svårigheter

Under utvecklingen av vårt program har vi också stött på olika typer av svårigheter. Vi tar här upp de viktigaste.

1. Att lära sig språket.

   Ett problem har varit att, med hjälp av den begränsade litteratur som hittills finns på området, få en intuitiv bild av hur programspråket med sina definitioner, regler, provison och strategier fungerar. För en nybörjare tycks det vara ganska svårt att lära sig GCLA II, i all synnerhet om man inte har någon större erfarenhet av Prolog eller andra logikprogrammeringsspråk. Vi efterlyser en lättfattlig och grundlig beskrivning av språkets uppbyggnad med mer intuitiva förklaringar av bland annat regler och strategier, till exempel i form av en lärobok i GCLA-programmering eller som ett slags informell semantik med många och utförliga programexempel. För att underlätta självstudier bör den i alla händelser vara mer omfattande än [Aro92b] och mindre teoretisk än [Kre91].

2. Att skriva bra strategier.

   För att återigen göra jämförelsen med Prolog, måste man i GCLA beskriva hur ett program (definitionen) skall utföras (av regeldefinitionen), medan kontrollen av detta i Prolog sköts av systemet, som alltid använder sig av resolutionsregeln. En GCLA-programmerare måste alltså verkligen sätta sig in i hur ett bevis byggs, vad olika härled-

# 8 Resultat

## 8.1 Söderholms harmonilära

En del av vår uppgift bestod i att försöka utröna i vilken mån kunskapen som förmedlas i [Söd59] går att formalisera på ett sådant sätt att den kan ligga till grund för ett expertsystem. Vi har funnit att vissa delar av denna kunskap på ett ganska enkelt sätt går att uttrycka precist, men också att stora delar är svåra att hantera. De enkla delarna beskriver vilka satser som är tillåtna. Det går därför ganska enkelt att utföra kontroll av en given sats. Det som är svårare att hantera är dels sådan kunskap där det går att dra vissa riktlinjer, dels sådant som helt enkelt lämnas underförstått eller osagt. Ett typiskt exempel på det senare är när man skall välja en annan fördubbling av ett ackord än den mest vanliga. Här räcker inte innehållet i [Söd59] till, utan det förutsätts att läsaren själv skall kunna bilda sig en uppfattning utifrån det fåtal exempel och riktlinjer som ges. Den inte oväntade slutsatsen blir alltså att man behöver tillgång till ett bredare kunskapsunderlag för att implementera ett expertsystem som på bästa sätt skriver fyrstämmig körsats. Denna kunskap skulle till exempel kunna utgöras av ett större antal harmoniläror och ett antal livs levande experter att fråga om råd.

## 8.2 Synpunkter på GCLA

De synpunkter på programspråk och programmeringsmiljö som följer i detta avsnitt är grundade på användning av version 0.2 av GCLA II. Nyare upplagor av programspråket kan därför ha utrustats med vissa av de förbättringar som föreslås nedan.

### 8.2.1 Programspråket GCLA II

I följande avsnitt framför vi våra synpunkter på GCLA II, utifrån de erfarenheter vi har gjort under arbetets gång. Vi gör dock inga anspråk på att ha utnyttjat hela potentialen hos programspråket, eller ens känna till alla möjligheter som ges vid programmering i GCLA. Vi kommer i tillämpliga fall att göra jämförelser med Prolog, eftersom vi ser GCLA främst som ett logikprogrammeringsspråk, och Prolog utgör standardexemplet på ett sådant.

#### 8.2.1.1 Möjligheter

GCLA är ett programmeringsspråk som ger flera möjligheter som Prolog saknar. Vi har utnyttjat vissa av dem:

1. Funktionell programmering.

   Möjligheten att skriva funktionella program (som även kan köras baklänges) uppfattar vi som något positivt. För att få det beteende man väntar sig måste man dock se till att klausuler för olika fall är ömsesidigt uteslutande.

## 7.5 Övrigt

I programmets nuvarande skick är det ganska arbetskrävande att ställa frågor - det är väldigt många `vert` och parenteser att hålla reda på. I en verklig implementering skulle man naturligtvis lägga ner mer arbete på att göra programmet användarvänligt, exempelvis med hjälp av fönster- och menyhantering. Utdata skulle lämpligen presenteras i notskrift.

Vid satskontroll kan man få reda på att en fördubbling inte är tillåten. Däremot får man ingen fullständig information om *varför* fördubblingen är förbjuden. Detta tycks vara mycket komplicerat att åtgärda, men är icke desto mindre ett önskvärt beteende hos programmet.

I kontrollstrategierna saknas en kontroll av att ackorden innehåller fyra toner.

viss musikalisk regel skall sättas ur spel vid ett givet ackord eller att en viss följd av ackordfunktioner skall innebära att fördubblingsmönstret ändras.

## 7.4 Kontrollstrategier och funktionella definitioner

Felsökning av en given sats fungerar tillfredsställande så länge det rör sig om definitioner som implementerats relationellt som nedan.

```
omfang_OK(S,Ton,_) <=
       tillatet_omfang(S,Lag,Hog),
       under(Lag,Ton),
       under(Ton,Hog).
```

Kontrollen av denna definition hanteras av strategin `omfang_OK3K` i filen `stamforing.rul`. Under lång tid hade vi dock svårigheter vid felsökning med funktionellt implementerade definitioner, exempelvis kontroll av en given skala.

```
skala(Grundton,Skaltyp)#{Skaltyp \= mel_moll} <=
       ityper(Skaltyp,Ityper)
       -> iskala(Ityper,Grundton).
...
```

I detta fall utför den vanliga strategin `skala` följande operationer:

- härleder det *förväntade* resultatet av aktuellt funktionsanrop
- jämför detta med det *instansierade* svaret
- ”rapporterar” om det inte stämmer, men talar inte om vad felet är

Man fick alltså veta *att* ett fel påträffats, men inte *vilken typ* av fel. Lösningen på detta problem visade sig vara en enkel ändring i regeln `a_left/4`.

```
a_left((A -> C1),I,PT,PT1) <=
       (PT -> (I@Y \- A)),
       (PT1 -> (I@[C1|Y] \- C))
       -> (I@[(A -> C1)|Y] \- C).
```

Genom att byta plats på de två mellersta raderna i denna regel ändrar vi evalueringsordningen, så att man först tar fram det instansierade svaret och därefter kontrollerar om detta kan härledas från funktionsanropet. På så sätt får man också samma typ av felutskrifter som i resten av systemet.

# 7 Programmets brister

## 7.1 Effektivitet

Trots utveckling av effektivare sökstrategier som avsevärt förkortar exekveringstiden har vi fortfarande problem med att simuleringen går alltför långsamt. Sökrymden exploderar i dessa fall fakultetiskt i förhållande till satsens längd. Detta problem uppstår i synnerhet vid fel som kräver omfattande backtracking. Satskontroll och funktionsanalys får dock anses ha tillräcklig effektivitet.

## 7.2 Dubbletter av svaren

Det kanske största problemet med vårt program är det stora antalet dubbletter av varje svar som ges som förslag vid simulering. Detta innebär att man ofta måste gå igenom en mängd identiska svar innan man får ett nytt. Programmets stora komplexitet och starka inslag av indeterminism har gjort det komplicerat att eliminera dessa dubbletter. Den huvudsakliga orsaken till dubbletterna står att finna i definitionen av `hitta_par/3`, `bygg.def`.

## 7.3 Musikaliska begränsningar

Eftersom vår tid inte är obegränsad, har vi varit tvungna att begränsa systemets musikaliska kunskap enligt följande (se även avsnitt **3.2 Avgränsning av problemet**).

I sitt nuvarande skick omfattar systemet sex oktaver. I vissa (sällsynta) fall kan det kanske vara önskvärt med ett större tonomfång. För att åstadkomma detta krävs bara några enkla ingrepp i filen `ton.def`. Planer på att parametrisera oktaverna för att få större generalitet och utbyggbarhet har förekommit, men inte omsatts i praktiken.

Dubbelhöjda och dubbelsänkta toner som cississ (dubbelhöjt c) och essess (dubbelsänkt e) är inte implementerade. Det är dock mycket enkelt att åtgärda - bara att lägga till dem som extra klausuler i definitionen av `alter/3`, `ton.def`.

Vi har också varit tvungna att begränsa mängden av ackordfunktioner som skall kunna hanteras av programmet. Endast tonika-, dominant- och subdominantfunktioner i sina vanligaste former är implementerade. Orsaken till detta är att arbetet annars hade blivit alltför omfattande. Att införa flera musikaliska funktioner är relativt enkelt, om än inte trivialt.

Det finns inget sätt att ange notvärden i programmet. En fyrstämmig sats definieras endast som en följd av fyrtuppler. I vilken rytm dessa ackord skrivs framgår inte av vår representation. Att införa rytm och notvärden är komplicerat och kräver stora ändringar i implementeringen.

Användaren har möjlighet att tillåta företeelser som normalt är förbjudna och vice versa. Denna möjlighet är dock begränsad. Det vore till exempel önskvärt att kunna ange att en

```
harmo(Ton,vert(FN1,[B,T,A,S],Pos),vert(FN2,[B1,T1,A1,S1],Pos1),
        vert(F3,T3),Dr) <=
             till_fkn(FN1,FKN1),
             till_fkn(FN2,FKN2),
             (undantag(otillaten_foljd(FKN1,FKN2)) -> false),
             (ackordtoner(Ton,FKN2) -> Acktoner),
             narmast(S,Acktoner,S1,Pos1),
             dubb_krav(Ton,par(FKN1,[B,T,A,S]),
               par(FKN2,[B1,T1,A1,S1]),par(F3,T3),Dr),
             dubblering(FKN1,par(FKN2,S1),Acktoner,
               val(bas(Bas),Kvar),par([B1,T1,A1],Pos1),Dr),
             hitta_par([A,T],Kvar,[[A,A1,[T,T1]]),
             ingen_stamkorsning(T1,A1,S1),
             baston(FKN1,FKN2,T1,B,Bas,B1),
             stamforing_OK(Ton,par(FKN1,FKN2),par([B,T,A,S],Pos),
               par([B1,T1,A1,S1],Pos1)).
```

Vid anrop av `harmo` vid funktionsanalys är `FN1` redan instansierad och det är funktionen `FN2` som skall bestämmas. Detta görs genom att den instansieras till någonting i kroppens andra villkor. Därefter avgör tredje till åttonde villkoren i kroppen om detta var en korrekt instansiering, genom att se om kraven i dessa går att uppfylla med det värde på funktionen som valts i andra villkoret. Till detta används samma strategier som vid simulering, med undantag för `dubb_krav`. Den utförs inte eftersom den då skulle instansiera `F3`, vilket inte är önskvärt. Därefter kan man tillämpa kontrollstrategierna på `stamforing_OK` för att få information om eventuella stämföringsfel. Strategin `harmoA` får följande utseende:

```
harmoA <=
       harmo(und_pil_false,ackordtoner,narmast,dubb_kravA,
         fordubblingS,hitta_p3,ingen_stamkorsning3K,baston,
         stamforing_OKA).
```

Vi anropar alltså strategin `harmo` med ett antal argument som specificerar hur denna skall utföras. Genom att variera dessa fås olika beteenden. Exempelvis lyckas `dubb_kravA` varje gång utan att göra något med sekventen (se `skip` i avsnitt **6.2.9 Grundregler**), medan man vid simulering istället använder `dubb_krav`, som kontrollerar att de krav som ställts är uppfyllda. Strategin `harmo` ser ut så här:

```
harmo(Und_pil_false,Ackordtoner,Narmast,Dubb_krav,Dubblering,
  Hitta_par,Ingen_stamkorsning3,Baston,Stamforing_OK) <=
       d_right(_,v11_right(_,till_fkn,till_fkn,Und_pil_false,
         a_right(_,Ackordtoner),Narmast,Dubb_krav,Fordubbling,
         Hitta_par,Ingen_stamkorsning3,Baston,Stamforing_OK)).
```

Varje villkor i kroppen har sin egen strategi och genom att ge olika strategier som parametrar får vi olika beteenden hos samma atom `harmo/5`.

#### 6.3.2.2 Felutskrift

Felutskriftsregeln i vårt exempel har följande utseende (en felutskriftsregel utmärks av suffixet `E`):

```
ingen_stamkorsningE <=
       (((ton(_) -> (_ \- ton(S,_))),
       (ton(_) -> (_ \- ton(A,_))),
       (ton(_) -> (_ \- ton(T,_))),
       (ton(_) -> (_ \- ton(B,_))),
       format('Stämkorsning i ackord nr ~w:~n',[N]),
       format('  Sopran  ~w~n  Alt     ~w~n',[S,A]),
       format('  Tenor   ~w~n  Bas     ~w~n',[T,B]))
       -> (_ \- ingen_stamkorsning(par([B,T,A,S],pos(N))))).
```

Eftersom kontrollen inte får misslyckas, måste vi fortsätta även om en given ton inte existerar i systemet. Detta ger upphov till en mängd följdfel, då denna ton ”inte passar in i mönstret”. De fyra första raderna i regeln ovan, på formen

```
(ton(_) -> (_ \- ton(S,_)))
```

används sålunda för att kontrollera att det aktuella felet inte är ett följdfel av att någon ton inte existerar. Skulle detta vara fallet, ges ingen felutskrift. Regeln misslyckas istället och överlåter till kontrollstrategin att se till att felsökningen fortsätter. Genom den sista raden i kontrollstrategin, *nödutgången*, garanteras att denna lyckas även om felutskriften ”misslyckas” (se avsnitt **6.2.9 Grundregler**).

## 6.4 Funktionsanalys

Funktionsanalys med viss stämföringskontroll åstadkoms genom att ha en blandning av strategierna för simulering och satskontroll. Kortfattat kan man säga att de delar av programmet som väljer ut toner vid simulering använder samma strategi som vid ren simulering och de delar av programmet som utför ren kontroll av stämföring använder sig av kontrollstrategier. Det hela fungerar, tack vare att valet av funktion sker innan stämföringskontrollen tar vid.

## 6.5 Generella strategier

För att visa hur man kan modifiera strategierna, så att de fungerar för såväl simulering som kontroll och analys, tittar vi på definitionen av `harmo/5` som innehåller en uppräkning av vad som är en korrekt förbindelse mellan två ackord.

felet, varefter kontrollen skulle avbrytas. Vid upptäckt av ett stämföringsfel används därför en felutskriftsregel, som alltid lyckas efter att först ha informerat användaren om felets art. I vissa undantagsfall kan dock regeln misslyckas (se nedan). Det blir då kontrollstrategins uppgift att se till att felsökningen fullföljs.

#### 6.3.2.1 Kontroll

Låt oss illustrera det hela med ett exempel från modulen `stamforing`.

```
ingen_stamkorsning(par([B,T,A,S],_)) <=
       under(B,T),
       under(T,A),
       under(A,S).
```

Ovanstående atom kontrollerar att ingen stämkorsning förekommer i ackordet <B,T,A,S>, d.v.s. att ingen av stämmorna har en högre ton än de stämmor som normalt ligger ovanför. Tenorstämmans ton måste exempelvis vara lägre än altstämmans, men högre än basens.

Grundstrategin för `ingen_stamkorsning/1` ser ut så här:

```
ingen_stamkorsning <=
       d_right(_,v3_right(_,under,under,under)).
```

Den används vid simulering, när expertsystemet skall kontrollera om ett *härlett* ackordförslag uppfyller kravet på icke-korsande stämmor. Om man däremot vill kontrollera om ett *givet* ackord uppfyller samma krav, använder man istället kontrollstrategin.

```
ingen_stamkorsningK <=
       ingen_stamkorsning,                    % Grundstrategi
       ingen_stamkorsningE,                 % Felutskriftsregel
       skip.                              % Nödutgång (se nedan)
```

Kontrollstrategierna känns igen på att de har suffixet `K`.

Kontrollen utföres genom att först applicera grundstrategin och, om denna misslyckas, därefter informera om felets art med hjälp av felutskriftsregeln. Detta är grundtanken för kontrollstrategiernas uppbyggnad. Ibland måste man dock göra vissa justeringar i grundstrategin innan den är tillämpbar - vanligtvis beroende på att även den måste innehålla vissa kontrollstrategier (se strategierna `ackord` och `ackordK` i modulen `ackord`).

Exempel:

```
| ?- skip \\- true \- false.

yes
```

## 6.3 Satskontroll

När man har skrivit en fyrstämmig sats kan det vara av intresse att kontrollera om den uppfyller de stämföringskrav som definieras av Söderholms harmonilära [Söd59]. Man kan dessutom vilja veta om satsen uppfyller andra grundläggande musikaliska regler - om alla toner existerar, om stämmorna håller sig inom rimliga omfång o.s.v. För detta ändamål behövs någon typ av felsökningsmekanism.

