

$\mathcal{D}$ LAB  
**A declarative language bias for  
concept learning and  
knowledge discovery engines**

*Luc Dehaspe  
Luc De Raedt  
Report CW 214, July 8, 1996*

Department of Computing Science, K.U.Leuven

**Abstract**

We describe the principles and functionalities of  $\mathcal{D}$ LAB (Declarative Language Bias), which is an algorithm for defining syntactically and traversing efficiently hypothesis spaces in the context of concept learning and knowledge discovery tasks. Though  $\mathcal{D}$ LAB is designed for first-order languages it can also be used to constrain propositional concept spaces. In an appendix we document a  $\mathcal{D}$ LAB Prolog library available via anonymous ftp. The WWW-homepage of  $\mathcal{D}$ LAB can be found at URL

*<http://www.cs.kuleuven.ac.be/cwis/research/ai/Research/dlab-E.shtml>*

**Keywords** : declarative language bias, machine learning, knowledge discovery

# 1 Introduction

Concept learning algorithms in general demand the syntactic delineation of a language  $\mathcal{L}$  in which to search for the target concept. Even if we choose the search space  $\mathcal{L}$  to be finite, it is in most cases impractical to define  $\mathcal{L}$  extensionally. We then need a formalism to formulate an intensional syntactic definition of language  $\mathcal{L}$ .

The problem of making this type of syntactic bias a parameter to the concept learner has been studied extensively, especially in frameworks that use first-order clausal logic (see [12; 1] for an overview).

In this paper we present  $\mathcal{DLAB}$  (Declarative LAnguage Bias) as a machine learning system component that allows for a straightforward specification of syntactic bias.  $\mathcal{DLAB}$  extends the syntactic bias of Adé et al. [1] which in turn integrates the schemata of Emde et al. [8; 9], and the predicate sets of Bergadano et al. [3; 2]. At the end of this article, we give a more detailed account of the relation between  $\mathcal{DLAB}$  and other formalisms. Prior to that we present an overview of  $\mathcal{DLAB}$  in two stages. First, we discuss syntax, semantics and a refinement operator for  $\mathcal{DLAB}^\ominus$ , a subset of  $\mathcal{DLAB}$ . We then extend  $\mathcal{DLAB}^\ominus$  to full  $\mathcal{DLAB}$  and show both formalisms at work in the domain of finite element mesh design (see e.g. [7; 10]). A simplified implementation of  $\mathcal{DLAB}$  can be found in the text itself. In an appendix we document the more sophisticated  $\mathcal{DLAB}$  Prolog library which is available by ftp access.

## 2 $\mathcal{DLAB}^\ominus$ syntax

A  $\mathcal{DLAB}^\ominus$  grammar is a set of templates to which the clauses in search space  $\mathcal{L}$  conform. We first define  $\mathcal{DLAB}^\ominus$  grammars syntactically.

**Definition 1 ( $\mathcal{DLAB}^\ominus$  grammar)** *DGRAM is a  $\mathcal{DLAB}^\ominus$  grammar if and only if DGRAM is a set of  $\mathcal{DLAB}^\ominus$  templates.*

**Definition 2 ( $\mathcal{DLAB}^\ominus$  template)** *DTEMP is a  $\mathcal{DLAB}^\ominus$  template if and only if DTEMP =  $HA \leftarrow BA$ , where  $HA$  and  $BA$  are  $\mathcal{DLAB}^\ominus$  atoms.*

**Definition 3 ( $\mathcal{DLAB}^\ominus$  atom)** *DATOM is a  $\mathcal{DLAB}^\ominus$  atom if and only if DATOM is an atomic formula, or DATOM =  $Min \cdot \cdot Max : L$ , with  $Min$  and  $Max$  integers such that  $0 \leq Min \leq Max \leq length(L)$ , and  $L$  is a list of  $\mathcal{DLAB}^\ominus$  atoms.*

The following are a few examples of syntactically well-formed  $\mathcal{DLAB}^\ominus$  grammars:

- $\{say(Hello) \leftarrow to\_world\}$
- $\{false \leftarrow 0 \cdot \cdot 2 : [male(X), female(X)]\}$
- $\{2 \cdot \cdot 2 : [a(X), b(Y)] \leftarrow 1 \cdot \cdot 2 : [c(X), 0 \cdot \cdot 1 : [d(Y)]]\},$   
 $0 \cdot \cdot 1 : [n, 1 \cdot \cdot 2 : [o, 1 \cdot \cdot 1 : [p, q], r], s] \leftarrow true\}$

### 3 $\mathcal{DLAB}^\ominus$ semantics

The generation of a language  $\mathcal{L}$  given a  $\mathcal{DLAB}^\ominus$  grammar then basically consists of the (recursive) selection of all subsets of  $L$  with length within range  $Min \dots Max$  from each  $\mathcal{DLAB}^\ominus$  atom  $Min \dots Max : L$  in the grammar. To simplify our definition of a generation function we here introduce (and will continue to use) a special list notation for clauses, such that  $h_1 \vee \dots \vee h_n \leftarrow b_1 \wedge \dots \wedge b_m$  will be written as  $[h_1, \dots, h_n] \leftarrow [b_1, \dots, b_m]$ .

The following definitions provide a generator for  $\mathcal{DLAB}^\ominus$  grammars.

**Definition 4 (dlab\_generate(DGRAM))** *Let DGRAM be a  $\mathcal{DLAB}^\ominus$  grammar.*

$$dlab\_generate(DGRAM) = \{dlab1(HT) \leftarrow dlab1(BT) \mid (HT \leftarrow BT) \in DGRAM\}$$

where  $dlab1(DATOM)$  is a list of literals generated by the definite clause grammar [4; 14]  
 $dlab1$ :

$$dlab1(A) \longrightarrow [A], \{A \neq Min \dots Max : L\}. \quad (1)$$

$$dlab1(Min \dots Max : []) \longrightarrow \{Min \leq 0\}, []. \quad (2)$$

$$dlab1(Min \dots Max : [_|L]) \longrightarrow dlab1(Min \dots Max : L). \quad (3)$$

$$dlab1(Min \dots Max : [A|L]) \longrightarrow \{Max > 0\}, dlab1(A), \\ dlab1((Min - 1) \dots (Max - 1) : L). \quad (4)$$

From the previous definition we can derive a formula for calculating the number of clauses in  $dlab\_generate(DGRAM)$ , which corresponds to the size of the search space defined by a  $\mathcal{DLAB}^\ominus$  grammar  $DGRAM$ .

**Definition 5 (dlab\_size(DGRAM))** *Let  $DGRAM = \{H_1 \leftarrow B_1, \dots, H_m \leftarrow B_m\}$  be a  $\mathcal{DLAB}^\ominus$  grammar.*

$$dlab\_size(DGRAM) = \sum_{i=1}^m (ds(H_i) * ds(B_i))$$

$$ds(A) = 1, \text{ where } A \neq Min \dots Max : L$$

$$ds(Min \dots Max : [L_1, \dots, L_n]) = \sum_{k=Min}^{Max} e_k(ds(L_1), \dots, ds(L_n))$$

$$e_0(L) = 1$$

$$e_n(s_1, \dots, s_n) = \prod_{i=1}^n s_i$$

$$e_k(s_1, s_2, \dots, s_n) = e_k(s_2, \dots, s_n) + s_1 * e_{k-1}(s_2, \dots, s_n), \text{ with } k < n$$

**Proof** The first rule states that the size of language defined by a  $\mathcal{DLAB}^\ominus$  grammar equals the sum of the sizes of the languages defined by its individual  $\mathcal{DLAB}^\ominus$  templates. The latter size can be found by multiplying the number of headlists and the number of bodylists covered by the head and body  $\mathcal{DLAB}^\ominus$  atoms.

