Institut für Informatik

DisLog Library Manual

November 1998

This manual documents the DisLog Library, a toolkit
for reasoning in disjunctive deductive databases.
DisLog runs under the Prolog of SICStus and SWI.

Written by Dietmar Seipel

Version 1.3.1

Institut für Informatik
Universität Würzburg
Am Hubland
D–97074 Würzburg
Germany
seipel@informatik.uni-wuerzburg.de

Table of Contents

1 Introduction

To consult the DisLog package, you will normally enter the query

```
| ?- [dislog].
```

A general clause $C = L1 \vee \ldots \vee Ln$ is represented by a list *[L1,...,Ln]* of literals. A general Herbrand state – for short Herbrand state – $S = \{C1, \ldots, Cn\}$ is represented by a list *[C1,...,Cn]* of general clauses.

A disjunctive normal logic program contains three kinds of rules: disjunctive normal logic rules $Head \leftarrow Pos\_Body \wedge Neg\_Body$, disjunctive logic rules $Head \leftarrow Body$, and disjunctive facts D_Fact. They are represented in DisLog by terms like follows:

```
Head - Pos_Body - Neg_Body
Head - Body
D_Fact
```

A clause tree *Tree* is represented by a list *[Root,Tree1,...,Treek]* consisting of its root clause *Root* and its subtrees *Tree1,...,Treek*. The subtrees *Tree1,...,Treek* are clause trees themselves, and k can be *0*.

A support–for–negation set *SN(E)* = {*C1,...,Ck*} is represented by the list of the conjunction *E* and the disjunctions *C1,...,Ck*:

```
[E,C1,...,Ck]
```

A state pair *S* = <{*C1,...,Cn*},{*E1,...,Em*}> is represented by a list

```
[[C1,...,Cn],[E1,...,Em]]
```

2 Herbrand States and Coins

`state_prune(+`*State1*`,`*?State2*`)`

removes the duplicates among the disjunctions in the Herbrand state *State1*, yields *State2*. Example:

```
| ?- state_prune([[a,b],[a,b],[b,c]],State2).

State2 = [[a,b],[b,c]] ?

yes
```

`state_subsumes(+`*State1*`,+`*State2*`)`

is true when the Herbrand state *State1* subsumes the state *State2*. Example:

```
| ?- state_subsumes([[a],[b,c]],[[a,b],[b,c,d]]).

yes
```

`state_subtract(+`*State1*`,+`*State2*`,`*?State3*`)`

computes the Herbrand state *State3* of all disjunctions from *State1* which are not subsumed by disjunctions from *State2*. Example:

```
| ?- state_subtract([[a,b],[b,d]],[[a],[b,c]],State3).

State3 = [[b,d]] ?

yes
```

`state_can(+`*State1*`,`*?State2*`)`

the Herbrand state *State2* is the canonical form of *State1*. Example:

```
| ?- state_can([[a],[a,b],[a,b,c],[b,d]],State2).

State2 = [[a],[b,d]] ?

yes
```

`state_disjunction(+`*State1*`,+`*State2*`,`*?State3*`)`

for two Herbrand states *State1* and *State2* the state disjunction is the Herbrand state $State3 = State1 \vee State2$. Example:

```
| ?- state_disjunction([[a],[b]],[[b,c],[b,d]],State3).

State3 = [[b,c],[b,d],[a,b,c],[a,b,d]] ?

yes
```

`state_resolve`(+*State1*,+*Literal*,?*State2*)

State2 is the state resolvent of the Herbrand state *State1* and the literal –*Literal*. Example:

```
| ?- state_resolve([[a,b,d],[a,b,e],[a,c]],b,State2).

State2 = [[a,d],[a,e]] ?

yes
```

`state_reduce`(+*State1*,+*Literal*,?*State2*)

State2 is the state reduction of the Herbrand state *State1* and the ground literal *Literal*. Example:

```
| ?- state_reduce([[a,b,d],[a,b,e],[a,c]],b,State2).

State2 = [[a,c]] ?

yes
```

`state_to_coin`(+*State*,-*Coin*)

Coin is the set of all minimal Herbrand models of the Herbrand state *State*. Example:

```
| ?- state_to_coin([[a,b],[b,c],[d]],Coin).

Coin = [[a,c,d],[b,d]] ?

yes
```

`coin_to_state`(+*Coin*,-*State*)

State is the Herbrand state of all positive disjunctions that are true in all Herbrand interpretations in *Coin*. Example:

```
| ?- coin_to_state([[a,c,d],[b,d]],State).

State = [[a,b],[b,c],[d]] ?

yes
```

3 The Clause Tree – A Data Structure

`state_to_tree(+`*State*`, `*?Tree*`)`

transforms a Herbrand state *State* to a clause tree *Tree*. Example:

```
| ?- state_to_tree([[a,b,d],[a,b,e],[a,c]],Tree).

Tree = [[a],[[c]],[[b],[[e]],[[d]]]] ?

yes
```

`tree_to_state(+`*Tree*`, `*?State*`)`

transforms a clause tree *Tree* to a Herbrand state *State*. Example:

```
| ?- tree_to_state([[a],[[b],[[d]],[[e]]],[[c]]],State).

State = [[a,b,d],[a,b,e],[a,c]] ?

yes
```

`tree_subsumes_clause(+`*Tree*`,+`*Clause*`)`

is true when the clause tree *Tree* subsumes the disjunctive clause *Clause*. Example:

```
| ?- tree_subsumes_clause([[a],[[b],[[d]],[[e]]],[[c]]],[a,b,c]).

yes
```