### 6.3.1 Princip

Felsökningsmekanismen bygger på idén att definitionerna utgör den del av programmet, som är mest intressant för användaren och om möjligt bör bevaras intakt. Vi har därför valt att implementera denna del av systemet på metanivå. Detta medför att man vid satskontroll använder samma definitioner som vid simulering, men applicerar en annan strategi.

Vid simulering kan en körning se ut så här

```
| ?- simulering \\- sats(c,[vert(f_D,[g,d1,g1,h1]),
   vert(f_T,[_,_,_,c2])]) \- Arr.

Arr = [[g,d1,g1,h1],[c,e1,g1,c2]] ?

yes
```

medan satskontroll kan utföras på följande sätt:

```
| ?- kontroll \\- sats(c,[vert(f_D,[g,d1,g1,h1]),
   vert(f_T,[c,c1,e1,g1])]) \- _.

Kontroll av ackord nr 1 - 2
Medrorelse mellan ackord nr 1 och 2:
  Sopran  h1 - g1
  Alt     g1 - e1
  Tenor   d1 - c1
  Bas     g - c

yes
```

### 6.3.2 Funktion

Eftersom systemet arbetar sig igenom satsen ackord för ackord, är det nödvändigt att en kontrollstrategi inte tillåts misslyckas, då detta skulle innebära att man bara kom till första

Som exempel på användning kan vi se på strategin för `under/2` från modulen `ton`.

```
% under \\- A \- under(T1,T2).
under <= d_right(_,v3_right(_,ton(_),ton(_),jfr)).
```

Modulen `bas` innehåller också ett par listhanteringsfunktioner. För att avgöra om ett element tillhör en given lista använder man `member/2`. Med `removefirst/2` plockar man bort första förekomsten av ett visst element ur en given lista.

### 6.2.9 Grundregler

Utöver standardreglerna i `rules.rul` har vi även definierat ett antal nya härledningsregler som används i hela programmet. Dessa finns i modulen `grundregler`, som endast består av en regelfil.

När man har ett antal termer till höger om deduktionssymbolen blir det tjatigt att behöva skriva följder av `v_right` i den här stilen:

```
v_right(_,PT1,v_right(_,PT2,v_right(_,PT3,PT4)))
```

Därför har vi infört ett antal `vi_right`-regler med vilkas hjälp man kan formulera ovanstående på ett enklare sätt:

```
v4_right(_,PT1,PT2,PT3,PT4)
```

Regeldefinition:

```
v4_right((C1,C2,C3,C4),PT1,PT2,PT3,PT4) <=
         (PT1 -> (A \- C1)),
         (PT2 -> (A \- C2)),
         (PT3 -> (A \- C3)),
         (PT4 -> (A \- C4))
         -> (A \- (C1,C2,C3,C4)).
```

Det skulle naturligtvis gå lika bra att kalla varje `vi_right`-regel för `v_right`, eftersom de ändå särskiljs genom olika ställighet.

Vi har också en regel `skip` som visar en del av en härledning utan att göra någonting. I detta fall uppkommer inga variabelbindningar. Med denna regel kan man visa vad som helst.

```
skip <= (_ \- _).
```

```
plus5(X,Y) <=
       (X+5 -> Y).

storre(X,Y) <=
       X > Y.
```

eller för att ta exempel ur modulen `ton`

```
under(Ton1,Ton2) <=
       ton(Ton1,X1),
       ton(Ton2,X2),
       X =< X2.

dist(Ton1,Ton2,Avstand) <=
       ton(Ton1,N1),
       ton(Ton2,N2),
       (N2 - N1 -> Avstand).
```

De aritmetiska funktioner som finns är addition, subtraktion och absolutbelopp, samt jämförelseoperatorerna `>`, `=<`, `=` och `\=`. Regeln för addition ser ut så här:

```
plus_left(I,X+Y) <=
       number(X),
       number(Y),
       Z is X+Y
       -> (I@[X+Y|_] \- Z).
```

`number/1` är ett fördefinierat proviso som kontrollerar att dess argument är ett tal. Själva additionen utförs med Prolog-aritmetik (`Z is X+Y`). Vi visar också regeln `gt_right`.

```
gt_right(X>Y) <=
       number(X),
       number(Y),
       X > Y
       -> (_ \- X>Y).
```

Det finns även ett par strategier att använda: `jfr` används vid jämförelse och användningen av `plus_minus` torde framgå av dess namn.

```
% jfr \\- A \- X1 op X2 där op är någon av =<, >, =, \=
jfr <=
       le_right(_),
       gt_right(_),
       eq_right(_),
       ne_right(_).
```

Notera att den erfarne komponisten/arrangören har tillåtelse att göra vad han vill. Vi har därför gjort det möjligt för användaren att ta bort förbud. Många av stämföringsreglerna kan sättas ur spel med hjälp av ”kommandona” `tillat` och `forbjud`. Oktavparalleller, som normalt är förbjudna, kan tillåtas genom att man i antecedenten lägger till kommandot

```
tillat(oktavparalleller)
```

som med hjälp av en fördefinierad konstruerare `rem_def/2` plockar bort definitionen av `oktav_paraller/2`.

#### 6.2.7.1 Positiv och negativ implementering

I implementeringshänseende kan man dela in stämföringsreglerna i två huvudgrupper; regler med positiv respektive negativ implementering. Dessa typer av regler påverkar på olika sätt instansieringen av fyrtuppler (ackord).

1. Positiv implementering.

   En positivt implementerad regel styr instansieringen av ett ackord. I implementeringen anges hur instansen skall härledas.

   De flesta *påbud* har positiv implementering. I avsnitt **3.3.1.7** finns ett påbud som gäller omläggning från sextackord till treklang. Detta påbud är positivt implementerat som en del av relationen för vilken baston som följer på en annan (se filen `kbs.def`).

   Det finns även vissa förbud som kan placeras i denna grupp. De förbjudna fördubblingarna i avsnitt **3.3.2.2** har positiv implementering i form av generering av fördubbling.

2. Negativ implementering.

   Denna implementering innebär att ett redan instansierat ackord kontrolleras mot bestämda krav, som måste vara uppfyllda. I annat fall begärs en ny instans. Hur denna skall härledas anges inte i implementeringen av regeln.

   I denna grupp återfinns främst ett antal *förbud*. Regeln för varje stämmas tillåtna omfång (avsnitt **3.3.1.6**) kan kanske sägas utgöra ett påbud, men är snarare implementerat som ett förbud i form av en kontroll att varje stämma finns inom dess givna omfång.

   Även här finns dock undantag. I avsnitt **3.3.1.7** finns ett påbud som behandlar ackordföljden D4-3. Detta påbud är implementerat negativt som en kontroll av att det gäller. Om det inte är uppfyllt backar programmet och letar efter en ny lösning.

### 6.2.8 Aritmetik och listhantering

I modulen `bas` finns ett antal regler som används vid aritmetiska beräkningar. Dessa gör det möjligt att skriva definitioner som

definitionerna i denna modul, definitionen av `inga_oktpar`, som kontrollerar att inga oktavparalleller förekommer mellan de två givna ackorden (se avsnitt **3.3.2.5 Generella förbud**).

```
inga_oktpar(par([B,T,A,S],_),par([B1,T1,A1,S1],_)) <=
       ton(B,Bn),ton(T,Tn),
       ton(A,An),ton(S,Sn),
       ton(B1,B1n),ton(T1,T1n),
       ton(A1,A1n),ton(S1,S1n),
       (oktav_paraller([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) ->
         false).
```

På sista raden används atomen `oktav_paraller/2` (sic!) som lyckas om oktavparalleller förekommer mellan ackorden.

```
oktav_paraller(X,Y) <= okp(X,Y).
```

`okp/2` är en så kallad *konstruerare*[1)], som används i regeln `par_eights_left` med följande utseende:

```
par_eights_left(okp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]),PT) <=
       number(Bn),number(Tn),
       number(An),number(Sn),
       number(B1n),number(T1n),
       number(A1n),number(S1n),
       (not(oktp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n])),
         (PT -> (I@[false|R] \- C));
       oktp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]),
         (PT -> (I@[true|R] \- C)))
       -> (I@[okp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n])|R] \- C).
```

med tillhörande provison `oktp/2` och `oktp/4`:

```
oktp([Bn,Tn,An,Sn],[B1n,T1n,A1n,S1n]) :-
       oktp(Bn,Tn,B1n,T1n);
       oktp(Bn,An,B1n,A1n);
       oktp(Bn,Sn,B1n,S1n);
       oktp(Tn,An,T1n,A1n);
       oktp(Tn,Sn,T1n,S1n);
       oktp(An,Sn,A1n,S1n).

oktp(X1,X2,Y1,Y2) :-
       Noll is ((abs((X2-X1) mod 12)) + (abs((Y2-Y1) mod 12))),
       Noll = 0,
       ((X1 \= Y1);( X2 \= Y2)).
```

1) *konstruerare*, se avsnitt **6.2.5.1**

I många fall är flera klausuler i `dubblering` tillämpbara samtidigt. De provas då i den ordning de är givna i definitionen. Detta gör att den fördubbling som generellt anses mest önskvärd för ett visst ackord provas först.

Strategin för `A \- dubblering(F1,Par,A,Val,Pos,R)` ser ut så här:

```
dubblering <=
  d_right(_,v_right(_,und_pil_false,dubblering(_))).
```

Följande hjälpstrategi hanterar kroppen till `dubblering`.

```
dubblering(C) <=
         (_ \- C).
dubblering(member(_,_)) <=
         member.
dubblering((member(_,_),identifiera_ackordton(_,_,_,_))) <=
         v_right(_,member,identifiera_ackordton).
dubblering((member(_,_),identifiera_ackordton(_,_,_,_),
  (removefirst(_,_) -> _))) <=
       v3_right(_,member,identifiera_ackordton,a_right(_,
         removefirst)).
dubblering((identifiera_ackordton(_,_,_,_),
  (removefirst(_,_) -> _))) <=
       v_right(_,identifiera_ackordton,
         a_right(_,removefirst)).
dubblering(basval(_,_,_)) <=
         dt_right(_).
```

Strategin `dubblering/0` använder sig alltså av `dubblering/1` som består av fem olika strategier för de fem olika typer av dubbleringsklausuler som finns. Den första klausulen i `dubblering/1` ger oss tillgång till den sekvent som skall visas, där `C` är den del av kroppen till `dubblering/6` som inte hanteras direkt av strategin `dubblering/0`. De olika klausulerna i `dubblering1`, förutom den första, är ömsesidigt uteslutande, vilket innebär att precis en av dessa är tillämpbar tillsammans med den första. På detta sätt kan man välja precis den strategi som är tillämplig i varje fall (se även [Fal93]). Alla dubbleringsklausuler börjar med `undantag(nagot) -> false`. Eftersom detta alltid görs, hanteras det av strategin `dubblering/0`.

### 6.2.7 Stämföringsregler

I Söderholms harmonilära finns ett antal mer eller mindre klart definierade regler för hur stämmorna får utformas, dels i förhållande till de andra stämmorna och dels med tanke på de föregående tonerna i stämman. Dessa regler finns implementerade i modulen `stamforing`, som har en central roll vid felsökning av en given sats. Det är här som de flesta kontrollstrategierna och felutskriftsreglerna finns. Vid normal simulering måste dessa regler vara uppfyllda innan ett svar kan presenteras. Vid satskontroll ges en felutskrift om en stämföringsregel inte är uppfylld. Som exempel studerar vi en av de mer komplicerade

numret på dubbleringsklausulen. Definitionen av `dubblering` består av ett stort antal klausuler. Vi studerar bara några av dem närmare.

1. Fördubbling av meloditonen i sextackord.

```
dubblering(FKN,par(fkn(F,DM,3),S1),[G,T,K],val(bas(T),[G,K]),_,1) <=
            (undantag(dubbel_meloditon,fkn(F,DM,3)) -> false),
            member(DM,[dur,moll]).
```

   Denna regel ger oss fördubbling av den ton som finns i sopranstämman, och som baston ges ackordets ters (`T`). Det andra villkoret i kroppen och 3:an i huvudet bestämmer regelns domän till alla dur- och mollackord med tersen i basstämman. Det första villkoret är av samma typ som de vi sett i föregående avsnitt och innebär att regeln är tillämpbar om inget annat sägs.

2. Fördubbling av grundtonen i kvartsextackord.

```
dubblering(F1,par(fkn(F,DM,5),S1),[G,T,K],val(bas(K),AT),_,18) <=
            (undantag(dubbel_grundton,fkn(F,DM,5)) -> false),
            member(DM,[dur,moll]),
            identifiera_ackordton(S1,[G,T],S1_ackton,_),
            (removefirst(S1_ackton,[G,G,T]) -> AT).
```

   Här får vi fördubbling av grundtonen i ett kvartsextackord. Första och andra villkoret i kroppen har samma funktion som i regeln ovan. I det tredje villkoret tar vi reda på vilken ackordton som finns i sopranstämman. Då ett kvartsextackord alltid har kvinten i basen, vet vi att vi måste ha två förekomster av grundtonen bland sopran-, alt- och tenorstämmorna för att få grundtonen fördubblad. Genom att i fjärde villkoret plocka bort ur en lista med två grundtoner och en ters den ackordton som sopranstämman har, uppnås precis detta. Notera att denna fördubbling är förbjuden, d.v.s. det finns definitioner sådana att första villkoret i kroppen misslyckas. Det är dock möjligt för användaren att tillåta den.

3. Ackord med fyra toner utan fördubbling.

```
dubblering(FKN,par(fkn(F,Typ,1),S1),[G,T,K,S],val(bas(G),AT,_,11) <=
            (undantag(odubblerat,fkn(F,Typ,1)) -> false),
            identifiera_ackordton(S1,[T,K,S],S1_ackton,_),
            (removefirst(S1_ackton,[T,K,S]) -> AT).
```

   Denna regel tar hand om alla ackord bestående av fyra toner (t. ex. D7) som har grundtonen (`G`) i basen. I andra villkoret tar vi reda på vilken ackordton som finns i sopranstämman och i tredje villkoret plockas den bort ur listan `Kvar` som innehåller ackordtonerna för alt och tenor. På detta sätt fördubblas ingen av ackordets toner.

I kbs.def finns en stor mängd undantag specificerade. Dessa kan i många fall plockas bort av användaren och det är även möjligt att definiera nya undantag. Sista klausulen i baston/6 som placerar ut bastonen i ett ackord ser till exempel ut så här:

```
baston(FKN1,FKN2,_,B,Bas,B1) <=
       (undantag(bas(FKN1,FKN2)) - false),
       lamplig_oktav(Bas,B,B1).
```

Detta läses ”om vi inte har något undantag från det allmänna fallet så ges den nya bastonen B1 av lamplig_oktav”. Men vissa undantag finns och ges i form av definitioner av undantag för dessa fall.

```
undantag(bas(fkn(F,dur,3),fkn(F,dur,1))).
undantag(bas(fkn(F,moll,3),fkn(F,moll,1))).
...
```

De två klausulerna ovan innebär att det inte är tillåtet att använda den sista generella klausulen i baston/6 då man går från sextackord till grundläge (se avsnitt **3.3.1.7 Stämföringspåbud**).

En sekvent på formen

```
| ?- A \- undantag(nagot) -> false.
```

beräknas av följande strategi, som lyckas om undantag(nagot) inte är definierad.

```
und_pil_false <=
       a_right(_,d_left(_,_,false_left(_))).
```

#### 6.2.6.3 Fördubbling

Valet av fördubbling sköts av dubblering/6. Situationen vid anrop av denna är att man vet vilken ton sopranstämman har och vilka toner som kan ingå i ackordet. Det gäller nu att avgöra vilken av dessa toner som skall placeras i basstämman och vilka toner som finns att välja på för alt- och tenorstämmorna. Huvudet till dubblering ser ut så här:

```
dubblering(F1,par(F2,S1),Atoner,val(bas(B),Vtoner),P,R)
```

Här är F1 funktionen hos föregående ackord, F2 funktionen hos det ackord vi skall välja fördubbling för, S1 sopranstämman i samma ackord, Atoner de toner som kan finnas i det ackord vi skall välja fördubbling för, B valet av baston och Vtoner de toner som finns att välja på för alt- och tenorstämmorna. P och R är en intern positionsangivelse respektive

```
| ?- hitta_par1 \\- hitta_par1([[g,[e,g]],[d,[e,g]]]) \- L.

L = [[g,g],[d,e]] ? ;

L = [[g,e],[d,g]] ? ;

no
```

### 6.2.6 Kunskapsbas

Modulen `kbs` innehåller dels diverse uppräkningar, exempelvis av vilka intervall som är tillåtna inom en stämma, och dels definitionen av en atom `dubblering/6` som väljer hur tonerna i ett ackord skall fördubblas. Detta är den enda del av programmet som behöver ändras om man vill utöka programmet med flera ackordfunktioner, exempelvis tonika-parallell.