A  $\mathcal{DLAB}^\ominus$  atom which is not of the form  $Min \dots Max : L$  has a coverage of exactly one, as is expressed in the second rule.

Some more intricate combinatorics underlies the third rule. Basically, we select  $k$  objects from  $\{L_1, \dots, L_n\}$ , for each  $k$  in range  $Min \dots Max$ , hence the summation  $\sum_{k=Min}^{Max}$ . Inside this summation we would have the standard formula  $n!/k! * (n - k)!$  if our case had been an instance of the prototypical problem of finding all combinations, without replacement, of  $k$  marbles out of an urn with  $n$  marbles. This formula does not apply due to the fact that we rather have  $n$  urns ( $\{L_1, \dots, L_n\}$ ) with one or more marbles ( $ds(L_i) \geq 1$ ), and only combinations that use at most one marble from each urn should be counted. Therefore we need  $e_k(s_1, \dots, s_n)$ , where  $e_k$  is the elementary symmetric function [11] of degree  $k$  and the  $s_i$  are the numbers of marbles in each urn. The first base case of this recursive function accounts for the fact that there is only one way to select 0 objects. In the second base case, where  $k = n$ , one has to take an object from each urn. As for each urn there are  $s_i$  choices, the number of combinations equals the product of all  $s_i$ . The final recursive case applies if  $k < n$ . It is an addition of two terms, one for each possible operation on urn 1 (represented by  $s_1$ ). Either we skip this urn, and then we still have to select  $k$  elements from urns 2 to  $n$ . The number of such combinations is given by  $e_k(s_2, \dots, s_n)$ . Or else we do take a marble from the first urn. We then have to multiply  $s_1$ , the choices for the first urn, with  $e_{k-1}(s_2, \dots, s_n)$ , the number of  $k - 1$  order combinations of elements from urns 2 to  $n$ .  $\square$

A few illustrations of the definitions above should give a first idea of the expressive power of the relatively simple  $\mathcal{DLAB}^\ominus$  formalism. Given a  $\mathcal{DLAB}^\ominus$  atom  $Min \cdot \cdot Max : L$ , we can distinguish the following cases of special interest:

- **all subsets:**  $Min = 0, Max = length(L)$

$$DGRAM_1 = \{0 \cdot \cdot 1 : [human(X)] \leftarrow 0 \cdot \cdot 2 : [female(X), male(X)]\}$$

$$dlab\_generate(DGRAM_1) = \left\{ \begin{array}{l} [] \leftarrow [] \\ [] \leftarrow [male(X)] \\ [] \leftarrow [female(X)] \\ [] \leftarrow [female(X), male(X)] \\ [human(X)] \leftarrow [] \\ [human(X)] \leftarrow [male(X)] \\ [human(X)] \leftarrow [female(X)] \\ [human(X)] \leftarrow [female(X), male(X)] \end{array} \right.$$

$$dlab\_size(DGRAM_1) = 8$$

- **all non-empty subsets:**  $Min = 1, Max = length(L)$

$$DGRAM_2 = \{0 \cdot \cdot 1 : [human(X)] \leftarrow 1 \cdot \cdot 2 : [female(X), male(X)]\}$$

$$dlab\_generate(DGRAM_2) = \left\{ \begin{array}{l} [] \leftarrow [male(X)] \\ [] \leftarrow [female(X)] \\ [] \leftarrow [female(X), male(X)] \\ [human(X)] \leftarrow [male(X)] \\ [human(X)] \leftarrow [female(X)] \\ [human(X)] \leftarrow [female(X), male(X)] \end{array} \right.$$

$$dlab\_size(DGRAM_2) = 6$$

- **exclusive or:**  $Min = Max = 1$

$$DGRAM_3 = \{0 \cdot 1 : [human(X)] \leftarrow 1 \cdot 1 : [female(X), male(X)]\}$$

$$dlab\_generate(DGRAM_3) = \left\{ \begin{array}{l} [] \leftarrow [male(X)] \\ [] \leftarrow [female(X)] \\ [human(X)] \leftarrow [male(X)] \\ [human(X)] \leftarrow [female(X)] \end{array} \right.$$

$$dlab\_size(DGRAM_3) = 4$$

- **combined occurrence:**  $Min = Max = length(L)$

$$DGRAM_4 = \{1 \cdot 1 : [human(X)] \leftarrow \\ 0 \cdot 2 : [2 \cdot 2 : [female(X), is\_daughter(X)], \\ 2 \cdot 2 : [male(X), is\_son(X)]] \\ \}$$

$$dlab\_generate(DGRAM_4) = \left\{ \begin{array}{l} [human(X)] \leftarrow [] \\ [human(X)] \leftarrow [male(X), is\_son(X)] \\ [human(X)] \leftarrow [female(X), is\_daughter(X)] \\ [human(X)] \leftarrow [female(X), is\_daughter(X), \\ male(X), is\_son(X)] \end{array} \right.$$

$$dlab\_size(DGRAM_4) = 4$$

## 4 A $\mathcal{DLAB}^\ominus$ refinement operator

A refinement operator for  $\mathcal{DLAB}^\ominus$  is based on the observation that clauses  $c$  in  $dlab\_generate(DGRAM)$  are defined by a sequence of subset selections from  $\mathcal{DLAB}^\ominus$  atoms occurring in  $DGRAM$ . If we enlarge one of these subsets then the clause  $c' \supseteq c$  defined by the new sequence is a specialization of  $c$  under  $\theta$ -subsumption. If we somehow enlarge one subset in a minimal way, then  $c'$  will be a refinement, i.e. a maximally general specialization

of  $c^1$ . To implement this idea we adapt the definite clause grammar *dlab1* in Definition 4 in three steps.

First, in order to formalize the above notion of a sequence of subset selections, we add to *dlab1* an extra argument we will refer to as the  $\mathcal{DLAB}^\ominus$  path. The  $\mathcal{DLAB}^\ominus$  path is meant to keep track of applications of Rules (3) and (4) in *dlab1*. The application of these rules determines whether the first  $\mathcal{DLAB}^\ominus$  atom in list  $L$  of  $Min \cdot \cdot Max : L$  is either skipped (Rule (3)) or included in the subset (Rule (4)).