`tree_subsumes_state(+`*Tree*`,+`*State*`)`

is true when the clause tree *Tree* subsumes the state *State*. Example:

```
| ?- tree_subsumes_state([[a],[[b],[[d]],[[e]]],[[c]]],
        [[a,b,d,e],[a,b,c]]).

yes
```

`state_tree_subtract(+`*State1*`,+`*Tree*`, `*?State2*`)`

computes the Herbrand state *State3* of all disjunctions from the Herbrand state *State1* which are not subsumed by disjunctions from the clause tree *Tree*. Example:

```
| ?- state_tree_subtract([[a,d],[a,b,c]],
        [[a],[[b],[[d]],[[e]]],[[c]]],State2).

State2 = [[a,d]] ?

yes
```

`tree_state_subtract(+`*Tree1*`,+`*State*`,`*?Tree2*`)`

computes the clause tree *Tree2* containing all clauses of the clause tree *Tree1* which are not subsumed by the Herbrand state *State*. Example:

```
| ?- tree_state_subtract([[a],[[d]],[[b,c]]],
        [[a,b,d],[a,b,e],[a,c]],Tree2).

Tree2 = [[a,d]] ?

yes
```

`tree_resolve(+`*Tree1*`,+`*Literal*`,`*?Tree2*`)`

Tree2 is the tree resolvent of the clause tree *Tree1* and the literal *–Literal*. Example:

```
| ?- tree_resolve([[a],[[b],[[d]],[[e]]],[[c]]],b,Tree2).

Tree2 = [[a],[[],[[d]],[[e]]]] ?

yes
```

`tree_reduce(+`*Tree1*`,+`*Literal*`,`*?Tree2*`)`

Tree2 is the tree reduction of the clause tree *Tree1* and the ground literal *Literal*. Example:

```
| ?- tree_reduce([[a],[[b],[[d]],[[e]]],[[c]]],b,Tree2).

Tree2 = [[a],[[c]]] ?

yes
```

4 Disjunctive Consequence Operators

`tps_operator(+`*Program*`,+`*State1*`,?`*State2*`)`

for a disjunctive logic program *Program* the T_P^S–operator applied to the Herbrand state *State1* yields the state *State2*. Example:

```
| ?- tps_operator([[c]-[a],[a,d]-[c]],[[a,b,c],[a,e],[b,d]],
        State2).

State2 = [[a,b,d],[b,c],[c,e]] ?

yes
```

`tps_operator_dc(+`*Program*`,+`*State1*`,?`*State2*`)`

for a disjunctive logic program *Program* the T_P^S–operator applied to the Herbrand state *State1* followed by delta computation w.r.t. *State1* and canonization yields the state *State2*. Example:

```
| ?- tps_operator_dc([[c]-[a],[a,d]-[c]],[[a,b,c],[a,e],[b,d]],
        State2).

State2 = [[b,c],[c,e]] ?

yes
```

`tps_delta_operator(+`*Program*`,+`*State1*`,+`*State2*`,?`*State3*`)`

for a disjunctive logic program *Program* the T_P^S–operator applied to the Herbrand states *State1* and *State2* yields the state *State3*. Example:

```
| ?- tps_delta_operator([[a]-[b,c]],[[b,d],[c,e],[d,g]],
        [[b,f],[c,g]],State3).

State2 = [[a,d,g],[a,e,f]] ?

yes
```

`tps_delta_operator_dc(+`*Program*`,+`*State1*`,+`*State2*`,?`*State3*`)`

for a disjunctive logic program *Program* the T_P^S–operator applied to the Herbrand states *State1* and *State2* followed by delta computation w.r.t. *State1* and canonization yields the state *State3*. Example:

```
| ?- tps_delta_operator_dc([[a]-[b,c]],[[b,d],[c,e],[d,g]],
        [[b,f],[c,g]],State3).

State2 = [[a,e,f]] ?

yes
```

`tree_tps_operator(`+*Program*`,`+*Tree1*`,`?*Tree2*`)`

for a disjunctive logic program *Program* the T_P^S–operator applied to the clause tree *Tree1* yields the derived clause tree *Tree2*. Example:

```
| ?- tree_tps_operator([[c]-[a],[a,d]-[c]],
        [[],[[b,d]],[[a],[[e]],[[b,c]]]],Tree2).

Tree2 = [[],[[c,e]],[[b],[[c]],[[a,d]]]] ?

yes
```

`tree_tps_operator_dc(`+*Program*`,`+*Tree1*`,`?*Tree2*`)`

for a disjunctive logic program *Program* the T_P^S–operator applied to the clause tree *Tree1* followed by delta computation w.r.t. *Tree1* and canonization yields the derived clause tree *Tree2*. Example:

```
| ?- tree_tps_operator_dc([[c]-[a],[a,d]-[c]],
        [[],[[b,d]],[[a],[[e]],[[b,c]]]],Tree2).

Tree2 = [[c],[[e]],[[b]]] ?

yes
```

`tree_tps_delta_operator(`+*Program*`,`+*Tree1*`,`+*Tree2*`,`?*Tree3*`)`

for a disjunctive logic program *Program* the T_P^S–operator applied to the clause trees *Tree1* and *Tree2* yields the derived clause tree *Tree3*. Example:

```
| ?- tree_tps_delta_operator([[a]-[b,c]],
        [[],[[c,e]],[[d],[[g]],[[b]]]],
        [[],[[c,g]],[[b,f]]],Tree3).

Tree2 = [[a],[[e,f]],[[d,g]]] ?

yes
```