#### 6.2.6.1 Intern representation av musikaliska funktioner

Användaren anger musikaliska funktioner på ett sätt som försöker efterlikna Söderholm, tonikan T skrivs `f_T`, tonika i tersläge T3 skrivs `f_T3` o.s.v. Internt har det dock visat sig vara mer praktiskt att parametrisera detta. En musikalisk funktion representeras därför av `fkn(Funk,Typ,Baston)` där `Funk` anger funktion (`fT`, `fS` eller `fD`), `Typ` anger ackordtyp (`dur`, `moll` o.s.v.) och `Baston` anger vilken ackordton som återfinns i basstämman. Ett tonika-ackord i dur, med tersen i basen, representeras då av `fkn(fT,dur,3)`. I `kbs.def` finns översättningen mellan intern och extern representation definierad enligt följande:

```
till_fkn(f_T,fkn(fT,dur,1)).
till_fkn(f_oT,fkn(fT,moll,1)).
till_fkn(f_T3,fkn(fT,dur,3)).
...
```

#### 6.2.6.2 Undantag från mönstret

För att hantera situationer av typen ”detta gäller om inget annat sägs” finns en mängd undantag från den normala situationen uppräknade. Det typiska testet är

```
undantag(nagot) -> false
```

vilket misslyckas om det finns en klausul

```
undantag(nagot).
```

Exempel:

```
| ?- lampliga_oktaver \\- lampliga_oktaver([g1,d1],[e,g]) \- L.

L = [[g1,[e1,g1]],[d1,[e1,g1]]] ? ;

L = [[g1,[e1,g1]],[d1,[e1,g]]] ? ;

L = [[g1,[e1,g1]],[d1,[e,g1]]] ?

yes
```

Resultatet från `lampliga_oktaver` utnyttjas av `hitta_par1`.

```
hitta_par1([]) <=
       [].
hitta_par1([[T,Valtoner]|Xs]) <=
       ((narn(T,Valtoner) -> [Narmast_T]),
       (hitta_par1(tag_bort(Narmast_T,Xs)) -> Resten))
       -> [[T,Narmast_T]|Resten].                    % Svar
hitta_par1(X)#{X\=[],X\={[_|_]} <=
       (X -> Y)                          % Reducera argumentet
       -> hitta_par1(Y).
```

Precis som i definitionen av `identifiera_ackordton` har vi här en klausul som hanterar icke-evaluerade argument. Den intressanta klausulen är återigen den andra. `Narm` hittar den ton i `Valtoner` som ligger närmast `T`. Denna ton `Narmast_T` är sedan förbrukad och en oktavförekomst av den avlägsnas av `tag_bort` vid det rekursiva anropet till `hitta_par1`. Vi illustrerar med några exempel.

```
| ?- narm \\- narm(g,[e,g]) \- [Narmast_T].

Narmast_T = g ? ;

Narmast_T = e ? ;

no
```

```
| ?- tag_bort \\- tag_bort(g,[[d,[e,g]]]) \- L.

L = [[d,[e]]] ? ;

no
```

```
| ?- hitta_par1 \\- hitta_par1([[d,[e]]]) \- Resten.

Resten = [[d,e]] ? ;

no
```

#### 6.2.5.3 `hitta_par/3`

`hitta_par/3` används för att placera ut stämmor. Definitionen, som hör till de mer kryptiska i vårt program, ser ut så här:

```
hitta_par([],_,[]).
hitta_par([X|Xs],Ackordtoner,Svar) <=
       hitta_par1(lampliga_oktaver([X|Xs],Ackordtoner)) ->
         Svar.
hitta_par([A,T],Tonlista,[[A,A1,[T,T1]]) <=
       identifiera_ackordton(A1,Tonlista,_,_),
       identifiera_ackordton(T1,Tonlista,_,_).
```

Den tredje klausulen används endast då alla variabler är instansierade och finns endast med av effektivitetsskäl. Resultatet `Svar` är en lista på formen `[[T1,F1],[T2,F2],...]` där `Ti` är en stämma i föregående ackord och `Fi` är den ton som stämman får i nästa ackord. Tonerna `Ti` är de toner som finns i första argumentet och tonerna `Fi` är oktavförekomster av tonerna i `Ackordtoner`. En ton i `Ackordtoner` kan inte associeras till mer än en ton i listan `[X|Xs]`. `hitta_par` ger alla möjliga svar (ett möjligt svar är ett svar där avståndet mellan `Ti` och `Fi` inte är större än en oktav för något `i`) i följande ordning

a) Som första `Svar` ges det som innebär minsta möjliga förflyttning av stämmorna.

b) Därpå ges de `Svar` som visserligen innebär större förflyttning av stämmorna, men där en stämmas nya ackordton ligger så nära föregående ton som möjligt.

c) Därefter ges övriga svar i godtycklig ordning.

Exempel:

Om vi har `f1` i altstämman och `a` i tenorstämman och skall placera ut tonerna e och g i dessa stämmor i följande ackord, så kommer som första `Svar` att ges `[[f1,e1],[a,g]]` och som andra `Svar` ges `[[f1,g1],[a,e]]`. Låt oss se hur detta går till. Den andra klausulen i `hitta_par` är den intressanta:

```
hitta_par([X|Xs],Ackordtoner,Svar) <=
       hitta_par1(lampliga_oktaver([X|Xs],Ackordtoner)) ->
         Svar.
```

Här ser vi ett anrop till funktionen `lampliga_oktaver`. Denna har som argument två listor av toner. Som resultat ges en lista på formen `[[X1,[A11,...,A1m]],...,[Xn,[An1,...,Anm]]`, där `Xi` är tonerna i första argumentet och `Aij` är oktavförekomster av tonerna i `Ackordtoner`. Första svaret till `lampliga_oktaver` placerar tonerna `Ai1,...,Aim` så nära `Xi` som möjligt. Vid backtracking ges fler svar.

#### 6.2.5.2 `identifiera_ackordton/4`

Ibland vill man veta om en ton tillhör ett visst ackord eller i vilken oktav den förekommer i ackordet. Till detta används `identifiera_ackordton/4`.

Definition:

```
identifiera_ackordton(Ton,Tonlista,Svar,_) <=
       ton(Ton,Nton),
       member(Svar,Tonlista),
       ton(Svar,Nsvar),
       oktav_avst(Nton,Nsvar).
identifiera_ackordton(Ton,T,Svar,Pos)#{T \= [], T \= [_|_]} <=
       (T -> Tonlista),
       identifiera_ackordton(Ton,Tonlista,Svar,Pos).
```

Det fjärde argumentet är en intern positionsangivelse. Den andra klausulen används då andra argumentet inte är evaluerat. För att göra de båda klausulerna ömsesidigt uteslutande finns en så kallad *vakt* vid den andra klausulen. `identifiera_ackordton` fungerar helt enkelt så att man kontrollerar att `Svar` finns med i `Tonlista` och befinner sig på någon form av oktavavstånd till `Ton`.

Strategi:

```
identifiera_ackordton <=
       d_right(_,v4_right(_,ton(_),member,     % 1:a klausulen
         ton(_),oktav_avst_right(_))),
       d_right(_,v_right(_,                    % 2:a klausulen
         a_right(_,ia_reduce),identifiera_ackordton)).

ia_reduce <=
       removefirst.
```

Vi har en strategi för varje klausul. I den andra strategin finns ett anrop till strategin `ia_reduce` som består av alla sätt att reducera andra argumentet till en lista (här endast *ett* sätt). Om man t. ex. skulle vilja använda `identifiera_ackordton` så här

```
identifiera_ackordton(g,sort([e1,g,c2]),Y).
```

där `sort` sorterar sitt argument, skulle man endast behöva utöka `ia_reduce` till

```
ia_reduce <= removefirst, sort.
```

förutsatt att man har en strategi `sort` för att utföra sorteringen.

Vi ser att `oktav_avst` här ersätts med `true` eller `false` i försöket att visa regelns sista rad (dess slutsats). Strategin för `lamplig_oktav/3` kan se ut så här:

```
lamplig_oktav <=
       d_right(_,v4_right(_,ton(_),ton(_),a_right(_,
         oktav_avst_left(_,false_left(_))),lamplig_oktav1)),
       d_right(_,v4_right(_,ton(_),ton(_),
         oktav_avst_right(_),lamplig_oktav2)).
```

Det finns alltså en strategi för varje klausul, där vi använder `oktav_avst_right` eller `oktav_avst_left` beroende på situationen. Denna strategi har nackdelen att man, på grund av likheten mellan de båda fallen, hinner utföra mycket arbete innan man kommer på att man skulle ha valt det andra fallet. I den nuvarande implementeringen är den därför utbytt mot en strategi som väljer rätt delstrategi direkt (se `bygg.rul` och den likartade strategin `dubblering` i avsnitt **6.2.6.3**).

Definition av `lamplig_oktav1/3`:

```
lamplig_oktav1(Ton,Jmfr,Val) <=
       str_over(Ton,Jmfr),
       laga_tonen(Ton,ren,oktav,ner,Tonner),
       omringa_ner(Tonner,Jmfr,Ton,Lag,Hog),
       mest_nara(Jmfr,Lag,Hog,Val).
lamplig_oktav1(Ton,Jmfr,Val) <=
       str_over(Jmfr,Ton),
       hoga_tonen(Ton,ren,oktav,ner,Tonupp),
       omringa_upp(Ton,Jmfr,Tonupp,Lag,Hog),
       mest_nara(Jmfr,Lag,Hog,Val).
```

Här omringas `Jmfr` av två förekomster av `Ton` (`Lag` och `Hog`), vilka befinner sig en ren oktav ifrån varandra. `mest_nara` ger som första `Val` den av dessa två som är närmast `Jmfr` räknat i antal halvtonsteg. Som andra `Val` ges den andra (se modulen `ton`). Strategin för `lamplig_oktav1` ser ut så här:

```
lamplig_oktav1 <=
       d_right(_,v4_right(_,under,hoga_tonen,
         omringa,mest_nara)).
```

Notera hur denna speglar definitionen (samma strategi används till båda klausulerna). Vi tar inte upp `lamplig_oktav2` här utan påpekar bara att denna kan ge tre olika `Val` vid backtracking (`Jmfr` samt tonerna en oktav upp och en oktav ner från denna).

### 6.2.5 Systemets kärna

Modulen `bygg` innehåller bland annat den del av programmet som bestämmer hur tonerna i ett ackord skall placeras. Vi beskriver implementeringen av några nyckelkoncept.

#### 6.2.5.1 `lamplig_oktav/3`

Definition (finns i filen `bygg.def`):

```
lamplig_oktav(Ton,Jmfr,Val) <=
       ton(Ton,Nton),
       ton(Jmfr,Njmfr),
       (oktav_avst(Nton,Njmfr) -> false),
       lamplig_oktav1(Ton,Jmfr,Val).
lamplig_oktav(Ton,Jmfr,Val) <=
       ton(Ton,Nton),
       ton(Jmfr,Njfr),
       oktav_avst(Nton,Njfr),
       lamplig_oktav2(Jmfr,Val).
```

Givet en ton `Jmfr` från ett tidigare ackord blir `Val` den förekomst av `Ton` som ligger närmast `Jmfr`. Vid backtracking ges även den näst närmaste förekomsten. Låt oss se hur detta åstadkommes.

Först tar man reda på tonernas interna nummer `Nton` respektive `Njmfr`. Därefter provas om avståndet mellan dessa är någon form av oktavavstånd. Detta gör de båda klausulerna ömsesidigt uteslutande. `oktav_avst/2` är en så kallad *konstruerare*. Detta innebär att den inte har någon definition, utan hanteras av en speciell härledningsregel som återfinns i filen `bygg.rul`. Eftersom `oktav_avst/2` under byggandet av härledningar förekommer både till höger och vänster om deduktionssymbolen ‘`\-`’, finns det två härledningsregler:

```
oktav_avst_right(oktav_avst(X1,X2)) <=
       number(X1),
       number(X2),
       Noll is abs((X2-X1) mod 12),
       Noll = 0
       -> (_ \- oktav_avst(X1,X2)).
```

Detta innebär att `oktav_avst` till höger om symbolen ‘`\-`’ lyckas då avståndet mellan `X1` och `X2` är en multipel av 12.

```
oktav_avst_left(oktav_avst(X1,X2),PT) <=
       number(X1),
       number(X2),
       Noll is abs((X2-X1) mod 12),
       (Noll = 0,(PT -> (I@[true|R] \- C));
       Noll \= 0,(PT -> (I@[false|R] \- C)))
       -> (I@[oktav_avst(X1,X2)|R] \- C).
```

och när inga fler anrop till `pd_left` eller `v_left` är möjliga provas strategin `remadd`. För att förstå denna tittar vi först på en klausul i definitionen av `forbjud`:

```
forbjud(ackord([B,T,A,S]),pos(N)) <=
       add_def(undantag(ackord([B,T,A,S]),pos(N)),true).
```

Vad som sker här är att man lägger till en klausul till definitionen. För detta används regeln `add_def` som i likhet med `add_l` nedan finns i standardfilen rules.rul. Där finns även motsvarigheter `rem_def` och `rem_l` som används till att ta bort delar av definitionen. Låt oss återgå till strategin `remadd`.

```
remadd(Satstyp) <=
       rem_l(_,_,remadd(Satstyp)),
       add_l(_,_,remadd(Satstyp)),
       SatsTyp.
```

Här tar man först hand om alla `rem_def` och `add_def` genom att rekursivt anropa `remadd`. När detta är klart fortsätter man med strategin `SatsTyp` som anger om man skall fortsätta med arrangering, satskontroll eller funktionsanalys, utifrån den förändrade definitionen.

#### 6.2.4.1 Att gå igenom en sats

En sats representeras av en lista av `vert(Funktion,[Bas,Tenor,Alt,Sopran])` samt dess `Tonart`. Vid såväl simulering som analys och kontroll gås satsen igenom från början till slutet, med backtracking om så behövs. Vid kontroll och analys är dock backtracking starkt begränsad. Vi tittar som hastigast på `ga_igenom` som sköter själva genomlöpningen av satsen.

```
ga_igenom(Tonart,[vert(F1,T1,pos(N)),vert(F2,T2)]) <=
       ((N+1 -> N1),
       harmo(Tonart,vert(F1,T1,pos(N),vert(F2,T2,pos(N1)),
         vert(tom,ton),Regel))
       -> [T1,T2].
ga_igenom(Tonart,[vert(F1,T1,pos(N)),vert(F2,T2),
  vert(F3,T3)|Xs]) <=
       ((N+1 -> N1),
       harmo(Tonart,vert(F1,T1,pos(N)),vert(F2,T2,pos(N1)),
         vert(F3,T3),Regel),
       (ga_igenom(Tonart,[Tonart,[vert(F2,T2,pos(N1)),
         vert(F3,T3)|Xs) -> Resten))
       -> [T1|Resten].
```

Vi ser att varje `vert` som användaren givit utökas med en positionsangivelse `pos(_)` som används vid kontroll och analys för att tala om i vilket ackord ett fel finns. Det ackord som kontrolleras/instansieras är andra elementet i listan av vertikaler, `vert(F2,T2)`. Själva beskrivningen av hur satsen skall vara beskaffad finns i `harmo`. Notera att `harmo` i varje steg tar hänsyn till närmast föregående och efterföljande ackord.

```
skala(Grundton,Skaltyp)#{Skaltyp \= mel_moll} <=
       ityper(Skaltyp,Ityper)
       -> iskala(Ityper,Grundton).
skala(Grundton,mel_moll) <=
       ityper(mel_moll,Ityper1),
       ityper(ren_moll,Ityper2),
       (iskala(Ityper1,Grundton)->[U1,U2,U3,U4,U5,U6,U7,U8]),
       (iskala(Ityper2,Grundton)->[N1,N2,N3,N4,N5,N6,N7,N8])
       -> [U1,U2,U3,U4,U5,U6,U7,U8,N7,N6,N5,N4,N3,N2,N1].
```

Modulen är implementerad funktionellt, vilket har medfört vissa problem vid försöken att införa kontrollstrategier (se avsnitt **7.4 Kontrollstrategier och funktionella definitioner**).