**Definition 6 ( $\mathcal{DLAB}^\ominus$  path)** *Let  $DATOM$  be a  $\mathcal{DLAB}^\ominus$  atom, and  $C$  a list of literals generated by *dlab1*( $DATOM$ ).  $DPATH$  is a  $\mathcal{DLAB}^\ominus$  path of  $C$  with regard to  $DATOM$  if and only if*

- $DATOM \neq Min \cdot \cdot Max : L$  and  $DPATH = DATOM$  or
- $DATOM = Min \cdot \cdot Max : [L_1, \dots, L_n]$  and  $DPATH = [P_1, \dots, P_n]$ , with, for each  $P_i \in DPATH$ ,
  - $P_i = *$  and  $L_i$  is excluded during generation of  $C$  (application of Rule (3)/(7)), or
  - $P_i$  is the  $\mathcal{DLAB}^\ominus$  path of  $C$  with regard to  $\mathcal{DLAB}^\ominus$  atom  $L_i$  and  $L_i$  is included during generation of  $C$  (application of Rule (4)/(8))

For instance,

$DATOM = 0 \cdot \cdot 2 : [human(X), 1 \cdot \cdot 1 : [female(X), male(X)]]$	
$C = dlab1(DATOM)$	$\mathcal{DLAB}^\ominus$ path of $C$ with regard to $DATOM$
[]	[*,*]
[male(X)]	[*,[*,male(X)]]
[female(X)]	[*,[female(X),*]]
[human(X)]	[human(X),*]
[human(X),male(X)]	[human(X),[*],male(X)]]
[human(X),female(X)]	[human(X),[female(X),*]]

The following is an adaptation of *dlab1*, with the  $\mathcal{DLAB}^\ominus$  path in the second argument position.

$$dlab2(A, A) \longrightarrow [A], \{A \neq Min \cdot \cdot Max : L\}. \quad (5)$$

$$dlab2(Min \cdot \cdot Max : [], []) \longrightarrow \{Min \leq 0\}, []. \quad (6)$$

$$dlab2(Min \cdot \cdot Max : [_|L], [*|Y]) \longrightarrow dlab2(Min \cdot \cdot Max : L, Y). \quad (7)$$

$$dlab2(Min \cdot \cdot Max : [A|L], [X|Y]) \longrightarrow \{Max > 0\}, dlab2(A, X), \\ dlab2((Min - 1) \cdot \cdot (Max - 1) : L, Y). \quad (8)$$

In a second step, we can use the  $\mathcal{DLAB}^\ominus$  path  $DP$  of a list of literals  $C$  to generate supersets of  $C$ . Every  $*$  in  $DP$  marks an occasion for extending  $C$ . In terms of Definition 6:

<sup>1</sup>Depending on the  $\mathcal{DLAB}^\ominus$  grammar, this refinement (under  $\theta$ -subsumption) can be proper or not.

we have to locate a  $P_i = *$  in  $DP$  indicating the corresponding  $\mathcal{DLAB}^\ominus$  atom  $L_i$  is excluded during generation of  $C$ , and then include  $L_i$  during generation of supersets  $C'$  of  $C$ . Definite clause grammar  $d labs$  does that, and moreover returns the  $\mathcal{DLAB}^\ominus$  path  $DP'$  of  $C'$  in the third argument position.

$$d labs(\_ \cdot \cdot Max : [], [], []) \longrightarrow []. \quad (9)$$

$$d labs(\_ \cdot \cdot Max : [A|L], [*|Y], [X|Z]) \longrightarrow \{Max > 0\}, dlab2(A, X), \\ d labs(\_ \cdot \cdot (Max - 1) : L, Y, Z). \quad (10)$$

$$d labs(\_ \cdot \cdot Max : [-|L], [*|Y], [*|Z]) \longrightarrow d labs(\_ \cdot \cdot Max : L, Y, Z). \quad (11)$$

$$d labs(\_ \cdot \cdot Max : [A|L], [P|Y], [Q|Z]) \longrightarrow \{P \neq *, Max > 0\}, d labs(A, P, Q), \\ d labs(\_ \cdot \cdot (Max - 1) : L, Y, Z). \quad (12)$$

$$d labs(\_ \cdot \cdot Max : [A|L], [X|Y], [X|Z]) \longrightarrow \{X \neq *, Max > 0\}, dlab2(A, X), \\ d labs(\_ \cdot \cdot (Max - 1) : L, Y, Z). \quad (13)$$

Notice how in Rule (10) of  $d labs$  the previously excluded  $A$  (cf. the  $*$  in  $Arg2$ ) is now included with the call of  $dlab2(A, X)$ . For instance,

$DATOM = 0 \cdot \cdot 3 : [human(X), female(X), male(X)]$ $C = [female(X)]$ $DP = [*, female(X), *]$	
$C' = d labs(DATOM, DP, DP')$	$DP'$
$[human(X), female(X), male(X)]$ $[human(X), female(X)]$ $[female(X), male(X)]$ $[female(X)]$	$[human(X), female(X), male(X)]$ $[human(X), female(X), *]$ $[*, female(X), male(X)]$ $[*, female(X), *]$

The rules in  $d labs$  can be used to find specializations  $c'$  of  $c$ . As we want our refinement operator to generate only maximally general specializations of  $c$ , a final adaptation of  $d labs$  is required such that it will generate only smallest supersets of  $C$ . Roughly stated, exactly one  $*$  in the  $\mathcal{DLAB}^\ominus$  path  $DP$  of a list of literals  $C$  should be expanded, and then only in a minimal way. The first requirement, again in terms of Definition 6, says that we should locate exactly one  $P_i = *$  in  $DP$ , and then include  $L_i$  during generation of supersets of  $C$ . The second requirement says that the inclusion of  $L_i$  should be minimal in the sense that the corresponding  $\mathcal{DLAB}^\ominus$  path  $P'_i$  should contain the maximally allowed number of  $*$ 's. For this we need a modified version of  $dlab2$ , that, given a  $\mathcal{DLAB}^\ominus$  atom  $Min \cdot \cdot Max : L$ , will only generate subsets of length  $Min$ . The first requirement is realized in  $dlabr$  by eliminating some recursive calls, the second by initialization the newly included  $\mathcal{DLAB}^\ominus$  atom  $A$  with  $dlabi$  instead of  $dlab2$ .

$$dlabr(Min \cdot \cdot Max : [A|L], [*|Y], [X|Y]) \longrightarrow \{not(dlab\_optimal, member(E, Y), E \neq *)\}, \\ \{Max > 0\}, dlabi(A, X), \\ dlab2((Min - 1) \cdot \cdot (Max - 1) : L, Y). \quad (14)$$

$$dlabr(Min \cdot \cdot Max : [-|L], [*|Y], [*|Z]) \longrightarrow dlabr(Min \cdot \cdot Max : L, Y, Z). \quad (15)$$

$$\begin{aligned}
dlabr(\text{Min} \cdot \text{Max} : [A|L], [X|Z], [Y|Z]) &\longrightarrow \{X \neq *, \text{Max} > 0\}, dlabr(A, X, Y), \\
& \quad dlab2((\text{Min} - 1) \cdot (\text{Max} - 1) : L, Z). \tag{16}
\end{aligned}$$

$$\begin{aligned}
dlabr(\text{Min} \cdot \text{Max} : [A|L], [X|Y], [X|Z]) &\longrightarrow \{X \neq *, \text{Max} > 0\}, dlab2(A, X), \\
& \quad dlabr((\text{Min} - 1) \cdot (\text{Max} - 1) : L, Y, Z). \tag{17}
\end{aligned}$$

$$dlabi(A, A) \longrightarrow [A], \{\text{not}(A = \text{Min} \cdot \text{Max} : L)\}. \tag{18}$$

$$dlabi(0 \cdot \_ : [], []) \longrightarrow []. \tag{19}$$

$$\begin{aligned}
dlabi(\text{Min} \cdot \_ : [A|L], [X|Y]) &\longrightarrow dlabi(A, X), \\
& \quad dlabi((\text{Min} - 1) \cdot \_ : L, Y). \tag{20}
\end{aligned}$$

$$dlabi(\text{Min} \cdot \_ : [_|L], [*|Y]) \longrightarrow dlabi(\text{Min} \cdot \_ : L, Y). \tag{21}$$

Notice that Rule 14 of *dlabr* contains an extra initial condition:

$$\text{not}(dlab\_optimal, \text{member}(E, Y), E \neq *)$$

A call to *dlab\_optimal* should succeed, if we want the refinement operator to be optimal, and fail otherwise.