`tree_tps_delta_operator_dc(`+*Program*`,`+*Tree1*`,`+*Tree2*`,`?*Tree3*`)`

for a disjunctive logic program *Program* the T_P^S–operator applied to the clause trees *Tree1* and *Tree2* followed by delta computation w.r.t. *Tree1* and canonization yields the derived clause tree *Tree3*. Example:

```
| ?- tree_tps_delta_operator_dc([[a]-[b,c]],
        [[],[[c,e]],[[d],[[g]],[[b]]]],
        [[],[[c,g]],[[b,f]]],Tree3).

Tree2 = [[a,e,f]] ?

yes
```

`tps_fixpoint(`+*Program*,+*State1*,?*State2*`)`

for a disjunctive logic program *Program* and a Herbrand state *State1* the canonized Herbrand state *State2* of all logically implied disjunctive facts is computed. Example:

```
| ?- tps_fixpoint([[c]-[a],[a,d]-[c]],[[a,b,c],[a,e]],State2).

State2 = [[a,b,d],[a,e],[b,c],[c,e]] ?

yes
```

`tps_fixpoint_cc(`+*Program*,+*State1*,?*State2*`)`

for a disjunctive logic program *Program* and a Herbrand state *State1* the canonized Herbrand state *State2* of all logically implied disjunctive facts is computed by a fast c++ implementation. Example:

```
| ?- tps_fixpoint_cc([[c]-[a],[a,d]-[c]],[[a,b,c],[a,e]],State2).

State2 = [[a,b,d],[a,e],[b,c],[c,e]] ?

yes
```

`tree_tps_fixpoint(`+*Program*,+*Tree1*,?*Tree2*`)`

for a disjunctive logic program *Program* and a clause tree *Tree1* the derived clause tree *Tree2* representing all logically implied disjunctive facts is computed. Example:

```
| ?- tree_tps_fixpoint([[c]-[a],[a,d]-[c]],
        [[a],[[b,c]],[[e]]],Tree2).

Tree2 = [[],[[b],[[a,d]],[[c]]],[[e],[[c]],[[a]]]] ?

yes
```

`tps_fixpoint_iteration(`+*Program*,+*State*`)`

for a disjunctive logic program *Program* the ordinal powers of the T_P^S–operator are computed and pretty printed, *State* is the least fixpoint. Example:

```
| ?- tps_fixpoint_iteration([[c]-[a],[a,d]-[c],[a,b,c],[a,e]],
        State).

tps_iteration 1:   { a;b;c. a;e. }
tps_iteration 2:   { a;b;d. b;c. c;e. }
tps_iteration 3:   {  }
tps_fixpoint:      { a;b;d. a;e. b;c. c;e. }

State = [[a,b,d],[a,e],[b,c],[c,e]] ?

yes
```

`tree_tpi_operator(`+*Program*,+*Tree1*,?*Tree2*`)`

for a disjunctive logic program *Program* and the model tree *Tree1* the T_P^I–operator applied to *Tree1* yields the model tree *Tree2*. Example:

```
| ?- tree_tpi_operator([[p(X,Y)]-[a(X,Z),p(Z,Y)]],
        [[p(b,d),p(c,d)],[[a(a,b)]],[[a(a,c)]]],Tree2).

Tree2 = [[p(b,d),p(c,d)],[[a(a,b),p(a,d)]],[[a(a,c),p(a,d)]]] ?

yes
```

`tree_tpi_operator_ground(`+*Program*,+*Tree1*,?*Tree2*`)`

for a ground disjunctive logic program *Program* and the model tree *Tree1* the T_P^I–operator applied to *Tree1* yields the model tree *Tree2*. Example:

```
| ?- tree_tpi_operator_ground(
        [[p(a,d)]-[a(a,b),p(b,d)],[p(a,d)]-[a(a,c),p(c,d)]],
        [[p(b,d),p(c,d)],[[a(a,b)]],[[a(a,c)]]],Tree2).

Tree2 = [[p(b,d),p(c,d)],[[a(a,b),p(a,d)]],[[a(a,c),p(a,d)]]] ?

yes
```

`tree_tpi_delta_operator(`+*Program*,+*Tree1*,+*Delta1*,?*Tree2*,?*Delta2*`)`

for a disjunctive logic program *Program* and two model trees *Tree1* and *Delta1* the T_P^I–operator is applied to *Delta1*, *Delta2* represents the derived interpretations which are not subsumed by *Tree1*, the interpretations in *Delta1* which are not extended by T_P^I are added to *Tree1*, this yields the model tree *Tree2*. Example:

```
| ?- tree_tpi_delta_operator([[d,e]-[a,c],[b,d,f]-[a,c,e]],
        [[c],[[b]]],[[c],[[a],[[d]],[[e]]]],Tree2,Delta2).

Tree2 = [[c],[[b]],[[a,d]]],
Delta2 = [[a,c,e,f]] ?

yes
```

`tree_tpi_fixpoint(`+*Program*,+*Tree1*,?*Tree2*`)`

for a disjunctive logic program *Program* and a model tree *Tree1* the derived model tree *Tree2* representing all models of *Program* which are supersets of some model represented by *Tree1* is computed. Example:

```
| ?- tree_tpi_fixpoint([[d,e]-[a,c],[b,d,f]-[a,c,e]],
        [[c],[[a]],[[b]]],Tree2).

Tree2 = [[c],[[b]],[[a,d]],[[a,e,f]]] ?

yes
```