Definitionen av `skala` har ett specialfall för melodisk moll, en skala som är olika i uppåt- och nedåtgående och därför blir längre än en ”normal” skala ([Söd59] sid 8, ex 8). Mot slutet av programutvecklingsfasen har vi dock implementerat en rekursiv variant av `skala`. Den kallas `r_skala` och har fördelen att den fungerar på alla längder av skalor.

### 6.2.4 Huvudprogrammet

I modulen `topp` finns de definitioner och regler som får det hela att snurra. En typisk fråga vid simulering har formen

```
simulering \\- forbjud(nagot),...,tillat(nagotannat),...,
  sats(Ta,[vert(F1,L1),...,vert(Fn,Ln)]) \- Svar.
```

där `tillat` och `forbjud` ändrar programmets beteende och `sats` är en beskrivning av den sats man vill simulera. Vårt första problem är nu att garantera att ändringarna av programmets beteende (`tillat`, `forbjud`) utförs innan man börjar arbeta på `sats`. Detta löses av strategin `simulering` på följande sätt (strategierna `analys` och `kontroll` fungerar likadant):

```
simulering <=
       start(sats(identifiera_ackordton,dubblering,harmoS)).

start(SatsTyp)<=
       pd_left(_,_,start(SatsTyp)),
       v_left(_,_,start(SatsTyp)),
       remadd(SatsTyp).
```

Argumentet `SatsTyp` specificerar hur man skall fortsätta i `sats`. Detta varierar för strategierna `simulering`, `analys` och `kontroll`. I strategin `start` fungerar regeln `pd_left` som den fördefinierade regeln `d_left`, förutom restriktionen att den inte är tillämpbar på `sats`. Effekten av detta blir att man först kommer att prova `pd_left` på allting utom `sats`

någon typ av sext-intervall. Vi får också talet 9, som är antalet halvtonsteg för en stor sext. Med hjälp av `dist` fås antalet halvtonsteg mellan de aktuella tonerna, som i detta fall beräknas till 52 - 45 = 7. Ur subtraktionen 7 - 9 (`D - Stamdist`) får vi talet -2. Uppslagning av `dist_intervall(-2,stor,Ityp)` ger resultatet `Ityp = forminskad`, och sålunda får vi svaret

```
X = forminskad,
Y = sext ?
```

Av effektivitetsskäl finns det definitioner även för omvändningen. Dessa bestämmer vilken ton som ligger på ett visst intervall från en given ton. Det finns två sådana definitioner - en för beräkning av den högre tonen och en för den lägre. Dessa toner går givetvis bra att beräkna med hjälp av `intervall`, men det kräver alltför mycket sökning. Definitionen för den förra ser ut så här:

```
hoga_tonen(Lag,Ityp,Int,Hog) <=
       stam_intervall(VD,R_S,Int,Stamdist),
       dist_intervall(N,R_S,Ityp),
       (N + Stamdist -> D),
       stamton(Lag,S1),
       vit(S1,VN1),
       (VN1 + VD -> VN2),
       vit(S2,VN2),
       distton(Lag,D,Hog),
       stamton(Hog,S2).
```

Som synes överensstämmer denna definition ganska väl med definitionen av `intervall`. Den största skillnaden består i att de aritmetiska operationerna utförs explicit innan `stam_intervall` respektive `dist_intervall` anropas. Skälet till detta är att *strategierna* på så sätt blir enklare och effektivare. I definitionen av `intervall` har hänsyn tagits till enkelheten hos *definitionen* snarare än hos strategierna, då denna inte används lika ofta i programmet.

### 6.2.2 Ackord

Vid simulering finns vanligtvis ackordfunktionerna givna. Modulen `ackord` innehåller en atom `ackord/3` som tillhandahåller de ingående tonerna i givet ackord. I denna modul finns fler ackordtyper implementerade än de som för närvarande kan hanteras på högre nivå.

### 6.2.3 Skalor

Som en fristående del av programmet har vi implementerat en modul `skala` för hantering av skalor. Den har ingen funktion att fylla i samband med arrangering och satskontroll, utan kan ses som ett komplement till huvudprogrammet. Definitionen av `skala/2` ser ut så här:

```
...
alter(a,a,34).
alter(a,'a#',35).
...
```

som ger en intern talrepresentation av tonerna. Där anges dessutom stamtonen till respektive ton ([Söd59] sid 7). Även stamtonerna representeras internt av tal. Dessa anges i definitionen av `vit`. (Namnet kommer av att stamtonerna är de vita tangenterna på ett vanligt piano.)

```
...
vit(a,20).
vit(h,21).
...
```

Det finns två olika musikaliska avståndsbegrepp i systemet - halvtonsteg och intervall.

Antalet halvtonsteg mellan två toner är detsamma som antalet tangenter mellan dem på ett piano. Det finns klausuler för att beräkna detta avstånd, liksom för att bestämma vilken ton som ligger på ett visst avstånd från en given ton. Dessutom kan man avgöra vilken av två toner som klingar högst (har högst frekvens). Härvid ses till exempel diss och ess som lika höga toner - trots att de egentligen skiljer sig en aning i frekvens.

Intervallet mellan två toner motsvarar antalet steg i notsystemet. Hur beräkningen av intervallet mellan två toner utförs, illustrerar vi med ett exempel. Antag att vi vill beräkna intervallet mellan ettstrukna giss och tvåstrukna ess. I modulen `ton` hittar vi definitionen av `intervall` och tillhörande strategi. Följande fråga utför beräkningen:

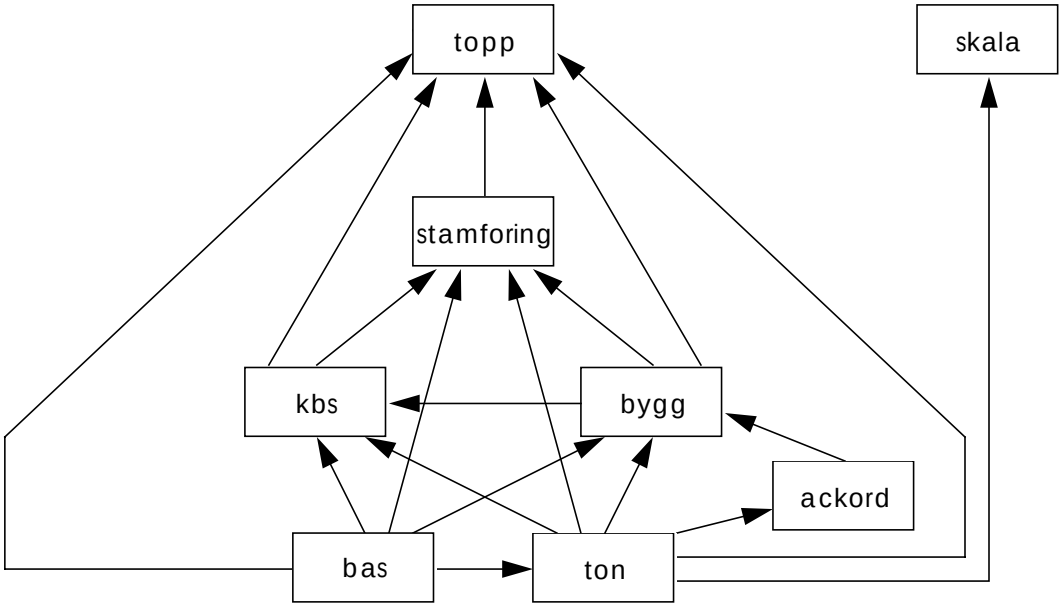
```
intervall \\- \- intervall('g1#',eb2,X,Y).
```

Låt oss studera hur detta sker. Definitionen av `intervall` ser ut så här:

```
intervall(Lag,Hog,Ityp,Int) <=
       stamton(Lag,S1),
       stamton(Hog,S2),
       vit(S1,VN1),
       vit(S2,VN2),
       stam_intervall(VN2 - VN1,R_S,Int,Stamdist),
       dist(Lag,Hog,D),
       dist_intervall(D - Stamdist,R_S,Ityp).
```

Först slås stamtonerna upp - för `g1#` får vi `g1` och för `e2b` fås `e2`. Därefter går vi till uppräkningen `vit` för att ta reda på dessas stamtonsnummer, som i detta fall är 26 respektive 31. Subtraktion (`VN2 - VN1`) ger talet 5, som är antalet steg i notsystemet mellan dessa stamtoner. En uppslagning i tabellen `stam_intervall` ger vid handen att vi här står inför

Figuren nedan visar programmets struktur.




De olika modulerna interagerar som synes flitigt med varandra. Pilarna i figuren anger ”export” av definitioner, härledningsregler och sökstrategier. Exempelvis har `bygg` tillgång till definitioner, regler och strategier i modulen `ackord`, som i sin tur använder sig av modulen `ton`. Förutom modulerna i figuren finns två filer med allmänna regler som används av hela programmet - dels den fördefinierade regelsamlingen rules.rul, dels en fil grundregler.rul med bland annat vissa utökningar och modifieringar av standardreglerna (se avsnitt **6.2.9 Grundregler**). Dessutom finns specialregler i vissa moduler, exempelvis `oktav_avst_right` i filen `bygg`.rul, samt felutskriftsregler som används vid satskontroll.

Programkoden finns i **Appendix A**.

## 6.2 Simulering

Vid simulering är satsens tonart, musikaliska funktioner och vanligen även melodistämman givna. Problemet är nu att hitta övriga toner så att man får en sats som uppfyller de krav i [Söd59] som vi implementerat. Då såväl simulering som satskontroll och funktionsanalys utnyttjar samma definition, är beskrivningen av denna spridd över avsnitten **6.2**, **6.3** och **6.4**.

### 6.2.1 Toner och intervall

I vår implementering betraktas toner vanligen som givna atomer. En ton benämnes med sitt namn, exempelvis får tonen lilla g heta ‘`g`’ och tonen ettstrukna ciss benämnes ‘`c1#`’.

Tonerna är systemets mest grundläggande byggstenar. Hanteringen av dessa sker i modulen `ton`. Tonerna finns uppräknade i definitionen av `alter`

vilket innebär att de flesta strategier inleds med `d_right` eller `d_left` för att komma ner till definitionen av `namn`.

Antag att definitionen av `namn` innehåller en atom ur en annan modul `mod`. När man skriver strategier för `namn` behöver man bara se till att man får en sekvent som är på samma form som den i modulhuvudet i `mod.def`, varefter man kan ”plugga in” den strategi som anges där. Om vi har huvudet i `mod.def`

```
% mod_def1 \\- A \- mod_def1(Arg1,Arg2).
% mod_def2 \\- A \- mod_def2(Arg).
```

och använder atomerna till följande definition

```
anvander_modul(X,Y,Z) <=
       mod_def2(Y),
       mod_def1(X,Z).
```

vilken vi använder i frågan

```
| ?- anvander_modul \\- \- anvander_modul(gcla,modul,K).
```

så skriver vi mycket enkelt strategin `anvander_modul`

```
anvander_modul <=
       d_right(_,v_right(_,mod_def2,mod_def1)).
```

som inte påverkas av eventuella ändringar i modulen `mod`.

Det finns tre typer av strategier: för simulering (*strategier*), för satskontroll (*kontrollstrategier*) och för funktionsanalys (*analysstrategier*). Gemensamt för alla strategier är att de bortser från de eventuella övriga förutsättningar i en fråga, som är ointressanta ur dess synvinkel. Ingen hänsyn tas alltså till antecedenten `A` i följande fråga:

```
| ?- namn \\- A \- namn(...).
```

De regler och provison, som används allmänt av hela programmet, finns i filerna `rules.rul` (fördefinierade regler och strategier) och `grundregler.rul`. Dessutom finns specialregler i vissa moduler, exempelvis `oktav_avst_right` i filen `bygg.rul`, samt felutskriftsregler som används vid satskontroll.

# 6 Implementering av systemet

## 6.1 Modularisering

På grund av programmets storlek har vi gjort en indelning i moduler. Avsikten med detta har varit densamma som i andra programmeringsspråk; att isolera programdelar ifrån varandra och göra det möjligt för flera programmerare att arbeta på ett gemensamt projekt utan att behöva känna till detaljer om varandras delar. Eftersom ett GCLA-program består av två delar, *objekt-* och *metanivå*, måste varje modul innehålla båda dessa delar för att ge en användbar och utåt sluten enhet. Varje modul `mod` innehåller en definitionsfil `mod.def` och en regelfil `mod.rul` med bland annat strategier till motsvarande definitioner och modulspecifika härledningsregler. (En av modulerna, `grundregler`, innehåller dock endast en regelfil.). Tanken är nu att användaren av en atom inte skall behöva veta något annat om denna än dess typiska användning. Inga ändringar av definitioner och regler inom modulen skall märkas utåt.

I varje definitionsfil finns ett huvud med typexempel på användningen av de definitioner som anses tillgängliga utåt. Huvudet i filen `bygg.def` ser ut så här:

```
% Strategierna finns i filen 'bygg.rul'.

% Definitioner som anvands utat:
%
% lamplig_oktav \\- A \- lamplig_oktav(T1,T2,T3).
% omringa \\- A \- omringa_ner(T1,T2,T3,T4,T5).
% omringa \\- A \- omringa_upp(T1,T2,T3,T4,T5).
% ackordtoner \\- A,ackordtoner(Ta,F) \- At.
% funktions_grundton \\- A \- funktionsgrundton(Ta,F,T).
% identiska \\- A \- identiska(X1,X2).
% identifiera_ackordton \\- A \-identifiera_ackordton(T1,L,T2).
% hitta_par \\- A \- hitta_par(L1,L2,L3).
% narmast \\- A \- narmast(T1,L,T2,P).
%
% A star for godtycklig antecedent. Ingen hansyn tas till denna.
```

Regelfilerna innehåller strategier, regler och provison för motsvarande definitioner. Huvudprincipen för strategierna är att en strategi med namnet `namn` är tillämpbar på en instans av atomen `namn`. En typisk användning av strategin är därför

```
| ?- namn \\- \- namn(...).
```

eller

```
| ?- namn \\- namn(...) \- Svar.
```

### 5.2.3 Algoritm *stämföring_OKK*

Går igenom samma krav som ställs upp av algoritm *stämföring_OK*, men om ett krav inte är uppfyllt avbryts inte kontrollen, utan ett felmeddelande skrivs ut. Kontrollen fortsätter därefter med nästa krav.

## 5.3 Funktionsanalys

Vid funktionsanalys är alla toner i satsen givna, men vi vet inte vilka funktioner ackorden har. Det gäller alltså att utnyttja kunskapen om vad en korrekt sats är för att finna ackordens funktioner.

### 5.3.1 Algoritm *funktionsanalys*

Givet en sats, sats(Tonart, [vertikal($F_1$,$Ackord_1$),...,vertikal($F_n$,$Ackord_n$)]), löp igenom listan av vertikaler och instansiera i varje steg en funktion $F_i$ enligt algoritm *harmoA*.

### 5.3.2 Algoritm *harmoA*

Givet satsens Tonart och tre vertikaler, vertikal($F_1$,<$B_1$,$T_1$,$A_1$,$S_1$>), vertikal($F_2$, <$B_2$,$T_2$,$A_2$,$S_2$>) och vertikal($F_3$,L), där $F_1$ och samtliga toner är instansierade, gör följande i tur och ordning för att hitta $F_2$.

1. Instansiera $F_2$ till någon funktion.
2. Slå upp ackordtonerna Acktoner till denna.
3. Kontrollera med algoritm *närmast* att $S_2$ ingår bland Acktoner, backa annars och instansiera $F_2$ till något annat.
4. Kontrollera att det finns en tillåten fördubbling som kan leda till <$B_2$,$T_2$,$A_2$,$S_2$>, backa annars och instansiera $F_2$ till något annat.
5. Kontrollera att det går att få <$B_2$,$T_2$,$A_2$,$S_2$> från denna fördubbling, backa annars och instansiera $F_2$ till något annat.
6. Om man kommer hit har man hittat rätt $F_2$. Kontrollera att stämföringen är korrekt med *stämföring_OKK*.

Med den lilla delmängd av alla möjliga funktioner vi har finns bara ett möjligt svar vid funktionsanalys (med undantag för funktionerna D64 och T5 som innehåller samma toner). Detta gäller dock inte om man utökar mängden av möjliga funktioner.