**Definition 7** A refinement operator  $\rho$  (with transitive closure  $\rho^*$ ) is optimal if and only if  $\forall c, c_1, c_2 \in \mathcal{L} : c \in \rho^*(c_1)$  and  $c \in \rho^*(c_2) \rightarrow c_1 \in \rho^*(c_2)$  or  $c_2 \in \rho^*(c_1)$ .

The extra condition ensures that when working in optimal mode, the refinement operator will never expand  $*$ 's to the left of already expanded  $*$ 's. For instance,

$DATOM = 0 \cdot 3 : [human(X), female(X), male(X)]$ $C = [female(X)]$ $DP = [*, female(X), *]$		
<i>dlab_optimal</i>	$C' = dlabr(DATOM, DP, DP')$	$DP'$
false	$[human(X), female(X)]$ $[female(X), male(X)]$	$[human(X), female(X), *]$ $[*, female(X), male(X)]$
true	$[female(X), male(X)]$	$[*, female(X), male(X)]$

To further enforce optimality we have to make sure refinement of the head of a clause blocks all future refinements of the body, or vice-versa<sup>2</sup>.

We can now formulate the definition of a  $\mathcal{DLAB}^\ominus$  refinement operator based on the twelve definite clause grammar rules of *dlabr*, *dlabi*, and *dlab2*.

**Definition 8** (*dlab\_refine(DINFO, c)*) Given

- $\mathcal{DLAB}^\ominus$  template  $HA \leftarrow BA$ ,
- clause  $c = H \leftarrow B$ , with  $c \in dlab\_generate(\{HA \leftarrow HB\})$

<sup>2</sup>In fact, both measures merely prevent the same couple of  $\mathcal{DLAB}^\ominus$  paths (one for the head, one for the body) from being generated more than once. In case the list of body- or headliterals of a single clause corresponds to  $n > 1$   $\mathcal{DLAB}^\ominus$  paths, e.g.  $[male(X)]$  given  $\mathcal{DLAB}^\ominus$  atom  $1 \cdot 1 : [male(X), male(X), male(X)]$  ( $n = 3$ ),  $\mathcal{DLAB}^\ominus$  is likely to generate this clause  $n$  times. Part of the responsibility for optimality is thus left to the  $\mathcal{DLAB}^\ominus$  user.

- $HP$  a  $\mathcal{DLAB}^\ominus$  path of  $H$  with regard to  $HA$ ,
- $BP$  a  $\mathcal{DLAB}^\ominus$  path of  $B$  with regard to  $BA$ ,
- $DINFO = (HA, HP, BA, BP)$ ,

If  $dlab\_optimal = false$

$$dlab\_refine(DINFO, c) = dlab\_refh(DINFO, c) \cup dlab\_refb(DINFO, c)$$

If  $dlab\_optimal = true$

$$dlab\_refine(DINFO, c) = dlab\_refh((HA, HP, [], []), c) \cup dlab\_refb(DINFO, c)$$

$$dlab\_refh((HA, HP, BA, BP), H \leftarrow B) =$$

$$\{((HA, HP', BA, BP), H' \leftarrow B) \mid H' = dlabr(HA, HP, HP')\}$$

$$dlab\_refb((HA, HP, BA, BP), H \leftarrow B) =$$

$$\{((HA, HP, BA, BP'), H \leftarrow B') \mid B' = dlabr(BA, BP, BP')\}$$

An initialisation function that returns the most general clauses in  $\mathcal{L}$  completes the  $\mathcal{DLAB}^\ominus$  refinement operator:

**Definition 9 (dlab\_initialize(DGRAM))** *Let DGRAM be a  $\mathcal{DLAB}^\ominus$  grammar, then the following function returns the top nodes in the refinement lattice:*

$$dlab\_initialize(DGRAM) = \{dlab\_refh(dlab\_refb(DINFO, \square)) \mid \\ (HA \leftarrow BA) \in DGRAM, \\ DINFO = (0 \cdot 1 : [HA], [*], 0 \cdot 1 : [BA], [*])\}$$

The procedure  $dlab\_generate\_df(DGRAM)$  is added in Figure 1 as an illustration of how  $dlab\_initialize$  and  $dlab\_refine$  can be integrated to print the refinement lattice in depth first order. We also list the output of this procedure for  $DGRAM = \{0 \cdot 2 : [a(X), b(X)] \leftarrow 1 \cdot 2 : [c(X), d(X)]\}$  and with  $dlab\_optimal = true$ <sup>3</sup>

$$\begin{aligned} &(HA, [[*, *], BA, [[c(X), *]], false \leftarrow c(X)) \\ &\quad (HA, [[a(X), *], [], []], a(X) \leftarrow c(X)) \\ &\quad\quad (HA, [[a(X), b(X)], [], []], a(X), b(X) \leftarrow c(X)) \\ &\quad\quad (HA, [[*, b(X)], [], []], b(X) \leftarrow c(X)) \\ &\quad\quad (HA, [[*, *], BA, [[c(X), d(X)]], false \leftarrow c(X), d(X)) \\ &\quad\quad\quad (HA, [[a(X), *], BA, [[c(X), d(X)]], a(X) \leftarrow c(X), d(X)) \\ &\quad\quad\quad\quad (HA, [[a(X), b(X)], BA, [[c(X), d(X)]], a(X), b(X) \leftarrow c(X), d(X)) \end{aligned}$$

---

<sup>3</sup>We use the following abbreviations:

$$\begin{aligned} HA &= 0 \cdot 1 : [0 \cdot 2 : [a(X), b(X)]] \\ BA &= 0 \cdot 1 : [1 \cdot 2 : [c(X), d(X)]] \end{aligned}$$

---

```

procedure dlab_generate_df(DGRAM)
  inputs : DGRAM:  $\mathcal{DLAB}^\ominus$  grammar

for all (DINFO, c)  $\in$  dlab_initialize(DGRAM) do
  dlab_generate_df(0, (DINFO, c))
endfor

  procedure dlab_generate_df(DEPTH, (DINFO, c))
    inputs : DEPTH: depth, (DINFO, c): node

    indent DEPTH, write (DINFO, c), begin new line
    for all (DINFO, c)  $\in$  dlab_refine(DINFO, c) do
      dlab_generate_df(DEPTH + 1, (DINFO, c))
    endfor endprocedure
  endprocedure