`tpi_fixpoint_iteration(+`*Program*`,+`*Models*`)`

for a disjunctive logic program *Program* the ordinal powers of the T_P^I–operator are computed and pretty printed, *Models* is the set of minimal models. Example:

```
| ?- tpi_fixpoint_iteration([[c]-[a],[a,d]-[c],[a,b,c],[a,e]],
        Models).

tpi_iteration 1:   {  }
tpi_iteration 2:   { b,e. }
tpi_iteration 3:   { a,c. b,e. c,d,e. }
tpi_fixpoint:      { a,c. b,e. c,d,e. }

Models = [[a,c],[b,e],[c,d,e]] ?

yes
```

5 Program Transformations

`partial_evaluation(+Program1,-Program2)`

The disjunctive normal logic program *Program1* is partially evaluated. The result is a set *Program2* of conditional facts. Conditional facts are disjunctive normal logic rules that are ground and do not have any positive body atoms. Example:

```
| ?- partial_evaluation([[tc(X,Y)]-[arc(X,Y)]-[blocked(X,Y)],
          [tc(X,Z)]-[tc(X,Y),arc(Y,Z)]-[blocked(Y,Z)],
          [arc(a,a),arc(a,b)],[blocked(b,b)]],Program2).

Program2 = [[tc(a,a),tc(a,b)]-[]-[blocked(a,a),blocked(a,b)],
     [arc(a,b),tc(a,a)]-[]-[blocked(a,a)],
     [arc(a,a),tc(a,b)]-[]-[blocked(a,b)],
     [arc(a,a),arc(a,b)]-[]-[],[blocked(b,b)]-[]-[]] ?

yes
```

`gl_transformation_non_ground(+Program1,+Interpretation,-Program2)`

The Gelfond/Lifschitz transformation w.r.t. the Herbrand interpretation *Interpretation* is applied to the disjunctive normal logic program *Program1*. The result is a ground disjunctive logic program *Program2*. Example:

```
| ?- gl_transformation_non_ground(
         [[tc(X,Y)]-[arc(X,Y)]-[blocked(X,Y)],
          [tc(X,Z)]-[tc(X,Y),arc(Y,Z)]-[blocked(Y,Z)],
          [arc(a,a),arc(a,b)],[blocked(b,b)]],
         [arc(a,b),blocked(a,a),blocked(b,a),blocked(b,b)],
         Program2).

Program2 = [[tc(a,b)]-[arc(a,b)],[tc(a,b)]-[tc(a,a),arc(a,b)],
     [tc(b,b)]-[tc(b,a),arc(a,b)],[arc(a,a),arc(a,b)],
     [blocked(b,b)]] ?

yes
```

`gl_transformation(+Program1,+Interpretation,-Program2)`

The Gelfond/Lifschitz transformation w.r.t. the Herbrand interpretation *Interpretation* is applied to the ground disjunctive normal logic program *Program1*. The result is a ground disjunctive logic program *Program2*. Example:

```
| ?- gl_transformation([[p(a)]-[]-[q(a)],[p(b)]-[]-[q(b)],
          [q(a),q(b)]],[p(c),q(a)],Program2).

Program2 = [[p(b)],[q(a),q(b)]] ?

yes
```

`evidential_transformation(+Program1,-Program2)`

For the disjunctive normal logic program *Program1* the evidential transformation *Program2* is computed. Example:

```
| ?-
evidential_transformation([[a,b]-[c]-[d],[d]-[]-[a]],Program2).

Program2 = [ [a,b,'E' d]-[c], [d,'E' a] ] ?

yes
```

`stratify_program(+Program,-Programs)`

For the stratifiable, disjunctive normal logic program *Program* a stratification is computed in the form of a set *Programs* of programs. For non-stratifiable programs the call fails. Example:

```
| ?- stratify_program([[p]-[]-[a],[q]-[]-[b],[a,b]],Programs).

Programs = [ [[a,b]], [[q]-[]-[b]], [[p]-[]-[a]] ] ?

yes
```

`tu_transformation(+Program1,-Program2)`

For the disjunctive normal logic program *Program1* the tu-transformation *Program2* is computed. Example:

```
| ?- tu_transformation([[a,b]-[c]-[d],[d]-[]-[a]],Program2).

Program2 = [ [u(X)]-[t(X)],
     [t(a),t(b)]-[t(c)]-[u(d)],[u(a),u(b)]-[u(c)]-[t(d)],
     [t(d)]-[]-[u(a)],[u(d)]-[]-[t(a)] ] ?

yes
```

6 Model–Based Semantics

`minimal_models(`+*Program*`,`?*Minimal_Models*`)`

for a disjunctive logic program *Program* the set *Minimal_Models* of minimal Herbrand models is computed as the least fixpoint of the T_P^I–operator. Example:

```
| ?- minimal_models([[c]-[a],[a,d]-[c],[a,b,c],[a,e]],
        Minimal_Models).

Minimal_Models = [[a,c],[b,e],[c,d,e]] ?

yes
```