### 5.1.6 Algoritm *undantag*

I många fall gäller att ett visst ackord, en viss ackordföljd eller liknande är tillåten så länge inget annat anges av användaren. Detta hanteras så, att allting som inte går att härleda i systemet antas vara falskt, även sådant som är odefinierat (*Closed World Assumption*). Algoritm *undantag* lyckas om det går att visa ¬ undantag(argument), vilket lyckas om undantag(argument) är odefinierat eller om man kan visa att undantag(argument) är falskt. Användaren har möjlighet att styra när algoritm *undantag* lyckas genom att ändra definition-en. Systemet är därför icke-monotont, då en härledning inte alltid fortsätter att vara giltig om nya fakta tillkommer.

## 5.2 Satskontroll

Vid satskontroll förväntas alla satsens toner och ackordfunktioner vara givna och man vill veta om det finns några fel i satsen. För att kontrollera en given sats används följande algoritm.

### 5.2.1 Algoritm *satskontroll*

Givet en sats, sats(Tonart, [$V_1$,...,$V_n$]), löp igenom listan av vertikaler och kontrollera med algoritm *harmoK* att varje ackord är korrekt, dels som egen enhet och dels i förhållande till föregående ackord. Vid satskontroll backar man aldrig till ett tidigare ackord. Alla fel skrivs ut då de hittas.

### 5.2.2 Algoritm *harmoK*

Givet en tonart och tre vertikaler, vertikal($F_1$,<$B_1$,$T_1$,$A_1$,$S_1$>), vertikal($F_2$,<$B_2$,$T_2$,$A_2$,$S_2$>) och vertikal($F_3$,L), där den första är kontrollerad, gör följande i tur och ordning för att kontrollera ackordet <$B_2$,$T_2$,$A_2$,$S_2$> och förbindelsen mellan den första och andra vertikalen.

1. Kontrollera att sopranstämman $S_2$ ingår i det ackord som ges av $F_2$ och tonarten. Om den inte gör det - skriv ut felmeddelande.
2. Kontrollera att fördubblingen av andra ackordet är tillåten genom att försöka generera den med algoritm *dubblering*. Om det inte går - skriv ut felmeddelande.
3. Kontrollera att övriga ackordtoner ingår i ackordet. I annat fall - skriv ut felmeddelande.
4. Kontrollera övriga stämföringskrav med algoritm *stämföring_OKK*.

Algoritmen *harmoK* lyckas alltid.

### 5.1.3 Algoritm *närmast*

Algoritmen *närmast* räknar upp alla möjliga fortsättningar, med början från den som innebär minsta möjliga förflyttning. För närmare beskrivning se avsnitten **6.2.5.3** och **3.3.3.2**.

### 5.1.4 Algoritm *dubblering*

Algoritmen *dubblering* ger alla tillåtna fördubblingar av det ackord som skall instansieras, i den ordning som anses bäst. För närmare beskrivning se avsnitten **3.3.3.1** och **6.2.6.3**.

### 5.1.5 Algoritm *stämföring_OK*

Givet tonart, föregående ackord $<B_1,T_1,A_1,S_1>$ och dess funktion $F_1$, samt det nyinstansierade ackordet $<B_2,T_2,A_2,S_2>$ och dess funktion $F_2$, går algoritm *stämföring_OK* steg för steg igenom följande lista av krav, vilka alla måste vara uppfyllda. Om ett krav inte är uppfyllt så provas inte de återstående, då man redan vet att stämföringen är felaktig. ($\texttt{Ton}_1$ **op** $\texttt{Ton}_2$ opererar på tonernas frekvenser.)

1. Kontrollera med algoritm *undantag a*tt ackordet $<B_2,T_2,A_2,S_2>$ inte är förbjudet av användaren.
2. Kontrollera att eventuella speciella stämföringskrav är uppfyllda (se `spec_sf_krav` i filen `stamforing.def`).
3. Kontrollera att ingen stämkorsning förekommer, d.v.s. försök visa att

   $B_2 \le T_2 \le A_2 \le S_2$
4. Kontrollera att stämmornas omfång är tillåtna, d.v.s. försök visa

   tillåtet_omfång(bas,$B_2$) ∧ tillåtet_omfång(tenor,$T_2$) ∧
   tillåtet_omfång(alt,$A_2$) ∧ tillåtet_omfång(sopran,$S_2$)

   där definitionen av tillåtet_omfång är den som ges i avsnitt **3.3.1.6**.
5. Kontrollera att avståndet mellan tenor-, alt- och sopranstämmorna inte är för stort genom att visa att

   (intervall($T_2$,$A_2$) ∈ M) ∧ (intervall($A_2$,$S_2$) ∈ M), där M = {tillåtna avstånd}
6. Kontrollera att inga oktavparalleller finns genom att visa att

   ¬ oktavparalleller($<B_1,T_1,A_1,S_1>$,$<B_2,T_2,A_2,S_2>$)

   där definitionen av oktavparalleller är den som ges i avsnitt **3.3.2.5**.
7. Kontrollera att inga kvintparalleller finns genom att visa att

   ¬ kvintparalleller($<B_1,T_1,A_1,S_1>$,$<B_2,T_2,A_2,S_2>$)

   där definitionen av kvintparalleller är den som ges i avsnitt **3.3.2.5**.
8. Kontrollera att intervallen inom stämmorna är tillåtna genom att visa att

   (intervall($B_1$,$B_2$) ∈ $M_B$) ∧ (intervall($T_1$,$T_2$) ∈ $M_T$) ∧
   (intervall($A_1$,$A_2$) ∈ $M_A$) ∧ (intervall($S_1$,$S_2$) ∈ $M_S$),

   där $M_I$ = {tillåtna intervall för stämman I} och beror på $F_1$ och $F_2$.

# 5 Algoritmer

Perfect Harmony har tre huvudsakliga användningsområden:

- *simulering* av arrangering för fyrstämmig kör (härefter kallat endast simulering)
- *kontroll* av en given sats
- funktions*analys* av en given sats

I analogi med detta finns det tre olika sätt att utnyttja den definierade kunskapen hos programmet. Vid simulering vill man skapa någonting, givet vissa förutsättningar. Vid satskontroll vill man helt enkelt kontrollera att samtliga krav som ställs på en sats är uppfyllda. Vid funktionsanalys vill man finna en del av de förutsättningar som gav upphov till satsen. Om man betraktar programmet procedurellt kan man se följande algoritmer eller metoder för hur man går tillväga.

## 5.1 Simulering

Vid simulering förväntas användaren ge satsens första ackord så att programmet har något att utgå ifrån. För att simulera arrangering används följande mycket enkla algoritm.

### 5.1.1 Algoritm *arrangering*

Givet en sats, sats(Tonart, [$V_1$,...,$V_n$]), sker sökning efter en korrekt sats så att programmet börjar med `V1` och försöker hitta bästa möjliga fortsättning $V_2$ med algoritm *harmo*, som beskriver alla krav på fortsättningen $V_2$. Från $V_2$ försöker programmet hitta bästa möjliga $V_3$ o.s.v. Om det inte finns någon tillåten fortsättning $V_{i+1}$ från ett visst $V_i$, så backar programmet till närmast föregående valmöjlighet och söker efter en ny lösning.

### 5.1.2 Algoritm *harmo*

Givet satsens Tonart och tre vertikaler, vertikal($F_1$,<$B_1$,$T_1$,$A_1$,$S_1$>), vertikal($F_2$, <$B_2$,$T_2$,$A_2$,$S_2$>) och vertikal($F_3$,L), där den första är instansierad, gör följande för att få fortsättningen <$B_2$,$T_2$,$A_2$,$S_2$>:

1. Kontrollera med algoritm *undantag* att ackordföljden $F_1$ - $F_2$ är tillåten
2. Slå upp vilka toner som ingår i det ackord som bestäms av Tonart och $F_2$
3. Instansiera sopranstämman $S_2$ med algoritm *närmast*
4. Bestäm utgående från $F_1$, $F_2$ och $F_3$ hur ackordet skall fördubblas med algoritm *dubblering*. Denna ger en ton Baston till basen och en lista Till_Alt_Tenor som resultat
5. Placera tonerna Till_Alt_Tenor med algoritm *närmast*. Detta ger $A_2$ och $T_2$
6. Finn $B_2$, den bästa oktavförekomsten av Baston i förhållande till $B_1$, med algoritm *baston*
7. Kontrollera att allt är korrekt med algoritm *stämföring_OK*. Om så är fallet är vi klara, annars backa tills ny lösning hittats och försök igen.

a) en strategi, kombinerad av de ingående reglerna, enligt följande mönster:

```
dt_right <=
       d_right(_,true_right(_)).
```

b) en specialiserad regel som utför hela operationen i ett enda steg:

```
dt_right(C) <=
       atom(C),
       clause(C,true)
       -> (_ \- C).
```

Med variant b) ovan reduceras antalet anrop i exemplet till 11.

7. Egen fantasi.

   Vid strategiutveckling är det inte minst viktigt att man angriper problemet med hänsyn till det aktuella fallet. Egen fantasi och anpassning till problemets natur är oumbärligt när det gäller att konstruera effektiva strategier.

   I exemplet ovan kan man lätt se att det går att använda samma strategi för `stamton` och `ton`. Denna kan dessutom konstrueras som en sammanslagen regel, `ton/1`, som i ett enda steg utför samma sak som `d_right(_,d_right(_,true_right))`. Regeln finns i filen grundregler.rul.

```
stamtonens_nummer <=
       d_left(_,_,a_left(_,_,v_right(_,ton(_),ton(_)),
         axiom(_,_,_))).

ton(C) <=
       not(data(C)),
       atom(C),
       clause(C,B),
       not(data(B)),
       atom(B),
       clause(B,true)
       -> (P \- C).
```

   Dessa justeringar medför att anropsstatistiken stannar vid detta utseende:

```
Antal anrop: 7              Antal lyckade anrop: 7
```

I ovanstående exempel har vi lyckats decimera antalet anrop till mindre än tio procent, jämfört med standardstrategin `arl`. Exemplets begränsade storlek gör att tidsvinsten knappast blir märkbar, men i ett system med större komplexitet kan en effektiv strategi vara enda möjligheten att överhuvudtaget få ut ett svar.

Man får dock inte stirra sig blind på antalet anrop. Det är inte alltid det går snabbare med en ny strategi. Man måste även beakta den tid ett strategianrop tar i förhållande till ett regelanrop. För detta krävs dock djupare insikter i implementeringen av GCLA II.

```
drstrat <=
       d_right(_,drstrat),
       true_right.
```

Detta ger oss statistiken

```
Antal anrop: 22             Antal lyckade anrop: 20
```

Denna strategiutvecklingsmetod kräver vanligtvis mer möda än de föregående. I större sammanhang kan det vara besvärligt att analysera vilka regelföljder som är vanligast förekommande. Analysen sker lämpligen med hjälp av spårning, varvid man studerar återkommande mönster. Dessutom kan man prova sig fram genom att i en strategi byta plats på två regler och statistiskt mäta vilken av strategivarianterna som är effektivast.

5. Optimalitetsmetoden.

   När man har att göra med klausuler med klart definierade användningsområden, kan man skriva strikt specialiserade strategier för maximal effektivitet. I princip kan dessa ange den exakta sökvägen till det önskade svaret. Detta är numera enkelt att åstadkomma med hjälp av statistikfunktionen `path`, som ger sökvägen för det aktuella beviset. Ett lämpligt sätt att omsätta detta i praktiken är att låta varje atom ha sin egen specialstrategi. Detta är den metod som används mest i vårt program, där regeln är att strategin har samma namn som motsvarande atom:

```
stamtonens_nummer <=
       d_left(_,_,a_left(_,_,v_right(_,stamton,ton),
         axiom(_,_,_))).

stamton <=
       d_right(_,d_right(_,true_right)).

ton <=
       d_right(_,d_right(_,true_right)).
```

   Detta kan betraktas som en optimal strategi i den meningen att den inte misslyckas med några anrop:

```
Antal anrop: 13             Antal lyckade anrop: 13
```

   Men man kan fortfarande göra vissa tidsvinster.

6. Sammanslagning av regler.

   I många program kan man vid spårning upptäcka vissa återkommande regelsekvenser av typen

```
d_right(_,true_right(_)).
```

   Dessa kan slås ihop till en enhet, i form av

```
d_right/2 is exited 4 times
left2/1 is exited 2 times
right2/1 is exited 7 times
strat2/0 is exited 10 times
true_right/0 is exited 2 times
v_right/3 is exited 1 times
```

Vi sorterar reglerna så att de som lyckas flest gånger prövas först.

```
strat3 <=
       right3(strat3),
       left3(strat3),
       axiom(_,_,_).

right3(PT) <=
       d_right(_,PT),
       true_right,
       v_right(_,PT,PT).

left3(PT) <=
       a_left(_,_,PT,PT),
       d_left(_,PT).
```

Statistiken blir nu

```
Antal anrop: 49             Antal lyckade anrop: 29
```

4. Regelberoende strategier.

   Under en fas av strategiutvecklingen använde vi en metod som bygger på att man väljer strategi beroende på vilken regel man använde sist. Till var och en av de vanligaste fördefinierade reglerna konstrueras en strategi. Principen illustreras med strategin nedan.

```
strat4 <=
       d_left(_,_,dlstrat).

dlstrat <=
       a_left(_,_,alstrat1,alstrat2).

alstrat1 <=
       v_right(_,vrstrat1,vrstrat2).

alstrat2 <=
       axiom(_,_,_).

vrstrat1 <=
       d_right(_,drstrat).

vrstrat2 <=
       d_right(_,drstrat).
```

2. Utrensningsmetoden.

   Vi gör en ny ansats för att förbättra vår strategi. Vi använder den fördefinierade statistikfunktionen `get_not_used`. Med hjälp av denna funktion får vi reda på vilka regler som anropats men aldrig lyckats. I vårt fall får vi följande lista:

```
These rules are not used:
         a_right/2
         false_left/1
         o_left/4
         o_right/3
         user_add_left/3
         user_add_right/2
         v_left/3
```

   Dessa regler behövs alltså inte för den aktuella frågan och kan sålunda tas bort ur strategin. Med kunskap om detta konstruerar vi en ny strategi.

```
strat2 <=
       axiom(_,_,_),
       right2(strat2),
       left2(strat2).

right2(PT) <=
       v_right(_,PT,PT),
       true_right,
       d_right(_,PT).

left2(PT) <=
       a_left(_,_,PT,PT),
       d_left(_,_,PT).
```

   När man rensar ut de regler som aldrig används, måste man emellertid vara försiktig, så att man inte plockar bort en regel som visserligen inte används vid den aktuella frågan, men som kanske behövs i andra sammanhang.

   Vi har nu lyckats minska antalet anrop med en tredjedel:

```
Antal anrop: 57             Antal lyckade anrop: 29
```

   Det finns dock fler förbättringar att göra.

3. Frekvenssorteringsmetoden.

   Ett annat effektiviseringssteg, som också kan utföras mekaniskt, är att sortera de ingående reglerna i en strategi med hänsyn till hur ofta de lyckas. Med hjälp av statistikfunktionen `get_exits` kan man placera den vanligaste regeln först i strategin. I exemplet ger `get_exits` följande resultat:

```
a_left/4 is exited 1 times
axiom/3 is exited 1 times
d_left/3 is exited 1 times
```

```
N = stamtonens_nummer(‘a#’) ? ;

N = 34 ? ;

yes
```

Det första svar vi får är bara en ”kopia” av frågan. Detta beror på att regeln `axiom/3`, som är den som prövas först i strategin `arl`, fungerar på alla sorters termer. Vidare avslutas körningen med ‘`yes`’, vilket tyder på att något inte är som det borde vara. Vid spårning[1)] visar det sig bero på att regeln `d_left/3` under körningens gång appliceras på talet 34.

För att komma tillrätta med problemen, inför vi restriktioner på reglerna `axiom/3` och `d_left/3`. I form av ett proviso inför vi en särskild klass av termer som vi kallar `data`. Dit hänför vi vissa hittills odefinierade strukturer, bland annat tal och variabler.

```
data(X) :- number(X).
data(X) :- var(X).

axiom(T,C,I) <=                    % Applicera endast på data
       data(T),
       data(C),
       unify(T,C)
       -> (I@[T|_] \- C).

d_left(T,I,PT) <=                    % Applicera inte på data
       not(data(T)),
       atom(T),
       definiens(T,Dp,N),
       (PT -> (I@[Dp|Y] \- C))
       -> (I@[T|Y] \- C).
```

Nu får vi bara det önskade svaret

```
N = 34 ? ;

no
```

och med hjälp av GCLA II:s *Performance Tool* (avsnitt **2.3.2**) får vi fram följande statistik:

```
Antal anrop: 89            Antal lyckade anrop: 29
```

Det är knappast acceptabelt att ungefär ett av tre anrop lyckas. För strategierna `alr` och `lra` är antalet anrop ännu större (131 respektive 137). Standardstrategierna är inte särskilt effektiva, varför det lönar sig att utveckla bättre strategier. Låt oss försöka med en annan metod.