```

---

Figure 1: Prototypical integration of *dlab\_initialize*(*DGRAM*) and *dlab\_refine*(*DINFO*, *c*)

$$\begin{aligned}
 & (HA, [[*, b(X)]], BA, [[c(X), d(X)]], b(X) \leftarrow c(X), d(X) \\
 & (HA, [[*, *]], BA, [[*, d(X)]], false \leftarrow d(X) \\
 & (HA, [[a(X), *]], BA, [[*, d(X)]], a(X) \leftarrow d(X) \\
 & (HA, [[a(X), b(X)]], BA, [[*, d(X)]], a(X), b(X) \leftarrow d(X) \\
 & (HA, [[*, b(X)]], BA, [[*, d(X)]], b(X) \leftarrow d(X)
 \end{aligned}$$

## 5 Extended $\mathcal{DLAB}^\ominus$ : $\mathcal{DLAB}$

Although  $\mathcal{DLAB}^\ominus$  grammars as they are introduced in the previous section are purely declarative, their readability rapidly deteriorates as their complexity rises. In an extended version  $\mathcal{DLAB}$  mainly two features have been added to alleviate this problem: second order predicate variables, and subsets on the term level. We will now define  $\mathcal{DLAB}$  and conversion functions that map grammars from the  $\mathcal{DLAB}$  to the  $\mathcal{DLAB}^\ominus$  format.

**Definition 10** ( *$\mathcal{DLAB}$  grammar*) *DGRAM* is a  $\mathcal{DLAB}$  grammar if and only if  $DGRAM = (DTEMP S, DVARS)$ , where *DTEMP S* is a set of  $\mathcal{DLAB}$  templates, and *DVARS* is a set of  $\mathcal{DLAB}$  variables.

**Definition 11** ( *$\mathcal{DLAB}$  variable*) *DVAR* is a  $\mathcal{DLAB}$  variable if and only if  $DVAR = dlab\_variable(P_0, Min \dots Max, [P_1, \dots, P_n])$ , where *Min* and *Max* are integers with  $0 \leq Min \leq Max \leq n$ , and for all  $P_i \in \{P_0, \dots, P_n\}$ ,  $P_i$  is a predicate symbol or a function symbol.

---

```

function ConvertToDLAB⊖1
  inputs : (DTEMPS, DVARS): DLAB grammar
  outputs : equivalent DLAB grammar (DTEMPS' ,  $\emptyset$ )

DTEMPS' := DTEMPS
CONTINUE := true
while CONTINUE do
  find  $P(t_1, \dots, t_n)$  in DTEMPS' , with  $0 \leq n$ ,
    for which  $dlab\_variable(P, Min \cdot Max, [P_1, \dots, P_m]) \in DVARS
  if found
  then replace  $P(t_1, \dots, t_n)$  in DTEMPS' with
     $Min \cdot Max : [P_1(t_1, \dots, t_n), \dots, P_m(t_1, \dots, t_n)]$ 
  else CONTINUE := false
endwhile
endfunction$ 
```

Figure 2: Removal of *DLAB* variables from *DLAB* grammars

---

**Definition 12** (*DLAB template*) *DTEMP* is a *DLAB template* if and only if  $DTEMP = HA \leftarrow BA$ , where *HA* and *BA* are *DLAB atoms*.

**Definition 13** (*DLAB atom*) *DATOM* is a *DLAB atom* if and only if

- $DATOM = P(t_1, \dots, t_n)$ , where *P* is a predicate symbol  $t_1, \dots, t_n$  ( $0 \leq n$ ) is a sequence of *n DLAB terms*, or
- $DATOM = Min \cdot Max : L$ , where *Min* and *Max* are integers with  $0 \leq Min \leq Max \leq length(L)$ , and *L* is a list of *DLAB atoms*.

**Definition 14** (*DLAB term*) *DTERM* is a *DLAB term* if and only if

- *DTERM* is a variable symbol, or
- $DTERM = F(t_1, \dots, t_n)$ , where *F* is a function symbol and  $t_1, \dots, t_n$  ( $0 \leq n$ ) is a sequence of *n DLAB terms*, or
- $DTERM = Min \cdot Max : L$ , where *Min* and *Max* are integers with  $0 \leq Min \leq Max \leq length(L)$ , and *L* is a list of *DLAB terms*.

The function *convert\_to\_dlab\_minus*(*DGRAM*) required for converting grammars from *DLAB* to the *DLAB*<sup>⊖</sup> format is defined with two helpfunctions: one to remove (and expand) the *DLAB* variables (see Figure 2), and one to move the subsets on the termlevel outside to the atomlevel (see Figure 3).

---

```

function ConvertToDLAB⊖2
  inputs : (DTEMPS, DVARS): DLAB grammar
  outputs : equivalent DLAB grammar without subsets on termlevel

DTEMPS' := DTEMPS
CONTINUE := true
while CONTINUE do
  find  $P(t_1, \dots, t_i, \text{Min} \cdot \text{Max} : [L_1, \dots, L_n], t_{i+2}, \dots, t_m)$  in DTEMPS',
    with for all  $t_j \in \{t_1, \dots, t_i\}, t_j \neq A \cdot B : C$ 
  if found
  then replace  $P(t_1, \dots, t_i, \text{Min} \cdot \text{Max} : [L_1, \dots, L_n], t_{i+2}, \dots, t_m)$  in DTEMPS' with
     $\text{Min} \cdot \text{Max} : [P(t_1, \dots, t_i, L_1, t_{i+2}, \dots, t_m), \dots, P(t_1, \dots, t_i, L_n, t_{i+2}, \dots, t_m)]$ 
  else CONTINUE := false
endwhile
endfunction

```

Figure 3: Removal of subsets on termlevel from *DLAB* grammars

---

**Definition 15 (convert\_to\_dlab\_minus(DGRAM))** *Let DGRAM be a DLAB grammar and  $\text{ConvertToDLAB}^{\ominus 2}(\text{ConvertToDLAB}^{\ominus 1}(DGRAM)) = (DGRAM^{\ominus}, \emptyset)$ , then  $DGRAM^{\ominus}$  is the  $\text{DLAB}^{\ominus}$  grammar equivalent to  $DGRAM$ .*

## 6 $\text{DLAB}^{\ominus}$ and *DLAB* at work in the FE mesh domain

We demonstrate the power of  $\text{DLAB}^{\ominus}$  and *DLAB*<sup>4</sup> with the finite-element mesh design application frequently used for testing and comparing ILP systems. The data for this application describe a number of structures. For each edge of each structure the knowledge base contains:

- the number of subedges (resolution), e.g. *mesh(a1, 17)*
- edge type, e.g. *long(a1)*, *short(a17)*
- boundary condition, e.g. *fixed(a1)*
- loading, e.g. *not\_loaded(a1)*
- topology, e.g. *neighbour(a16, a17)*

The purpose is to find rules that predict the number of subedges given characteristics of an edge and related edges within the same structure. *MeshDGRAM*<sup>⊖</sup> is a first  $\text{DLAB}^{\ominus}$  gram-