`minimal_model_state(`+*Program*`,`?*Minimal_Model_State*`)`

for a disjunctive logic program *Program* the minimal model state *Minimal_Model_State* is computed as the least fixpoint of the T_P^S–operator. Example:

```
| ?- minimal_model_state([[c]-[a],[a,d]-[c],[a,b,c],[a,e]],
        Minimal_Model_State).

Minimal_Model_State = [[a,b,d],[a,e],[b,c],[c,e]] ?

yes
```

`minimal_model_semantics_s(`+*Program*`,`?*Minimal_State*`)`

for a disjunctive logic program *Program* the minimal model semantics *Minimal_State* is computed from the minimal model state. Example:

```
| ?- minimal_model_semantics_s([[c]-[a],[a,d]-[c],
        [a,b,c],[a,e]],Minimal_State).

Minimal_State = [[a,b,d],[a,d,~c],[a,e],[b,c],[b,d,~e],[c,e],
    [c,~a],[c,~d],[d,~c,~e],[e,~b],[e,~d],[~a,~b],[~a,~d],
    [~a,~e],[~b,~c],[~b,~d]] ?

yes
```

`minimal_model_semantics_i(`+*Program*`,`?*Minimal_State*`)`

for a disjunctive logic program *Program* the minimal model semantics *Minimal_State* is computed from the set of minimal Herbrand models. Example:

```
| ?- minimal_model_semantics_i([[c]-[a],[a,d]-[c],
        [a,b,c],[a,e]],Minimal_State).

Minimal_State = [[a,b,d],[a,d,~c],[a,e],[b,c],[b,d,~e],[c,e],
    [c,~a],[c,~d],[d,~c,~e],[e,~b],[e,~d],[~a,~b],[~a,~d],
    [~a,~e],[~b,~c],[~b,~d]] ?

yes
```

`perfect_models(+`*Program*`,`*?Perfect_Models*`)`

for a stratified disjunctive normal logic program *Program* the set *Perfect_Models* of perfect Herbrand models is computed. Example:

```
| ?- perfect_models([[p]-[]-[a],[p]-[]-[b],[a,b]],
        Perfect_Models).

Perfect_Models = [[a,p],[b,p]] ?

yes
```

`perfect_model_state(+`*Program*`,`*?Perfect_Model_State*`)`

for a stratified disjunctive normal logic program *Program* the perfect model state *Perfect_Model_State* is computed as the disjunction of all perfect Herbrand models. Example:

```
| ?- perfect_model_state([[p]-[]-[a],[p]-[]-[b],[a,b]],
        Perfect_Model_State).

Perfect_Model_State = [[a,b],[p]] ?

yes
```

`perfect_model_semantics(+`*Program*`,`*?Perfect_State*`)`

for a stratified disjunctive normal logic program *Program* the perfect model semantics *Perfect_State* is computed from the set of perfect Herbrand models. Example:

```
| ?- perfect_model_semantics([[p]-[]-[a],[p]-[]-[b],[a,b]],
        Perfect_State).

Perfect_State = [[a,b],[p],[~a,~b]] ?

yes
```

`stable_models(+`*Program*`,`*?Stable_Models*`)`

for a disjunctive normal logic program *Program* the set *Stable_Models* of all stable models is computed. Example:

```
| ?- stable_models([[win(X)]-[move(X,Y)]-[win(Y)],
        [move(a,b)],[move(b,c)]],Stable_Models).

Stable_Models = [[move(a,b),move(b,c),win(b)]] ?

yes
```

`stable_models_normal(+`*Program*`,`*?Stable_Models*`)`

for a normal logic program *Program* the set *Stable_Models* of all stable models is computed using a sophisticated backtracking algorithm. Example:

```
| ?- stable_models_normal([[win(X)]-[move(X,Y)]-[win(Y)],
        [move(a,b)],[move(b,c)],[move(c,d)],[move(d,a)]],
        Stable_Models).

Stable_Models =
      [[move(a,b),move(b,c),move(c,d),move(d,a),win(a),win(c)],
       [move(a,b),move(b,c),move(c,d),move(d,a),win(b),win(d)]] ?

yes
```

`stable_models_normal_e(+`*Program*`,+`*E*`,`*?Stable_Models*`)`

for a normal logic program *Program* and a set E of ground atoms the set *Stable_Models* of all stable models that contain E is computed using a synthesis of resolution and backtracking. Example:

```
| ?- stable_models_normal_e([[win(X)]-[move(X,Y)]-[win(Y)],
        [move(a,b)],[move(b,c)],[move(c,d)],[move(d,a)]],
        [win(a)],Stable_Models).

Stable_Models =
      [[move(a,b),move(b,c),move(c,d),move(d,a),win(a),win(c)]] ?

yes
```

`stable_model_state`(+*Program*, *?Stable_Model_State*)

for a disjunctive normal logic program *Program* the stable model state *Stable_Model_State* is computed as the disjunction of all stable Herbrand models. Example:

```
| ?- stable_model_state([[win(X)]-[move(X,Y)]-[win(Y)],
        [move(a,b)],[move(b,c)]],Stable_Model_State).

Stable_Model_State = [[move(a,b)],[move(b,c)],[win(b)]] ?

yes
```

`stable_model_semantics`(+*Program*, *?Stable_State*)

for a disjunctive normal logic program *Program* the stable model semantics *Stable_State* is computed from the set of stable Herbrand models. Example:

```
| ?- stable_model_semantics([[win(X)]-[move(X,Y)]-[win(Y)],
        [move(a,b)],[move(b,c)]],Stable_State).

Stable_State = [[move(a,b)],[move(b,c)],[win(b)],
   [~move(a,a)],[~move(a,c)],[~move(b,a)],[~move(b,b)],
   [~move(c,a)],[~move(c,b)],[~move(c,c)],[~win(a)],[~win(c)]] ?

yes
```