1) Av eng. *tracing*

## 4.3 Utveckling av sökstrategier

För att minska sökrymden och på så sätt effektivisera programmet, skriver man specialiserade sökstrategier för olika programavsnitt. Under utvecklingen av vårt program fick vi anledning att ägna mycket tid åt detta problem. Vi hann med att pröva ett flertal olika metoder under arbetets gång.

Låt oss studera ett exempel. Vi har följande definitioner:

```
stamtonens_nummer(Ton) <=
       stamton(Ton,Stamton),
       ton(Stamton,Nummer),
       -> Nummer.

ton(Ton,N) <=
       alter(_,Ton,N).

stamton(Ton1,Ton2) <=
       alter(Ton2,Ton1,_).
```

där `alter/3` är en uppräkning av klausuler på formen

```
...
alter(a,a,34).
alter(a,'a#',35).
...
```

som återfinns i filen `ton.def`.

Vår önskan är att få ett så snabbt svar som möjligt på frågan

```
stamtonens_nummer('a#') \- N.
```

### 4.3.1 Metoder

1. Lata metoden.

   Det enklaste sättet är naturligtvis att använda någon av de fördefinierade strategier som finns i GCLA II, exempelvis `arl`, `alr` eller `lra` [Aro92b]. Vid ett försök med frågan

   ```
   | ?- arl \\- stamtonens_nummer('a#') \- N.
   ```

   uppstår dock vissa problem. Vi får följande resultat:

att man vet vilket intervall varje ackordton skall förflyttas för att ge ett korrekt resultat. Trots att antalet klausuler mer än halveras på detta sätt, behövs ändå ett stort antal klausuler, samt en särskild variant som tar hand om övriga fall:

```
harmo(Tonart,vert(F1,[B,T,A,S]),vert(F2,[B1,T1,A1,S1]),Resten) <=
            (sextackord(F2) -> false),                    % Övriga
            (ackordtoner(Tonart,F2) -> [X1,X2,X3,X4]),
            lamplig_oktav(X1,B,B1),omfang_OK(bas,B1),
            tonval(treklang,[X2,X3,X4],[TT,AA,SS]),
            lamplig_oktav(SS,S,S1),omfang_OK(sop,S1),
            lamplig_oktav(AA,A,A1),omfang_OK(alt,A1),
            lamplig_oktav(TT,T,T1),omfang_OK(ten,T1),
            ingen_stamkorsning([B1,T1,A1,S1]),
            inga_oktpar([B,T,A,S],[B1,T1,A1,S1]),
            inga_kvintpar([B,T,A,S],[B1,T1,A1,S1]),
            ej_medrorelse([B,T,A,S],[B1,T1,A1,S1]),
            intervall_OK([B,T,A,S],[B1,T1,A1,S1]).
```

I dessa varianter fanns ingen kunskap om fördubbling av ackord, bara en mängd regler som gav olika resultat och där vi var tvungna att skriva en helt ny stämföringsregel för att införa en ny fördubbling. I vår nuvarande representation har vi ersatt det stora antalet förflyttningsregler med ett antal fördubblingsregler och en förflyttningsmotor som placerar ut stämmorna. De många `harmo`-klausulerna har ersatts av en enda (`bygg.rul`):

```
harmo(Ton,vert(FN1,[B,T,A,S],Pos),vert(FN2,[B1,T1,A1,S1],Pos1),
        vert(F3,T3),Dr) <=
            till_fkn(FN1,FKN1),
            till_fkn(FN2,FKN2),
            (undantag(otillaten_foljd(FKN1,FKN2)) -> false),
            (ackordtoner(Ton,FKN2) -> Acktoner),
            narmast(S,Acktoner,S1,Pos1),
            dubb_krav(Ton,par(FKN1,[B,T,A,S]),
              par(FKN2,[B1,T1,A1,S1]),par(F3,T3),Dr),
            fordubbling(FKN1,par(FKN2,S1),Acktoner,
              val(bas(Bas),Kvar),par([B1,T1,A1],Pos1),Dr),
            hitta_par([A,T],Kvar,[[A,A1],[T,T1]]),
            ingen_stamkorsning(T1,A1,S1),
            baston(FKN1,FKN2,T1,B,Bas,B1),
            stamforing_OK(Ton,par(FKN1,FKN2),par([B,T,A,S],Pos),
              par([B1,T1,A1,S1],Pos1)).
```

Det är fortfarande det andra ackordet som skall instansieras. Detta går till så att de toner som är möjliga att välja på först slås upp av `ackordtoner`. Därefter instansieras sopranstämman av `narmast`, varefter `fordubbling` väljer ut vilka toner som skall ingå i ackordet. Utplaceringen av dessa toner sker sedan med hjälp av `hitta_par` och `baston`. Eftersom man nu inte vet om detta ackord är korrekt görs därefter en stämföringskontroll, här definierad av ett enda villkor istället för de fem sista i den förra upplagan.

Definitionen ovan räcker för att hantera vissa fall av ackordförbindelserna D - T (dominant - tonika) och T - S (tonika - subdominant). Idén är att varje ackordton skall flyttas ett visst intervall för att ge nästa ackord. `overgang/4` avgör om dess första argument är grundton, ters eller kvint, och ger som resultat det intervall som stämman skall förflyttas. Tonerna `S1`, `A1`, `T1` och `B1` bestäms sedan med `hoga_tonen`. Om man försöker tillämpa denna metod i större skala fås dessvärre ett mycket stort antal klausuler. Men det finns också en fördel: stämföringen är garanterat korrekt, så man behöver ingen stämföringskontroll. Vi var dock tvungna att ha en klausul som tog hand om de fall som saknade speciella regler. Den såg ut så här:

```
harmo(Tonart,F1,[B,T,A,S],F2,[B1,T1,A1,S1]) <=       % Övriga
       ackordtoner(Tonart,F2,[X1,X2,X3,X4]),
       oktaver(X1,B1),
       tonval(X2,X3,X4,TT,AA,SS),
       sok_oktav(TT,T,T1),
       sok_oktav(AA,A,A1),
       sok_oktav(SS,S,S1),
       omfang_OK([B1,T1,A1,S1]),
       ingen_stamkorsning([B1,T1,A1,S1]),
       inga_oktpar([B,T,A,S],[B1,T1,A1,S1]),
       inga_kvintpar([B,T,A,S],[B1,T1,A1,S1]),
       intervall_OK([B,T,A,S],[B1,T1,A1,S1]).
```

Det hela går till så att `tonval` väljer ut vilka ackordtoner stämmorna skall ha, varefter `sok_oktav` placerar dem i lämpliga oktaver. Valet av fördubbling sker implicit i `tonval`.

I ett senare försök såg stämföringsklausulerna ut som nedan:

```
harmo(Ton,vert(fkn(fD,DM_1,1),[B,T,A,S]),vert(fkn(fT,DM_2,1),
  [B1,T1,A1,S1]),r_D_T) <=
       (ackordtoner(Ton,fkn(fD,DM_1,1)) ->
         [X1,Grt,Ters,Kvint]),
       (identifiera_ackordtoner([Grt,Ters,Kvint],[T,A,S]) ->
         [Grt_i_D,Ters_i_D,Kvint_i_D]),
       ratt_Ityp([fD,fT,trs,grt,DM_1,DM_2],Trs_grt_Ityp),
       ratt_Ityp([fD,fT,kvnt,trs,DM_1,DM_2],Kvnt_trs_Ityp),
       identiska(Grt_i_D,Kvint_i_T),
       hoga_tonen(Ters_i_D,Trs_grt_Ityp,sekund,Grt_i_T),
       hoga_tonen(Kvint_i_D,Kvnt_trs_Ityp,sekund,Ters_i_T),
       (sortera_toner([Grt_i_T,Ters_i_T,Kvint_i_T]) ->
         [T1,A1,S1]),
       laga_tonen(B,ren,kvint,B11),
       kanske_upp(B11,B1),
       under(B1,T1),
       omfang_OK([B1,T1,A1,S1]).
```

Dur- och mollackord av samma funktion (funktionerna har blivit parametriserade) hanteras i denna version av samma klausul genom att `ratt_Ityp` slår upp intervalltyper som varierar för dur- och mollvarianterna. Principen är annars densamma som förut, nämligen

# 4 Arbetets gång

Examensarbetet har bestått av olika faser, i vilka arbetet haft olika inriktning. Avsikten med detta kapitel är att ge en beskrivning av hur arbetet fortgått och belysa olika skeden i utvecklingen av en GCLA-programmerare.

## 4.1 Att lära sig GCLA II

Den inledande och kanske tyngsta delen av arbetet bestod i att lära sig programspråket GCLA II. Då det ännu inte finns några läroböcker eller motsvarande i ämnet, krävdes det ganska mycket arbete för att försöka förstå hur saker och ting fungerar. De flesta av de avhandlingar som finns om GCLA och dess teoretiska bakgrund, är tämligen teoretiska och kan vara svåra att förstå för en oinvigd.

För att träna upp vår förmåga att skriva GCLA-program började vi tidigt med att skriva diverse mindre moduler, bland annat implementeringar av olika datatyper som listor, stackar och träd. I detta skede ägnade vi den mesta energin åt att skriva definitioner. Arbetet med regel- och strategiutveckling tog vid först senare.

## 4.2 Representation av musikalisk kunskap

När vi kände oss mogna för att påbörja utvecklingen av vårt program, blev det första problemet hur man representerar kunskapen från [Söd59] i GCLA II. Under arbetets gång har vi provat en mängd olika sätt att implementera musikaliska definitioner och föreskrifter.

Vi illustrerar med ett exempel. Vårt första försök att simulera arrangering använde sig av en speciell stämföringsklausul för varje tänkbar ackordföljd, samt en klausul som tog hand om de fall som blev över. I definitionen av `harmo` nedan instansieras det femte argumentet. Det tredje argumentet är föregående ackord och förutsätts vara instansierat.

```
harmo(Tonart,f_D,[B,T,A,S],f_T,[B1,T1,A1,S1]) <=       % D - T
       ackordtoner(Tonart,f_D,[X1,X2,X3,X4]),
       overgang(S,[_,X2,X3,X4],Typ1,Int1),
       overgang(A,[_,X2,X3,X4],Typ2,Int2),
       overgang(T,[_,X2,X3,X4],Typ3,Int3),
       hoga_tonen(S,Typ1,Int1,S1),
       hoga_tonen(A,Typ2,Int2,A1),
       hoga_tonen(T,Typ3,Int3,T1),
       laga_tonen(B,ren,kvint,B1).
harmo(Tonart,f_T,[B,T,A,S],f_S,[B1,T1,A1,S1]) <=       % T - S
       ackordtoner(Tonart,f_T,[_,X2,X3,X4]),
       overgang(S,[_,X2,X3,X4],Typ1,Int1),
       overgang(A,[_,X2,X3,X4],Typ2,Int2),
       overgang(T,[_,X2,X3,X4],Typ3,Int3),
       hoga_tonen(S,Typ1,Int1,S1),
       hoga_tonen(A,Typ2,Int2,A1),
       hoga_tonen(T,Typ3,Int3,T1),
       hoga_tonen(B,ren,kvart,B1).
```

1. Placera ut sopranstämman med minsta möjliga förflyttning. Vanligen är denna stämma given.
2. Välj fördubbling. Valet av fördubbling anger vilka ackordtoner som är möjliga för bas-, tenor- och altstämmorna.
3. Placera ut alt- och tenorstämmorna med minsta möjliga förflyttning.
4. Placera ut basstämman. För basstämman provas inte alltid det alternativ som ger minsta förflyttning först (se `baston/6` i `kbs.def`).
5. Prova om stämföringen är korrekt. Om den är felaktig - försök igen.

## 3.4 Definition av en sats

Def. Vertikal

En vertikal är ett par bestående av en musikalisk funktion och en fyrtuppel av fyra ovanpå varandra placerade toner <B,T,A,S>. Vi skriver detta

vertikal(Funktion, <B,T,A,S>).

Def. Sats

En sats definieras i vårt system som ett par bestående av en tonart och en lista av två eller flera vertikaler. Vi skriver detta som

sats(Tonart, $[V_1,...,V_n]$), där varje $V_i$ är en vertikal och $n \geq 2$.

#### 3.3.3.2 Placering av stämmor

I början av [Söd59] finns ett flertal stämföringsregler av denna typ:

> ”I detta notexempel (61) springer basstämman (1) uppåt eller nedåt från D-ackordets grundton till T-ackordets grundton. Den för båda ackorden gemensamma tonen (= D-ackordets fördubblade grundton) ligger kvar i samma stämma och blir kvint i T-ackordet (2). D-ackordets ters (3) stiger ett halvt tonsteg till T-ackordets grundton; slutligen stiger D-ackordets kvint (4) till T-ackordets ters.” ([Söd59] sid 20)

Detta fick oss att tro att det gick att skriva välspecificerade stämföringsregler av denna typ för förbindelsen av alla ackord. Vi fann dock snart att detta skulle leda till ett mycket stort antal stämföringsregler vilka ändå inte skulle täcka alla möjliga situationer. Man skulle dessutom vara tvungen att skriva andra stämföringsregler för alla de situationer som saknar regler liknande den i citatet ovan. Givna stämföringsregler är dessutom inte absoluta, vilket följande citat om upplösning av D7-ackordets omvändningar till tonika visar.

”... tersen stiger ett halvt tonsteg, kvinten faller vanligast men kan också stiga, och septiman faller till T-ackordets ters. Grundtonen ligger (som redan nämnts) vanligen kvar i samma stämma ...” ([Söd59] sid 49) Några rader längre fram sätts denna regel ur spel: ”Friare upplösningar av denna typ av omvändningsackordens grundton, ters och kvint finner man ofta”.

Vi har istället tagit fasta på att

”Stämföringsreglerna för sextackorden är desamma som i satserna med endast treklanger: gemensam ton ligger kvar i samma stämma; hänsyn tages till ledtonernas naturliga stigande och fallande; inga öppna kvint- eller oktavparalleller; inga förminskade eller överstigande tonsteg i stämmorna.” ([Söd59] sid 40)

Detta har vi sedan generaliserat till följande metod att placera ut stämmorna:

> Givet hur tonerna skall fördubblas, prova i första hand den fortsättning som innebär minsta möjliga förflyttning av stämmorna. Om detta leder till stämföringsfel - prova alla andra möjliga fortsättningar i tur och ordning.

Denna metod kommer att som första alternativ ge de lösningar som specificeras av citaten ovan. Notera att kunskapen om hur stämmor skall placeras är rent procedurell, bestående av procedurer som hittar förslag på lösningar i enlighet med ovanstående rekommendation. Placering av stämmor går i praktiken till så här:

1. Fördubbling av sextackord.

   Ett sextackord är en dur- eller molltreklang där ackordets ters återfinns i basstämman. Om fördubbling av dessa kan man läsa följande:

   ”Sextackordet klingar bäst, om man fördubblar den ackordton, som är i sopranstämman ... Detta gäller dock endast sextackordet sett som isolerad klang. I förbindelse med andra ackord kan en annan fördubbling av sextackordet ibland vara att föredraga.” ([Söd59] sid 36)

   ”Man kan dock inte uppställa denna kvintfördubbling av S3-ackordet ... som en i alla sammanhang gällande regel.” ([Söd59] sid 37)

   ”När flera sextackord följer efter varandra tar hänsynen till stämföringen överhand; man måste då växla mellan grundtons-, ters- och kvintfördubbling ...” ([Söd59] sid 38)

   ”I musik av den typ vi här arbetar med, är fördubblingen av sextackordets grundton och kvint vanligast .... I dominant-ackordet är tersen tonartens inledningston; denna får ej fördubblas.” ([Söd59] sid 39)

   Vi har nu gjort följande val beträffande fördubbling av sextackord:

   a) I D3-ackordet får inte tersen fördubblas.

   b) I följden S3 - D fördubblas i första hand kvinten om situationen liknar de exempel som ges hos [Söd59] sid 37.

   c) Med hänsyn till a) och b) provas i tur och ordning följande fördubblingar:

      - dubbel meloditon
      - dubbel grundton
      - dubbel kvint
      - dubbel ters

   De senare alternativen kommer endast att provas om den fördubbling som valts visar sig leda till stämföringsfel.