---

<sup>4</sup>As a minor additional extension we will also allow *DLAB* atoms of the type *Min* · *len* : *L* or *len* · *len* : *L*, where *len* is a constant symbol that abbreviates *length(L)*.

mar for discovering rules in which characteristics of only one edge are taken into account.

```

MeshDGRAM⊖ =
  {1 ..1 : [mesh(E, 1), mesh(E, 2), mesh(E, 3), mesh(E, 4), mesh(E, 5),
            mesh(E, 6), mesh(E, 7), mesh(E, 8), mesh(E, 9), mesh(E, 10),
            mesh(E, 11), mesh(E, 12), mesh(E, 17)]
    ]
  ←
  1 ..3 : [
    1 ..1 : [long(E), usual(E), short(E), circuit(E), half_circuit(E),
              quarter_circuit(E), short_for_hole(E), long_for_hole(E),
              circuit_hole(E), half_circuit_hole(E), not_important(E))
            ]
    1 ..1 : [free(E), one_side_fixed(E), two_side_fixed(E), fixed(E)]
    1 ..1 : [not_loaded(E), one_side_loaded(E), two_side_loaded(E), cont_loaded(E)]
  ]
}

dlab_size(MeshDGRAM) = 3887

```

The DLAB format allows for  $MeshDGRAM_1$ , an equivalent, but more convenient version of  $MeshDGRAM^{\ominus}$ .

```

MeshDGRAM1 = (MeshDTEMP1, MeshDVARS1)

MeshDTEMP1 =
  {mesh(E, resolution) ← 1 ..len : [type(E), boundary(E), loading(E)]}

MeshDVARS1 =
  {dlab_variable(resolution, 1 ..1, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 17]),
    dlab_variable(type, 1 ..1, [long, usual, short, circuit, half_circuit,
                                quarter_circuit, short_for_hole, long_for_hole,
                                circuit_hole, half_circuit_hole, not_important]),
    dlab_variable(boundary, 1 ..1, [free, one_side_fixed, two_side_fixed, fixed]),
    dlab_variable(loading, 1 ..1, [not_loaded, one_side_loaded, two_side_loaded, cont_loaded])
  }

dlab_size(MeshDGRAM1) = 3887

```

A next version extends the search space defined by  $MeshDGRAM_1$  with rules that include characteristics of a second edge in the structure.

$$MeshDGRAM_2 = (MeshDTemps_2, MeshDVars_2)$$

$$MeshDTemps_2 =$$

$$\{mesh(E, resolution)$$

$$\leftarrow$$

$$1 \cdot len : [type(E), boundary(E), loading(E),$$

$$mesh(E2, resolution), topology(E, E2),$$

$$type(E2), boundary(E2), loading(E2)]$$

$$\}$$

$$MeshDVars_2 = MeshDVars_1 \cup \{dlab\_variable(topology, 1 \cdot 1, [opposite, neighbour])\}$$

$$dlab\_size(MeshDGRAM_2) = 4.91 * 10^7$$

Suppose now that we are only interested in antecedents that say at least something about the type, boundary conditions, loading or resolution of the edges that occur in the rule. Moreover, if two edges occur, the antecedent should specify their topology. We can encode this background knowledge in the *DLAB* grammar as follows.

$$MeshDGRAM_3 = (MeshDTemps_3, MeshDVars_2)$$

$$MeshDTemps_3 =$$

$$\{mesh(E, resolution)$$

$$\leftarrow$$

$$len \cdot len : [1 \cdot len : [type(E), boundary(E), loading(E)],$$

$$0 \cdot len : [len \cdot len : [topology(E, E2),$$

$$1 \cdot len : [mesh(E2, resolution),$$

$$type(E2), boundary(E2), loading(E2)]$$

$$]$$

$$]$$

$$\}$$

$$dlab\_size(MeshDGRAM_3) = 3.26 * 10^7$$

## 7 Relation of $\mathcal{D}_{\text{LAB}}$ to other syntactic bias formalisms

We conclude with a brief situation of  $\mathcal{D}_{\text{LAB}}$  against some alternative declarative<sup>5</sup> syntactic bias formalisms that have been used for ILP. A detailed formalization would require a complete introduction into each of the alternatives, and would be outside the scope of this paper. We therefore restrict ourselves to illustrations of the, mostly rather obvious, links.

### Clausemodels of Adé et al. [1]

Closest to  $\mathcal{D}_{\text{LAB}}$  are the clausemodels proposed in [1]. Clausemodels are expressions of the form  $Head \leftarrow Body, BodySet$  whose conversion to  $\mathcal{D}_{\text{LAB}}$  templates is illustrated in the following example.

With the predicates  $male/1$ ,  $female/1$ ,  $parent/2$  in the background theory, the following clausemodel:

$$\{grandfather(X, Y) \leftarrow P(Y), Q(X, Z), \{parent(\{X, Z\}, Y)\}\}$$

corresponds to  $DGRAM_5$ :

$$DGRAM_5 = (DTEMP_5, DVARS_5)$$

$$DTEMP_5 = \{grandfather(X, Y) \leftarrow len \cdot len : [P(Y), Q(X, Z), \\ 0 \cdot len : [parent(1 \cdot len : [X, Z], Y)] \\ ] \\ \}$$

$$DVARS_5 = \{dlab\_variable(P, 1 \cdot 1, [male, female]), dlab\_variable(Q, 1 \cdot 1, [parent])\}$$

Generally speaking, clausemodels are special cases of  $\mathcal{D}_{\text{LAB}}$  templates in which the choice of  $Min$  and  $Max$  is restricted. In fact, due to these constraints, none of the previous example  $\mathcal{D}_{\text{LAB}}$  grammars can be translated to clausemodels without adding ad-hoc predicates to the background theory.

### Schemata of Emde et al. [9], and the predicate sets of Bergadano et al. [3]

As discussed in [1], schemata and predicate sets as used in MOBAL and the FILP system respectively, are special cases of clausemodels, and thus indirectly of  $\mathcal{D}_{\text{LAB}}$  templates.

---

<sup>5</sup>More procedural approaches to syntactic bias specifications use parameters such as the maximal variable depth or term level to control the complexity of the concept language, cf. [6; 13]. Parametrized languages should be considered complementary to  $\mathcal{D}_{\text{LAB}}$ , in the sense that the same parameters trivially define (a series of)  $\mathcal{D}_{\text{LAB}}$  grammars.