`partial_stable_models(+`*Program*`,`*?Partial_Stable_Models*`)`

for a disjunctive normal logic program *Program* the set *Partial_Stable_Models* of all partial stable models is computed. Example:

```
| ?- partial_stable_models([[win(X)]-[move(X,Y)]-[win(Y)],
        [move(a,b)],[move(b,c)]],Partial_Stable_Models).

Partial_Stable_Models = [[move(a,b),move(b,c),win(b),
   ~move(a,a),~move(a,c),~move(b,a),~move(b,b),
   ~move(c,a),~move(c,b),~move(c,c),~win(a),~win(c)]] ?

yes
```

`evidential_stable_models(+`*Program*`,`*?Models*`)`

for a disjunctive normal logic program *Program* the set *Models* of all evidential stable models is computed. Example:

```
| ?- evidential_stable_models([[a,b,c],
        [a]-[]-[b],[b]-[]-[c],[c]-[]-[a]],Models).

Models = [[t(a),'E' t(c)],[t(b),'E' t(a)],[t(c),'E' t(b)]] ?

yes
```

`partial_evidential_stable_models(+`*Program*`,`*?Models*`)`

for a disjunctive normal logic program *Program* the set *Models* of all partial evidential stable models is computed. Example:

```
| ?- partial_evidential_stable_models([[a,b,c],
        [a]-[]-[b],[b]-[]-[c],[c]-[]-[a]],Models).

Models = [[t(a),u(b),'E' u(c)],[t(b),u(c),'E' u(a)],
   [t(c),u(a),'E' u(b)]] ?

yes
```

7 Closed–World–Assumptions

`state_to_egcwa`(+*State*, ?*Egcwa*)

the extended generalized closed–world–assumption *Egcwa* of a Herbrand state *State* is computed. Example:

```
| ?- state_to_egcwa([[a,b,d],[b,c],[c,d]],Egcwa).

Egcwa = [[a,b],[a,d],[b,c,d]] ?

yes
```

`state_to_gcwa`(+*State*, ?*Gcwa*)

the generalized closed–world–assumption *Gcwa* of a Herbrand state *State* is computed. Example:

```
| ?- state_to_gcwa([[a,b],[a,b,c]],Gcwa).

Gcwa = [[c]] ?

yes
```

`state_to_sn_sets`(+*State*, ?*Supports*)

for a Herbrand state *State* the list *Supports* of all support–for–negation sets SN(L) for literals L occuring in *State* is computed. Example:

```
| ?- state_to_sn_sets([[a,b,d],[b,c],[c,d]],Supports).

Supports =
  [[[a],[b,d]],[[b],[a,d],[c]],[[c],[b],[d]],[[d],[a,b],[c]]] ?

yes
```

`snd_snd_product`(+*State*,+*Supports1*,+*Supports2*, ?*Supports3*, ?*Egcwa*)

the state disjunctions of all reduced sn–sets in *Supports1* with reduced sn–sets in *Supports2* form the derived reduced sn–sets in *Supports3*, which are reduced w.r.t. *State*, empty reduced sn–sets are omitted, *Egcwa* is the set of all minimal conjunctions *E*, whose reduced sn–set $SN^r(E)$ is empty. Example:

```
| ?- snd_snd_product([[a,b],[b,c]],[[[a],[b]],[[b],[a],[c]]],
        [[[c],[b]]],Supports3,Egcwa).

Supports3 = [[[a,c],[b]]],
Egcwa = [[b,c]] ?

yes
```

`egcwa_operator(+`*Program*`,`*?Con_Egcwa*`)`

for a disjunctive logic program *Program* the set *Con_Egcwa* of all non–atomary conjunctions in the extended generalized closed–world–assumption of *Program* is computed. Example:

```
| ?- egcwa_operator([[c]-[a],[a,d]-[c],[a,b,c],[a,e]],Con_Egcwa).

Con_Egcwa = [[a,b],[a,d],[a,e],[b,c],[b,d]] ?

yes
```

`gcwa_operator(+`*Program*`,`*?Gcwa*`)`

for a disjunctive logic program *Program* the generalized closed–world–assumption *Gcwa* is computed. Example:

```
| ?- gcwa_operator([[c]-[a],[a,c]-[d],[d,c]],Gcwa).

Gcwa = [[a],[d]] ?

yes
```

8 Well–Founded Semantics

`sd_operator(`+*Program*,+*State_Pair1*,?*State_Pair2*`)`

applying the S^D–operator of the disjunctive normal logic program *Program* to the *State_Pair1* yields *State_Pair2*. Example:

```
| ?- sd_operator([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
        [e]-[p]-[f],[f]-[]-[e]],[[],[]],State_Pair2).

State_Pair2 = [[[a,b]],[]] ?

yes
```

`se_operator(`+*Program*,+*State_Pair1*,?*State_Pair2*`)`

applying the S^E–operator of the disjunctive normal logic program *Program* to the *State_Pair1* yields *State_Pair2*. Example:

```
| ?- se_operator([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
        [e]-[p]-[f],[f]-[]-[e]],[[[a,b]],[]],State_Pair2).

State_Pair2 = [[[a,b],[a,q],[b,t],[e,f]],
               [[a,q],[b,t],[e,f],[p],[q,t]]] ?

yes
```