2. Fördubbling av kvartsextackord.

   ”Gemensamt för alla typer av kvartsextackord är, att de som regel har bastonen fördubblad.” ([Söd59] sid 60) I vårt program är normalt endast fördubbling av bastonen tillåten. Programmet besitter dock kunskap som gör andra fördubblingar möjliga om användaren explicit tillåter dessa.

4. Förbud mot vissa intervall inom en stämma.

   För att en sats skall vara lättare att sjunga, förbjuds svårsjungna intervall. Detta innebär att alla överstigande och förminskade intervall är förbjudna inom en stämma. Förminskade kvarter och kvinter kan få förekomma i samband med D7-ackordet ([Söd59] sid 55).

   ”I övriga stämmor bör som regel inte förekomma större språng än en kvint” ([Söd59] sid 30). Med ledning av detta är språng större än en kvint förbjudna i alt- och tenorstämmorna.

### 3.3.3 Rekommendationer

Betrakta

Plats för noter.

Med hjälp av ovan givna påbud och förbud vet vi vilka toner som kan ingå i andra ackordet, nämligen c, e och g. Vi vet också att oktav- och kvintparalleller o.s.v. är förbjudna. Men två viktiga problem kvarstår:

- Vilken av tonerna c, e och g skall fördubblas?
- Var skall dessa fyra toner placeras för att uppnå bästa resultat?

Dessa två frågor utgör en stor del av arrangeringen och uppenbarligen den del som är minst väldefinierad och svårast att beskriva. Om man hårdrar det hela en aning kan man säga att vi funnit följande svar på frågorna ovan:

- Välj med hjälp av ditt musikaliska sinne och ditt sunda förnuft den fördubbling som passar bäst med avseende på bland annat klangfärg och spänning mellan ackorden.
- Placera ut dessa fyra toner på det sätt som blir bäst med undvikande av överträdelser mot explicita förbud.

För att kunna simulera arrangering har vi försökt hitta rekommendationer eller tumregler för hur man skall gå tillväga.

#### 3.3.3.1 Fördubbling

Trots att vi bara betraktat en liten del av harmoniläran, har vi ändå inte möjlighet att här beskriva all kunskap vi funnit om fördubbling. Vi betraktar därför som exempel hur sextackord (till exempel T3 och S3) och kvartsextackord (T5, S5 med flera) fördubblas. Notera att användaren har möjlighet att bidra med egen kunskap genom att tillåta och förbjuda fördubblingar. För en fullständig beskrivning, se definitionen av `dubblering/6` i filen `kbs.def`.

#### 3.3.2.4 Förbud mot för stora avstånd mellan stämmor

Vid praktiska försök med simulering av arrangering fann vi att man måste begränsa det tillåtna avståndet mellan stämmorna. Då vi inte kunde hitta något explicit uttalande om detta i [Söd59] införde vi istället en regel från Henry Lindroth *Musikalisk satslär*a ([Lin] sid 14) som säger att avståndet mellan sopran och alt respektive mellan alt och tenor inte får vara större än en oktav.

#### 3.3.2.5 Generella förbud

De nedanstående förbuden gäller generellt mellan två på varandra följande fyrtuppler <B,T,A,S> och $<B_1,T_1,A_1,S_1>$.

1. Förbud mot medrörelse.

   ”Alla stämmor bör inte samtidigt föras i samma riktning, då detta medför en förskjutning av hela klangblocket” ([Söd59] sid 24).

   Def. Medrörelse

   Givet två på varandra följande fyrtuppler <B,T,A,S> och $<B_1,T_1,A_1,S_1>$ har vi medrörelse om

   $(B<B_1 \wedge T < T_1 \wedge A<A_1 \wedge S<S_1) \vee (B>B_1 \wedge T>T_1 \wedge A>A_1 \wedge S >S_1)$

   där X>Y och X<Y läses ”X har högre respektive lägre frekvens än Y”.

2. Förbud mot parallella oktaver.

   Def. Parallella oktaver

   Givet två på varandra följande fyrtuppler $<X_1,X_2,X_3,X_4>$ och $<Y_1,Y_2,Y_3,Y_4>$ har vi parallella oktaver om

   $(\text{oktaver}(X_i,X_j) \wedge \text{oktaver}(Y_i,Y_j), i = 1,...,4 \text{ samt } i \neq j) \wedge (X_i \neq Y_i \vee X_j \neq Y_j)$

   där oktaver(X,Y) läses ”intervallet mellan X och Y är en ren prim eller godtyckligt antal oktaver”.

3. Förbud mot parallella kvinter.

   Def. Parallella kvinter

   Givet två på varandra följande fyrtuppler $<X_1,X_2,X_3,X_4>$ och $<Y_1,Y_2,Y_3,Y_4>$ har vi parallella kvinter om

   $(\text{kvinter}(X_i,X_j) \wedge \text{kvinter}(Y_i,Y_j), i = 1,...,4 \text{ samt } i \neq j) \wedge (X_i \neq Y_i \vee X_j \neq Y_j)$

   där kvinter(X,Y) läses ”avståndet mellan X och Y är ren kvint eller godtyckligt antal oktaver och en kvint”.

- tillåtet_omfång(bas,B) ⇔ (stora e ≤ B ≤ ettstrukna d)
- tillåtet_omfång(tenor,T) ⇔ (lilla c ≤ T ≤ ettstrukna a)
- tillåtet_omfång(alt,A) ⇔ (lilla g ≤ A ≤ tvåstrukna d)
- tillåtet_omfång(sopran,S) ⇔ (ettstrukna c ≤ S ≤ tvåstrukna a)

#### 3.3.1.7 Stämföringspåbud

Det finns även ett litet antal påbud av typen ”i denna situation måste man flytta stämmorna så här”.

- I ackordföljden D4-3 ([Söd59] sid 62) upplöses kvarten i D4 alltid nedåt till tersen i efterföljande dominantackord.
- ”... vid omläggning från sextackord till treklang skall basen springa en ters nedåt” ([Söd59] sid 41). Detta gäller till exempel ackordföljderna T3 - T och S3 - S.
- Kvinten i S65 skall vara förberedd och faller vid upplösning till dominanten alltid till D-ackordets ters. (Detta påbud är för närvarande inte implementerat.)

### 3.3.2 Förbud

En annan klass av väldefinierad kunskap är mängden av förbud. Dessa anger vad som inte är tillåtet. Medan det typiska påbudet deltar aktivt i processen att skriva en sats, exempelvis genom att ange vilka toner som finns i ett visst ackord, så är förbudet passivt. Förbudet betraktar en del av en given sats och avgör om den är tillåten.

#### 3.3.2.1 Förbjudna ackordföljder

- ”På D7-ackord i grundläge får ej följa ett T-ackord i tersläge” ([Söd59] sid 47).
- ”Två kvartsextackord får ej följa varandra” ([Söd59] sid 61).

#### 3.3.2.2 Förbjudna fördubblingar

- Det är inte tillåtet att ha dubbel ters i D3
- Det är inte tillåtet att ha dubbel kvart i D4

#### 3.3.2.3 Förbud mot stämkorsning

”Stämkorsning får ej användas i övningssatserna” ([Söd59] sid 17).

Def. Stämkorsning

Givet en fyrtuppel <B,T,A,S> råder stämkorsning om

B>T ∨ T>A ∨ A>S

där X>Y läses ”tonen X har högre frekvens än tonen Y”.

Def. Skala

Följden <$T_1$,...,$T_n$> är en skala av typen Typ om det gäller att

$T_1$ är skalans grundton,

Typ ∈ {dur, melodisk moll, harmonisk moll, ren moll,
dorisk, frygisk, lydisk, mixolydisk},

intervall($T_1$, $T_i$), i = 2,...,n
är de intervall som föreskrivs för en skala av typen Typ.

### 3.3.1.4 Ackord

”De tre samtidigt klingande tonerna i två på varandra uppbyggda terser kallas *treklang*. Om den lägre tersen är stor och den övre liten blir ackordet en *durtreklang*, omvänt (med den lilla tersen under den stora) blir ackordet en *molltreklang*” ([Söd59] sid 15). Detta är definitionen av dur- och molltreklangerna eller dur- och mollackorden. Det finns även ackord med fyra toner. Notera att tonerna i ett ackord kan placeras fritt i tonsystemets oktaver (godtyckliga förekomster) men ändå betraktas som samma ackord. Exempelvis utgör tonerna i ackordet <lilla c, ettstrukna e, tvåstrukna g> en C-dur-treklang. Vi har definierat kunskapen om vilka intervall som skall gälla mellan tonerna i ett ackord i så kallat *grundläge*.

Def. Ackord i grundläge

Följden <$T_1$,...,$T_n$> där T ∈ {3,4} är ett ackord av typen Typ om det gäller att

$T_1$ är ackordets grundton,

Typ ∈ {dur, moll, 7, m7, maj7, 6, dim, ...},

intervall($T_1$,$T_i$), i = 2,...,n
är de intervall som föreskrivs för ett ackord av typen Typ.

### 3.3.1.5 Huvudtreklanger

«Treklangerna på skalans 1:a, 4:e och 5:e ton är tonartens *huvudtreklanger*. Treklangen på 1:a tonen kallas *Tonika* (T), treklangen på 4:e tonen *Subdominant* (S) och treklangen på 5:e tonen *Dominant* (D).» ([Söd59] sid 18)

Dessa huvudtreklanger förekommer i flera olika varianter, dels som olika typer av ackord (dur, moll m. fl.) och dels med olika bastoner (grundton, ters, kvint m. fl.).

### 3.3.1.6 Stämmornas omfång

I [Söd51] finns det tillåtna omfånget för varje stämma angivet (sid 4). Omfången för de olika stämmorna i en fyrtuppel <B,T,A,S> definieras nedan (X ≤ Y läses ”tonen X har lägre frekvens än tonen Y”):

## 3.3 Formalisering av kunskap

Vid vårt studium av [Söd59] har vi tyckt oss kunna urskilja tre typer av kunskap: för det första fakta eller positiv kunskap, för det andra beskrivningar av vad som är felaktigt/förbjudet och för det tredje en stor mängd lösare kunskap av typen ”vanligen gäller att ...”, ”oftast är det lämpligt att ...” och liknande. Vi kallar dessa tre kunskapstyper för *påbud*, *förbud* och *rekommendationer*.

### 3.3.1 Påbud

Det utmärkande draget för mängden av påbud är att de entydigt beskriver hur något skall se ut, givet vissa förutsättningar. Detta innebär att ett påbud om man så vill kan representeras av en funktion.

#### 3.3.1.1 Universum

Vårt universum är mängden av alla toner. Dessa definieras inte i [Söd59] utan tas för givna. Vad som definieras är relationer på toner. Vi gör en distinktion mellan typ och förekomst av en given ton. En ton av typ T representerar tonen T i alla oktaver. Tonerna lilla g, ettstrukna g och trestrukna g tillhör samma typ g, men är olika förekomster av denna.

#### 3.3.1.2 Intervall

Det grundläggande påbudet i [Söd59] är definitionen av intervall (sid 13-14). Definitionen av intervall utgör grunden för praktiskt taget alla andra definitioner. Ett intervall är en relation mellan två toner. Definitionen av ett intervall är ganska komplicerad och beror bland annat på tonernas stamtoner och avståndet i halvtoner mellan tonerna. Vi definierar endast stamton här och hänvisar den intresserade till avsnitt **6.2.1 Toner och intervall**.

Def. Stamton

Givet en ton ‘namn+suffix’, där suffix tillhör mängden {‘#’,’b’,’ ‘}, så är tonens stamton den ton som ges om man tar bort suffixet. Stamtonerna till c, d# och eb är exempelvis c, d respektive e.

#### 3.3.1.3 Skalor

Olika typer av skalor finns definierade i [Söd59] (sid 7-11). Här beskrivs inte vad en skala är, utan endast vilka toner som tillhör en viss skala. I programmet ser vi en skala som en följd av toner definierad av skalans typ och dess grundton.

# 3 Söderholms harmonilära

Ett av målen med vårt arbete har varit att försöka beskriva vilket slags kunskap som representeras av Valdemar Söderholms *Harmonilära* [Söd59]. I denna ger experten Söderholm sin beskrivning av hur man går tillväga för att skriva fyrstämmig sats.

## 3.1 Om fyrstämmig sats

Att skriva fyrstämmig sats kan definieras som problemet att på bästa sätt placera ut följder av fyratonsackord så att det uppkommer välljudande musik. För ytterligare beskrivningar hänvisar vi till läroböcker i allmän musiklära eller [Söd59].

Exempel:

Plats för noter.

Här anger C styckets tonart. T - S - D - T anger vilken musikalisk funktion det ovanstående ackordet har. Dessa parametrar bestämmer vilka toner som kan ingå i ackordet. Vi har försökt implementera på dator hur en god fyrstämmig sats skall se ut enligt Söderholm.

## 3.2 Avgränsning av problemet

Valdemar Söderholms *Harmonilära* är en lärobok omfattande mer än 200 sidor och vår tid har varit begränsad. Vi har därför varit tvungna att göra ett urval enligt följande:

- Vi tar inte någon hänsyn till rytm utan ser en fyrstämmig sats som en följd av fyrtuppler <B,T,A,S> bestående av fyra toner.
- Vårt system besitter ingen egentlig kunskap om hur man harmoniserar en sats, d.v.s. hur man väljer musikaliska funktioner. Att vi valt bort detta beror till stor del på att denna kunskap ofta beskrivs i rytmiska termer såsom betonad/obetonad taktdel.
- Vi behandlar endast huvudfunktionerna tonika, subdominant och dominant i deras vanligaste utformning ([Söd59] sid 1-75).

Med dessa begränsningar har vi försökt att beskriva harmonilära i GCLA så att den speglar harmonilära enligt Söderholm. Vår datormodell kan

- kontrollera att en given sats uppfyller de stämföringskrav som uppställs i [Söd59].
- analysera vilka funktioner ackorden har i en given sats.
- skriva fyrstämmig sats givet en följd av funktioner. Vanligen är även melodistämman given.

| | |
|---|---|
| `f (fail)` | Förorsakar att exekveringen av det aktuella målet omedelbart misslyckas. |
| `+` | Sätter en spy-point vid den aktuella regeln. |

En lista över samtliga tracer-kommandon finns i [Aro92a].

### 2.3.2 Statistikfunktioner

När programmet fungerar är det dags att effektivisera sökningen. Dessutom händer det ibland att man får oönskade dubbletter av svaret, vilka bör elimineras. För dessa ändamål finns ett statistikpaket, *Performance Tool*, med diverse hjälpmedel. Man laddar in statistikfunktionerna med kommandot

```
gcla_load_statpack.
```

När man vill använda statistikfunktionerna använder man som deduktionssymbol ‘`//-`’ istället för ‘`\\-`’. Frågan blir alltså på följande form:

```
| ?- strategi //- antecedent \- consequent
```

Det finns ett antal olika kommandon att använda för att få statistik om frågan:

| | |
|---|---|
| `get_calls` | Ger antalet anrop av respektive regel och strategi (förutom backtracking-anrop). |
| `get_exits` | Ger antalet lyckade anrop av respektive regel och strategi. |
| `get_retries` | Ger antalet backtracking-anrop av respektive regel och strategi. |
| `get_failures` | Ger antalet misslyckade anrop av respektive regel och strategi. |
| `get_not_used` | Ger en lista över de regler och strategier som anropas men aldrig lyckas. |
| `compare_paths` | Om flera svar har genererats, görs ett försök att finna var bevisen skiljer sig åt för första gången. |
| `path` | Ger sökvägen för beviset av frågan. |

Vid utveckling av effektiva sökstrategier är i synnerhet resultaten av de två första direktiven intressanta. Även kommandot `path` kan vara till nytta, särskilt när det gäller att finna starkt specialiserade strategier (se avsnitt **4.3 Utveckling av sökstrategier**).

## 2.3 Hjälpmedel vid programutveckling

För att underlätta utvecklingen av program finns vissa hjälpmedel tillgängliga. Med hjälp av tracer och statistikfunktioner kan man följa exekveringen och få reda på exempelvis hur många gånger en viss härledningsregel används för att visa en fråga.