## Antecedent description grammars of Cohen [5]

An antecedent description grammar, as used in GRENDDEL, is in essence a definite clause grammar that generates the antecedents of clauses in  $\mathcal{L}$ . The following example taken from [5] is used in the context of learning when a chess position containing two kings and one rook is illegal.

$$\begin{aligned} & \text{goal\_formula}(\text{illegal}(A, B, C, D, E, F)). \\ & \text{body}(\text{illegal}(A, B, C, D, E, F)) \longrightarrow \text{rels}(A, B, C, D, E, F) \\ & \text{rels}(A, B, C, D, E, F) \longrightarrow []. \\ & \text{rels}(A, B, C, D, E, F) \longrightarrow \text{rel}(A, B, C, D, E, F), \text{rels}(A, B, C, D, E, F). \\ & \text{rel}(A, B, C, D, E, F) \longrightarrow \text{pred}(X, Y) \\ & \quad \text{where } \text{member}(X, [A, B, C, D, E, F]), \text{member}(Y, [A, B, C, D, E, F]).. \\ & \text{pred}(X, Y) \longrightarrow [X = Y]. \\ & \text{pred}(X, Y) \longrightarrow [\neg X = Y]. \\ & \text{pred}(X, Y) \longrightarrow [\text{adj}(X, Y)]. \\ & \text{pred}(X, Y) \longrightarrow [\neg \text{adj}(X, Y)]. \\ & \text{pred}(X, Y) \longrightarrow [\text{less\_than}(X, Y)]. \\ & \text{pred}(X, Y) \longrightarrow [\neg \text{less\_than}(X, Y)]. \end{aligned}$$

For this example, we can write down an equivalent antecedent description grammar, using *dlab1* (see Definition 4)

$$\begin{aligned} & \text{goal\_formula}(\text{illegal}(A, B, C, D, E, F)). \\ & \text{body}(\text{illegal}(A, B, C, D, E, F)) \longrightarrow \text{dlab1}(BA_6). \end{aligned}$$

where  $BA_6$  is the  $\mathcal{DLAB}^\ominus$  bodyatom of the  $\mathcal{DLAB}^\ominus$  grammar equivalent to  $\mathcal{DLAB}$  grammar  $DGRAM_6$ :

$$\begin{aligned} DGRAM_6 &= (DTEMP S_6, DVARS_6) \\ DTEMP S_6 &= \{\text{illegal}(A, B, C, D, E, F) \leftarrow \text{pred}(1 \cdot \text{len} : [A, B, C, D, E, F], \\ & \quad 1 \cdot \text{len} : [A, B, C, D, E, F])\} \\ DVARS_6 &= \text{dlab\_variable}(\text{pred}, 0 \cdot \text{len}, [=, \neq, \text{adj}, \neg \text{adj}, \text{less\_than}, \neg \text{less\_than}]) \end{aligned}$$

In general however a conversion of antecedent description grammars to  $\mathcal{DLAB}$  is not always possible<sup>6</sup>. As suggested by the example,  $\mathcal{DLAB}$  is a special case of antecedent description

---

<sup>6</sup>A clear case where this conversion is impossible occurs when the antecedent description grammar generates an infinite language.

grammars, in which the grammar rules are restricted to those of *dlab1*. In other words, using *DLAB*, we work with a fixed and hidden definite clause grammar *dlab1* that takes our *DLAB* grammar as its single argument. In that sense *DLAB* is a higher order formalism based on the lower order antecedent description grammar, and from a practical view point both formalisms compare as do higher and lower order programming languages.

## Acknowledgements

This work is part of the Esprit Basic Research project no. 6020 on Inductive Logic Programming. Luc Dehaspe is paid by the Esprit Basic Research Action ILP (project 6020), and co-financed by the “Vlaamse Gemeenschap” through contract nr.93/014. Luc De Raedt is supported by the Belgian National Fund for Scientific Research.

The authors would like to thank Wim Van Laer and Timothy Chow for their elegant solutions of the *dlab\_size(DGRAM)* combinatorics problem.

## References

- [1] H. Adé, L. De Raedt, and M. Bruynooghe. Declarative Bias for Specific-To-General ILP Systems. *Machine Learning*, 1995. To appear.
- [2] F. Bergadano. Towards an inductive logic programming language. Technical Report ESPRIT project no. 6020 ILP Deliverable TO1, Computer Science Department, University of Torino, 1993.
- [3] F. Bergadano and D. Gunetti. An interactive system to learn functional logic programs. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1044–1049. Morgan Kaufmann, 1993.
- [4] W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin, 1981.
- [5] W.W. Cohen. Grammatically biased learning: learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68:303–366, 1994.
- [6] L. De Raedt. *Interactive Theory Revision: an Inductive Logic Programming Approach*. Academic Press, 1992.
- [7] B. Dolšák and S. Muggleton. The application of Inductive Logic Programming to finite element mesh design. In S. Muggleton, editor, *Inductive logic programming*, pages 453–472. Academic Press, 1992.
- [8] W. Emde, C.U. Habel, and C.R. Rollinger. The discovery of the equator or concept driven learning. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 455–458. Morgan Kaufmann, 1983.
- [9] J-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive logic programming*, pages 335–359. Academic Press, 1992.

- [10] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
- [11] I.G. MacDonald. *Symmetric functions and Hall polynomials*. Clarendon Oxford, 1979.
- [12] S. Muggleton. Predicate invention and utility. *Journal for Experimental and Theoretical Artificial Intelligence*, 1994. To appear.
- [13] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st conference on algorithmic learning theory*, pages 368–381. Ohmsma, Tokyo, Japan, 1990.
- [14] Leon Sterling and Ehud Shapiro. *The art of Prolog*. The MIT Press, 1986.

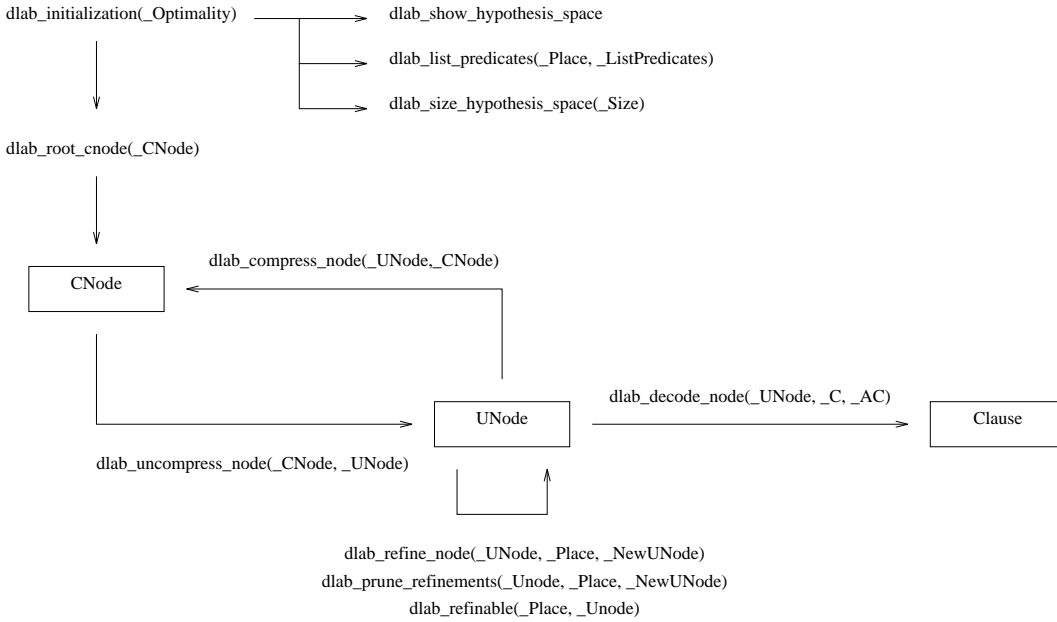


Figure 4: An overview of the predicates in the  $\mathcal{DLAB}$  library

## A The $\mathcal{DLAB}$ library

In this section we introduce a distributed Prolog<sup>7</sup> library<sup>8</sup> with predicates for manipulating  $\mathcal{DLAB}$  data structures. In this section we introduce the relevant predicates in this library.