`sed_operator(`+*Program*,+*State_Pair1*,?*State_Pair2*`)`

applying the S^{ED}–operator of the disjunctive normal logic program *Program* to the *State_Pair1* yields *State_Pair2*. Example:

```
| ?- sed_operator([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
        [e]-[p]-[f],[f]-[]-[e]],[[],[]],State_Pair2).

State_Pair2 = [[[a,b],[a,q],[b,t],[e,f]],
               [[a,q],[b,t],[e,f],[p],[q,t]]] ?

yes
```

`sd_fixpoint(`+*Program*,+*State_Pair1*,?*State_Pair2*`)`

the "disjunctive–well–founded" state pair *State_Pair2* is the result of a fixpoint iteration of the S^D–operator of the disjunctive normal logic program *Program* initialized with the state pair *State_Pair1*. Example:

```
| ?- sd_fixpoint([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
        [e]-[p]-[f],[f]-[]-[e]],[[],[]],State_Pair2).

State_Pair2 = [[[a,b]],[]] ?

yes
```

`se_fixpoint(`+*Program*,+*State_Pair1*,?*State_Pair2*`)`

the state pair *State_Pair2* is the result of a fixpoint iteration of the S^E–operator of the disjunctive normal logic program *Program* initialized with the state pair *State_Pair1*. Example:

```
| ?- se_fixpoint([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
        [e]-[p]-[f],[f]-[]-[e]],[[[a,b]],[]],State_Pair2).

State_Pair2 = [[[a,b],[a,q],[b,t],[e,f]],
               [[a,q],[b,t],[e,f],[p],[q,t]]] ?

yes
```

`sed_fixpoint(`+*Program*,+*State_Pair1*,?*State_Pair2*`)`

the "generalized–disjunctive–well–founded" state pair *State_Pair2* is the result of a fixpoint iteration of the S^{ED}–operator of the disjunctive normal logic program *Program* initialized with the state pair *State_Pair1*. Example:

```
| ?- sed_fixpoint([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
        [e]-[p]-[f],[f]-[]-[e]],[[],[]],State_Pair2).

State_Pair2 = [[[a,b],[a,q],[b,t],[f]],
               [[a,q],[b,t],[e],[p],[q,t]]] ?

yes
```

`dwfs_operator(`+*Program*,?*State_Pair*`)`

the "disjunctive–well–founded" state pair *State_Pair* of the disjunctive normal logic program *Program* is computed. Example:

```
| ?- dwfs_operator([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
        [e]-[p]-[f],[f]-[]-[e]],State_Pair).

State_Pair = [[[a,b]],[]] ?

yes
```

`gdwfs_operator(`+*Program*,?*State_Pair*`)`

the "generalized–disjunctive–well–founded" state pair *State_Pair* of the disjunctive normal logic program *Program* is computed. Example:

```
| ?- gdwfs_operator([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
        [e]-[p]-[f],[f]-[]-[e]],State_Pair).

State_Pair = [[[a,b],[a,q],[b,t],[f]],
               [[a,q],[b,t],[e],[p],[q,t]]] ?

yes
```

`sb_operator(+`*Program*`,+`*State1*`,?`*State2*`)`

applying Baral's S^B–operator of the disjunctive normal logic program *Program* to the general Herbrand state *State1* yields the general Herbrand state *State2*. Example:

```
| ?- sb_operator([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
          [e]-[p]-[f],[f]-[]-[e]],[],State2).

State2 = [[a,b],[~a,~b],[~a,~q],[~b,~t],[~e],[~p],[~q,~t]] ?

yes
```

`sb_fixpoint(+`*Program*`,+`*State1*`,?`*State2*`)`

computing the least fixpoint of Baral's S^B–operator of the disjunctive normal logic program *Program* for the general Herbrand state *State1* yields the general Herbrand state *State2*. Example:

```
| ?- sb_fixpoint([[p]-[q,t],[q]-[]-[a],[t]-[]-[b],[a,b],
          [e]-[p]-[f],[f]-[]-[e]],[],State2).

State2 = [[a,b],[a,q],[a,~t],[b,t],[b,~q],[f],[q,t],[q,~b],[t,~a],
           [~a,~b],[~a,~q],[~b,~t],[~e],[~p],[~q,~t]] ?

yes
```

`sb_fixpoint_tree(+`*Program*`,+`*Tree1*`,?`*Tree2*`)`

computing the least fixpoint of Baral's S^B–operator of the disjunctive normal logic program *Program* for the clause tree *Tree1* yields the clause tree *Tree2*. Example:

```
| ?- sb_fixpoint_tree([[p]-[]-[a],[p]-[]-[b],[a,b]],[],Tree2).

Tree2 = [[],[[p]],[[~a,~b]],[[a,b]]] ?

yes
```

`bdwfs_operator(+`*Program*`,?`*State*`)`

the "disjunctive–well–founded semantics" *State*, which is a general Herbrand state, of the disjunctive normal logic program *Program* is computed according to Baral. Example:

```
| ?- bdwfs_operator([[p]-[]-[a],[p]-[]-[b],[a,b]],State).

State = [[a,b],[p],[~a,~b]] ?

yes
```

`tsb_fixpoint`(+*Program*,+*State1*,?*State2*)

applying barals generalized negation-as-failure fixpoint operator of the disjunctive normal logic program *Program* to the general Herbrand state *State1* yields the general Herbrand state *State2*. Example:

```
| ?- tsb_fixpoint([[p]-[]-[a],[p]-[]-[b],[a,b]],
        [[~a,~b]],State2).

State2 = [[a,b],[p],[~a,~b]] ?

yes
```