### 2.3.1 Tracer

För att programmeraren skall kunna felsöka sina program på ett enkelt och effektivt sätt, finns i GCLA-systemet en *tracer* som stegar sig igenom exekveringen, anrop för anrop. För att starta tracern använder man kommandot

```
gcla_trace.
```

För varje regel eller strategi som anropas, får man en utskrift av ungefär följande utseende:

```
CALL 32 7 arl \\- \- true
```

Här känner vi igen slutet `arl \\- \- true`. Det är vad som ”återstår att visa” av den ursprungliga frågan. Detta kallas för vårt *mål*. Ordet `CALL` visar att det är första försöket att visa detta mål. Vid backtracking står på motsvarande plats ordet `REDO`. När ett mål uppnåtts används ordet `EXIT`, när det misslyckats ordet `FAIL`. Talet `32` betyder att det är det 32:a anropet (`CALL`) som sker. Siffran `7` är ett mått på exekveringens djup. Vi befinner oss här på nivå 7, d.v.s. vi har gått in i sju regler eller strategier som ännu inte har avslutats.

Om man är intresserad av att studera exekveringen kring en viss regel eller strategi, kan man placera en så kallad *spy-point* vid denna och på så vis få tracern att stanna när just denna skall anropas. Kommandot

```
gcla_spy arl.
```

gör sålunda att man kan hoppa över allt utom anrop av den fördefinierade strategin `arl`.

Under felsökningens gång kan man ge kortkommandon till tracern. Här är de mest användbara:

| | |
|---|---|
| `l (leap)` | Programmet körs utan stopp fram till nästa spy-point. |
| `s (skip)` | Hoppar över hela exekveringen av det aktuella målet, tills det antingen lyckas eller misslyckas. |
| `r (retry)` | Startar om aktuellt anrop från början. Om anropet har skett flera gånger (vid backtracking), återgår man till första anropet. |
| `r xxx` | Återgår om möjligt till anropet med nummer `xxx` och startar om detta. Här måste `xxx` vara mindre än aktuellt anropsnummer. |

```
right(PT) <=
       true_right,
       v_right(_,PT,PT),
       a_right(_,PT),
       o_right(_,_,PT),
       d_right(_,PT).
```

Det går också att ställa krav på när en strategi är tillämpbar. Den skrivs då på formen

```
strat <= krav.
strat <= strat1,strat2,...,stratn.
```

Detta skall tolkas så, att om `krav` är uppfyllt kan man försöka med `strat` och provar då `strat1,strat2,...,stratn` i tur och ordning.

Exempel:

```
left_if_false(PT) <=
       (_ \- false).                    % Är högerledet false?
left_if_false(PT) <=
       left(PT).
```

Mera om strategier finns att läsa i avsnitt **4.3 Utveckling av sökstrategier**.

#### 2.2.3.3 Provison

Provison skrivs i GCLA som ‘`huvud :- kropp`’ och skall bestå av Horn-klausuler. Det finns ett antal fördefinierade provison att tillgå och användaren har dessutom möjlighet att skriva egna provison.

Exempel på hur en proviso-definition kan se ut:

```
member(X,[X|_]).
member(X,[_|R]) :- member(X,R).
```

Det finns ett antal fördefinierade provison, exempelvis

| | |
|---|---|
| `atom(T)` | Lyckas om `T` är en objektnivåterm men inte en objektnivåvariabel. |
| `unify(T1,T2)` | Unifierar objektnivåtermerna `T1` och `T2`. |

En fullständig förteckning över fördefinierade provison finns i [Aro91].

En härledningsregel kodas i GCLA som

```
regelnamn(PT1,...,PTn) <=
       (Proviso,
       (PT1 -> Seq1),
       ...,
       (PTn -> Seqn))
       -> Seq.
```

där `Seq`, `Seq1`, ..., `Seqn` är objektnivåsekventer. Regeln ovan skall tolkas som ”om `Proviso` håller och om `PTi` beräknas till `Seqi`, då beräknas `regelnamn(PT1,...,PTn)` till `Seq`.”

Här följer ett par exempel från de härledningsregler som följer med systemet.

```
d_right(C,PT) <=
       atom(C),                                    % Proviso
       clause(C,B),                                % Proviso
       (PT -> (A \- B))        % Försök visa A \- B (premiss)
       -> (A \- C).      % Sekvent som skall visas (slutsats)

a_right((E -> C), PT) <=
       (PT -> ({A1|A] \- C))         % Försök visa [A1|A] \-C
       -> (A \- (A1 -> C)).         % Sekvent som skall visas
```

En fullständig förteckning över fördefinierade härledningsregler återfinns i [Aro91].

#### 2.2.3.2 Strategier

Till skillnad från exempelvis Prolog som använder sig av en enda härledningsregel, `resolution`, så har man i GCLA ett stort antal regler att välja på. Standardstrategin `gcla` innehåller inte mindre än 12 stycken. Tanken med att skriva strategier är att specificera i vilken ordning dessa regler skall provas för att så snabbt som möjligt hitta ett bevis till en given fråga.

Som ett exempel kan vi titta på standardstrategin `gcla`.

```
gcla <= arl.

arl <= axiom(_,_,_),
       right(arl),
       left(arl).
```

Den första klausulen ovan hänvisar helt enkelt till strategin `arl`. Den andra klausulen skall tolkas som att man skall fortsätta antingen med regeln `axiom/3` eller med någon av strategierna `left/1` eller `right/1`. De tre termerna i kroppen till andra klausulen provas i den ordning de står uppräknade. Strategin `right/1` i sin tur ser ut så här:

```
| ?- l_o \\- l_o(a1,[c,g]) \- Svar.

Svar = [c2,g1] ?

yes
```

### 2.2.3 Den procedurella delen

Den procedurella delen (regeldefinitionen, metanivån) av ett GCLA II-program innehåller de härledningsregler och sökstrategier som används för att visa en fråga till systemet. Då GCLA-systemet startas laddas även en uppsättning standardregler och sökstrategier in. Dessa kan sedan ändras fritt av användaren eller ersättas med helt egna strategier och härledningsregler. Det vanligaste (rekommenderade) förfarandet är att man använder sig av de härledningsregler som finns, och skriver strategier för att utnyttja dessa på ett effektivt sätt. En strategi anger i vilken ordning systemet skall prova de olika härledningsreglerna.

#### 2.2.3.1 Härledningsregler

Idén bakom regeldefinitionen är att en härledningsregel kodas som en funktion från regelns premisser till dess slutsats. En härledningsregel

$$\frac{P_1, \ldots, P_n}{C} \qquad regelnamn \qquad Proviso$$

kodas av funktionen

*regelnamn*($P_1$,...,$P_n$) = (*Proviso*, $P_1$, ..., $P_n$) -> $C$

där $P_i$ och $C$ är objektnivåsekventer, t. ex. `a \- b` eller `\- b1, b2`. Härledningsreglerna är alltså funktioner från sekventer till sekventer. För härledningsregeln `axiom/2` gäller det att

```
axiom(Term,Term)
```

beräknas till

```
Term \- Term.
```

För att visa att en fråga håller försöker systemet hitta ett funktionellt uttryck som beräknas till frågan.

cedent ses som premiss och consequent som slutsats, men då det är svårt att finna någon bra svensk översättning har de här fått behålla de engelska beteckningarna.

Man kan även utesluta den första delen och skriva

```
| ?- antecedent \- consequent.
```

vilket tolkas som

```
| ?- gcla \\- antecedent \- consequent.
```

där gcla är en fördefinierad strategi. Det är också tillåtet att utesluta antecedent och ställa frågor på formen

```
| ?- strategi \\- \- consequent.
```

Frågor på formen

i) `A \- C`
ii) `\- C`

tolkas

i) ”Om vi antar A, kan vi då härleda C (i definitionen *D*)?” eller ”Kan C härledas från A?”
ii) ”Går det att härleda C (i definitionen *D*)?”.

Fall ii) motsvarar en fråga i Prolog.

#### 2.2.2.1 Exempel på frågor

Om vi nu antar att vi har lämpliga strategier fs och l_o för exemplen ovan kan vi t. ex. ställa följande frågor:

```
| ?- fs \\- \- flyger(X).

X = tweety ? ;

X = polly ? ;

no


| ?- fs \\- flyger(X) \- false.

X = pengo ? ;

no
```

Vakten ‘`#{A \= dur, A \= moll}`’ ovan betyder att variabeln `A` inte får bindas till `dur` eller `moll`.

#### 2.2.1.4 Exempel på definitioner

Här följer ett par exempel på hur definitioner kan se ut. Allting som följer ‘`%`’ på en rad betraktas som kommentar.

```
flyger(X) <=
       fagel(X),
       (pingvin(X) -> false).

fagel(tweety).
fagel(polly).
fagel(X) <= pingvin(X).    % X är en fågel om X är en pingvin

pingvin(pengo).
```

Notera hur negation åstadkoms genom att vi försöker härleda `false`, vilket inte går om `X` i exemplet instansieras till `pengo`.

En funktionell definition

```
l_o(X,[Y]) <=
       lamplig_oktav_hp(Y,X,Y1)
       -> [Y1].                                        % Svar
l_o(X,[Y|Ys])#{Ys \= []} <=
       (lamplig_oktav_hp(Y,X,Y1),
       (l_o(X,Ys) -> Resten))                % Rekursivt anrop
       -> [Y1|Resten]                                  % Svar
```

Notera att vakten ‘`#{Ys \= []}`’ måste finnas med för att klausulerna skall vara ömsesidigt uteslutande, vilket är en nödvändighet för att en funktionell definition skall uppföra sig korrekt.

### 2.2.2 Frågor till GCLA-systemet

En generell fråga till GCLA-systemet har utseendet

```
| ?- strategi \\- antecedent \- consequent.
```

där `strategi` anger hur definitionen skall utnyttjas (se nedan). Symbolen ‘`\\-`’ kallas för deduktionssymbol på metanivån, medan ‘`\-`’ är deduktionssymbol på objektnivån. `ante-`

#### 2.2.1.1 Syntax

Syntaxen hos definitionen är mycket lik Prolog-syntax. I princip kan man säga att om man har ett Prolog-program

```
p(a) :- q(X), r(Y).
q(b).
```

så fås en syntaktiskt korrekt definition i GCLA om man byter ut ‘:-’ mot ‘<=’.

```
p(a) <= q(X), r(Y).
q(B).
```

#### 2.2.1.2 Terminologi

Då GCLA bygger på PID talar man om termer och villkor, där det gäller att

- en konstant är en term
- en variabel är en term
- om $A_1$,...,$A_n$ är termer och $f$ är en objektnivåkonstruerare av ställighet $n$ så är $f(A_1,...,A_n)$ en term
- alla termer är villkor
- om $C_1$ och $C_2$ är villkor så är även $C_1$ -> $C_2$, ($C_1$,$C_2$), ($C_1$;$C_2$), *true* och *false* villkor
- om $X$ är en objektnivåvariabel och $C$ ett villkor så är *pi* $X\backslash C$ ett villkor
- en atom är en term som inte är en variabel
- om $A$ är en atom och $C$ är ett villkor så är $A$ <= $C$ en klausul
- en ordnad mängd klausuler utgör en definition $D$

#### 2.2.1.3 Mera om syntax

Det som tillkommer hos GCLA jämfört med Prolog är möjligheten att göra antaganden. Vi kan till exempel skriva klausulen

```
a <= (b -> c)
```

vilket skall tolkas som att ”a gäller om man kan visa c då man antar b”. En klausul kan även ha en *vakt* som utgör en restriktion vid unifiering.

```
ackord(Grundton,A,[Grundton,T,K,S])#{A \= dur, A \= moll} <=
       intlista(A,Ints),
       iackord(Ints,Grundton,[T,K,S]).
```

# 2 GCLA II

Detta kapitel är avsett att ge en kort introduktion till programspråket GCLA. För mera omfattande beskrivningar hänvisas till [Aro89], [Aro92a], [Aro92b], [Eri92] och [Kre91].

## 2.1 Historik

GCLA[1)] (Generalized Horn Clause Language) är ett programspråk utvecklat vid SICS, *Swedish Institute of Computer Science*. Den teoretiska bakgrunden till GCLA II utgörs av Lars Hallnäs teori om *partiella induktiva definitioner* (PID). Den teoretiskt intresserade hänvisas till [Hal91], [Eri88] och [Eri91].

Under utvecklingen av GCLA har det funnits ett flertal olika versioner, varav två huvudversioner kan urskiljas: GCLA I och GCLA II (här även kallad bara GCLA), där GCLA II är en utveckling av GCLA I som ger användaren avsevärt förbättrade möjligheter att ge procedurell information för att göra programmen mer effektiva.

GCLA ger användaren möjlighet att skriva både funktionella och relationella program samt att blanda dessa stilar fritt, men torde bäst beskrivas som hörande till familjen logikprogrammeringsspråk. Anledningen till detta är att systemet under exekveringen av ett GCLA-program försöker bygga ett bevis för den fråga användaren ställt. De variabelbindningar som då uppkommer utgör vanligen svaret på frågan.

## 2.2 GCLA-programmets uppbyggnad

Ett GCLA-program består av två delar: en deklarativ del, *definitionen* eller *objektnivån*, och en procedurell del, *regeldefinitionen* eller *metanivån.* Man kan säga att objektnivån beskriver *vad* användaren vill göra medan metanivån beskriver *hur* man vill göra det, eller om man så vill att objektnivån beskriver *vilken kunskap* systemet besitter och metanivån hur denna kunskap skall *utnyttjas*. Dessa två nivåer är skilda åt. Det går inte att binda en metanivåterm till en objektnivåvariabel.

Det är möjligt att ha en deklarativ del med flera olika tillhörande regeldefinitioner som använder kunskapen på olika sätt, exempelvis till simulering och diagnos (se avsnitt **6.5 Generella strategier**).

### 2.2.1 Den deklarativa delen

Den deklarativa delen, definitionen, innehåller den deklarativa kunskap som en tillämpning besitter. Tanken är att denna skall innehålla ett minimum av kontrollinformation. Resterande kontrollinformation som ger ett effektivt program skall finnas i den procedurella delen, regeldefinitionen.

1) Uttalas ”Gisela”

# 1 Inledning

## 1.1 Bakgrund

Efter en tids fruktlöst sökande efter examensarbete inom näringslivet vände vi oss till *Institutionen för informationsbehandling* vid Göteborgs Universitet / Chalmers Tekniska Högskola. Genom studierektor Bror Bjerner fick vi kontakt med vår blivande handledare Lars Hallnäs, som behövde hjälp med att testa och utvärdera ett nytt programspråk, utvecklat i samarbete med SICS, *Swedish Institute of Computer Science*, i Stockholm. Då det visade sig att examensarbetare och handledare hade musiken som gemensamt intresse, föll det sig naturligt att studera en tillämpning inom detta område.

## 1.2 Syfte

Huvudsyftet med arbetet har varit att utreda hur programspråket GCLA II och tillhörande programmeringsmiljö fungerar vid användning i större skala. Under arbetets gång har vi även haft det underordnade syftet att formalisera och implementera regler för harmonilära i form av ett expertsystem, utgående från Valdemar Söderholms *Harmonilära* [Söd59].

# Appendix

# Innehåll

# Perfect Harmony

## Ett musikaliskt expertsystem i GCLA II

**Henrik Siverbo**
**Olof Torgersson**

## Förord

Denna uppsats redovisar resultatet av ett examensarbete omfattande 10 poäng vid matematikerlinjens datalogiska gren, Göteborgs Universitet, utfört av författarna under perioden juni-december 1992. Arbetet har genomförts vid Institutionen för informationsbehandling vid Göteborgs Universitet / Chalmers Tekniska Högskola.

Vi vill rikta några tacksamhetens ord till vår handledare Lars Hallnäs vid Institutionen för informationsbehandling, som hittade uppgiften och som bistått oss med hjälp och uppmuntran under arbetets gång. Ett stort tack också till Martin Aronsson vid SICS för värdefulla tips och svar på otaliga frågor.

Göteborg i december 1992

*Henrik Siverbo*
*Olof Torgersson*

---

## Sammanfattning

Denna uppsats beskriver ett expertsystem för arrangering av fyrstämmig körsats och dess implementering i språket GCLA II.

GCLA II är ett programmeringsspråk avsett att underlätta utvecklandet av framförallt kunskapsbaserade system, med möjlighet att skriva såväl relationella som funktionella program. Kunskapsunderlaget för programmet utgörs av Valdemar Söderholms *Harmonilära* och vi visar hur kunskapen i denna formaliserats för att kunna ligga till grund för expertsystemet. Olika implementeringsmöjligheter i GCLA II beskrivs också, bland annat ett exempel på hur vissa kategorier av GCLA-program kan modulariseras och hur man utgående från en kunskapsdefinition kan få flera olika procedurella beteenden med hjälp av så kallade strategier som beskriver hur kunskapen skall utnyttjas.