The main data structure manipulated by  $\mathcal{DLAB}$  predicates is the uncompressed node (UNode), which roughly corresponds to the  $\mathcal{DLAB}^\ominus$  path (cf. Definition 6). Each UNode represents a clause in the hypothesis space. Refinement and decoding (to clause format) operations all take UNodes as input. In case a large queue of UNodes has to be managed it might be advantageous to add a compression step and transform each UNode into a CNode (compressed node) before storing it. Figure 4 captures the interdependencies between the  $\mathcal{DLAB}$  predicates further described below.

### `dlab_initialization/1`

`dlab_initialization(_Optimality)`

*arg1: ground: atom*

Initializes  $\mathcal{DLAB}$  in optimal or non-optimal mode, depending on whether atom *arg1* is *optimal* or *nonoptimal*.

### `dlab_root_cnode/1`

`dlab_root_cnode(_CNode)`

*arg1: free: atom*

<sup>7</sup>Currently ProLog by BIM 4.0.5 and SICStus Prolog 2.1 are supported.

<sup>8</sup>This library is available at *ftp://ftp.cs.kuleuven.ac.be/pub/logic-prgm/ilp/dlab*.

Returns in *arg1* the compressed node corresponding to the empty clause. Uncompress this node and refine both body and head to obtain the/a top UNode in the hypothesis space.

### **dlab\_uncompress\_node/2**

**dlab\_uncompress\_node(\_CNode, unode(\_Nr, \_HeadUNode, \_BodyUNode))**  
*arg1: ground: atom*  
*arg2: free: term*

Returns in UNode *arg2* the uncompressed equivalent to CNode *arg1*. The UNode is a term *unode/3*. *\_Nr* is the sequence number of a template in the *DLAB* grammar. *\_HeadUNode* and *\_BodyUNode* are *DLAB* paths w.r.t. to this template.

### **dlab\_compress\_node/2**

**dlab\_compress\_node(unode(\_Nr, \_HeadUNode, \_BodyUNode), \_CNode)**  
*arg1: ground: term*  
*arg2: free: atom*

Returns in CNode *arg2* the compressed equivalent to UNode *arg1*.

### **dlab\_decode\_node/3**

**dlab\_decode\_node(\_UNode, \_Decoded, \_GroundDecoded)**  
*arg1: ground: term*  
*arg2: free: term*  
*arg3: free: term*

Decodes the UNode in *arg1* to *arg2* and *arg3*. *\_Decoded* is of the form *\_HeadList : -\_BodyList*, where *\_HeadList* (*\_BodyList*) is a list of head (body) literals of the clause identified by *arg1*. *\_GroundDecoded* is of the same form, but here variables of *\_Decoded* are instantiated with the names they have in the original *dlab\_template*. *\_GroundDecoded* is useful for output of rules in a more readable format.

### **dlab\_refine\_node/3**

**dlab\_refine\_node(\_UNode, \_Place, \_RefinedUNode)**  
*arg1: ground: term*  
*arg2: any: atom*  
*arg3: free: term*

Refines the UNode in *arg1* at place *arg2*. The result of refinement is returned in UNode *arg3*. *\_Place* is either *body* or *head*.

### **dlab\_refinable/2**

**dlab\_refinable(\_Place, \_UNode)**  
*arg1: any: atom*  
*arg2: ground: term*

Returns true if the UNode in *arg2* can still be refined at place *arg1*. *\_Place* is either *body* or *head*.

### **dlab\_prune\_refinements/3**

**dlab\_prune\_refinements(\_UNode, \_Place, \_PrunedUNode)**

*arg1: ground: term*

*arg2: any: atom*

*arg3: free: term*

Prunes the UNode in *arg1* at place *arg2*. The result of pruning is returned in UNode *arg3*, such that *dlab\_refinable(\_Place, \_PrunedUNode)* will fail. *\_Place* is either *body* or *head*.

### **dlab\_size\_hypothesis\_space/1**

**dlab\_size\_hypothesis\_space(\_Size)**

*arg1: free: integer*

Returns in *arg1* the number of clauses in the hypothesis space defined by the *DLAB* grammar.

### **dlab\_list\_predicates/2**

**dlab\_list\_predicates(\_Place, \_ListPredicates)**

*arg1: any: atom*

*arg2: free: list*

Returns in *arg2* the list of predicates occurring in place *arg1* of clauses in the hypothesis space. List *\_ListPredicates* contains couples (*\_PredicateName, \_Arity*).

### **dlab\_show\_hypothesis\_space/0**

**dlab\_show\_hypothesis\_space**

Prints the hypothesis space defined by the *DLAB* grammar depth first. The definition of this predicate should also be seen as a sample integration of predicates in the *DLAB* library.

## B Defining a concept language with the $\mathcal{D}$ LAB library

When using the  $\mathcal{D}$ LAB library the following additions to the definition of  $\mathcal{D}$ LAB (Definition 10 and following) should be taken into account.

Each  $\mathcal{D}$ LAB template is represented as a fact  $dlab\_template(\_Template)$ , where  $\_Template$  is a string surrounded by single quotes. This string should conform to the syntax of  $\mathcal{D}$ LAB templates with the following minor modifications:

- The range  $Min \cdot \cdot Max$  is written as  $Min - Max$ .
- The leftarrow  $\leftarrow$  separating head and body is written as  $\leftarrow$ .

Each  $\mathcal{D}$ LAB variable is represented as a fact  $dlab\_variable(\_P0, \_Min-\_Max, \_ListNames)$ . An important constraint here is that  $\_P0$  and the members of list  $\_ListNames$  should be atoms. Moreover,  $\_P0$  should be an atom that need not be quoted.

As a final addition,  $\mathcal{D}$ LAB macros are accepted. A  $\mathcal{D}$ LAB macro is represented as a fact  $dlab\_macro(\_SubString, \_NewSubString)$ . In the first stage of initialization  $\mathcal{D}$ LAB will scan the  $\mathcal{D}$ LAB templates for substrings  $\_SubString$  and replace these with  $\_NewSubString$ .

We here add the final version of the FE mesh design grammar in a format that can be processed by the  $\mathcal{D}$ LAB library.

```
dlab_template(' mesh(E,resolution)
    <--
    len-len:[ 1-len: [type(E),boundary(E),loading(E)],
              0-len: [len-len: [topology(E,E2),
                              1-len: [mesh(E2,resolution),type(E2),
                                      boundary(E2),loading(E2)]
                              ]
              ]
    ]
    '),
dlab_variable(resolution,1-1,[1,2,3,4,5,6,7,8,9,10,11,12,17]).

dlab_variable(type,1-1,[long,usual,short,circuit,half_circuit,quarter_circuit,
    short_for_hole,long_for_hole,circuit_hole,
    half_circuit_hole,not_important]).

dlab_variable(boundary,1-1,[free,one_side_fixed,two_side_fixed,fixed]).

dlab_variable(loading,1-1,[not_loaded,one_side_loaded,two_side_loaded,
    cont_loaded]).

dlab_variable(topology,1-1,[neighbour,opposite]).
```