`fsb_fixpoint`(+*Program*,+*State1*,?*State2*)

applying barals generalized negation-as-failure fixpoint operator of the disjunctive normal logic program *Program* to the general Herbrand state *State1* yields the negative Herbrand state *State2*. Example:

```
| ?- fsb_fixpoint([[p]-[]-[a],[p]-[]-[b],[a,b]],
        [[a,b]],State2).

State2 = [[~a,~b]] ?

yes
```

9 Input and Output of DisLog–Objects

dportray(dhs,+*State*)

pretty prints a disjunctive Herbrand state *State*. Example:

```
| ?- dportray(dhs,[[a,b,d],[a,b,e],[a,c]]).

 { a;b;d. a;b;e. a;c. }

yes
```

dportray(chs,+*State*)

pretty prints a conjunctive Herbrand state *State*. Example:

```
| ?- dportray(chs,[[a,b,d],[a,b,e],[a,c]]).

 { a,b,d. a,b,e. a,c. }

yes
```

dportray(ghs,+*State*)

pretty prints a general Herbrand state *State*. Example:

```
| ?- dportray(ghs,[[c,~d],[a,b],[~e,~f]]).

 { a;b.
   ~e;~f.
   c;~d. }

yes
```

dportray(lp,+*Program*)

pretty prints a disjunctive normal logic program *Program*. Example:

```
| ?- dportray(lp,[[p]-[q,t],[a,b],[e]-[p]-[f],[f]-[]-[e]]).

p :- q,t.
a;b.
e :- p,~f.
f :- ~e.

yes
```

`dportray(sp,+`*State_Pair*`)`

pretty prints a state pair *State_Pair*. Example:

```
| ?- dportray(sp,[[[a,b],[a,q]],[[b,t],[p]]]).

 T = { a;b. a;q. }
 F = { b,t. p. }

yes
```

`dportray(st,+`*State_Triple*`)`

pretty prints a state triple *State_Triple*. Example:

```
| ?- dportray(st,[[[a,b],[a,q]],[[e],[f]],[[b,t],[p]]]).

 T = { a;b. a;q. }
 U = { e. f. }
 F = { b,t. p. }

yes
```

`dportray(tree,+`*Tree*`)`

pretty prints a clause tree *Tree*. Example:

```
| ?- dportray(tree,[[a,b],[[c],[[d]],[[e]]],[[f]]]).

[a,b]
   [c]
      [d]
      [e]
   [f]

yes
```

`dportray(edge(state),+`*State*`)`

pretty prints a disjunctive Herbrand state *State* as its lattice by the tool *edge*. Example:

```
| ?- dportray(edge(state),[[a,b,d],[a,b,e],[a,c]]).

yes
```

`dportray(edge(tree),+`*Tree*`)`

pretty prints a clause tree *Tree* by the tool *edge*. Example:

```
| ?- dportray(edge(tree),[[a],[[c]],[[b],[[e]],[[d]]]]).

yes
```

`dportray(edge(graph),+`*Edges*`)`

pretty prints an undirected graph given by a set *Edges* of edges by the tool *edge*. Example:

```
| ?- dportray(edge(graph),[[a,b],[b,c],[b,d],[d,c]]).

yes
```

`dportray(edge(labeled_graph),+`*Labeled_Edges*`)`

pretty prints a labeled, directed graph given by a set *Labeled_Edges* of edges of the form *[Source,Destination,Label]* by the tool *edge*. Example:

```
| ?-
dportray(edge(labeled_graph),[[a,b,1],[b,c,2],[b,d,0],[d,c,8]]).

yes
```

`dportray(edge(dependency_graph),+`*[Nodes,E,P,N]*`)`

pretty prints the dependency graph of a disjunctive normal logic program given by the set *Nodes* of nodes and the sets *E,P,N* of equal, positive and negative edges, respectively, by the tool *edge*. Example:

```
| ?- dportray(edge(dependency_graph),[ [a,b,c,d],
         [[a,b]], [[a,c],[b,c]], [[a,d],[b,d]] ]).

yes
```

`dconsult(+`*Program_File*`, `*?Program*`)`

consults a disjunctive normal logic program *Program* from the file *Program_File*. Example:

```
| ?- dconsult('path/path_abcd',Program).

Program = [[a(a,b),a(a,c)],[a(b,d)],[a(c,d)],
    [p(X,Y)]-[a(X,Y)],[p(X,Y)]-[a(X,Z),p(Z,Y)]] ?

yes
```

`dsave(+`*Program*`,+`*Program_File*`)`

saves a disjunctive normal logic program *Program* into the file *Program_File*. Example:

```
| ?- dsave([[a(a,b),a(a,c)],[a(b,d)],[a(c,d)],[p(X,Y)]-[a(X,Y)],
         [p(X,Y)]-[a(X,Z),p(Z,Y)]],'path/path_abcd_2').

yes
```

`dlist(+`*Program_File*`)`

pretty prints the disjunctive normal logic program that is stored in the file *Program_File*. Example:

```
| ?- dlist('path/path_abcd').

p(X1,X2) :- a(X1,X3),p(X3,X2).
p(X1,X2) :- a(X1,X2).
a(a,b);a(a,c).
a(b,d).
a(c,d).

yes
```

Predicate Index