DLAB
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 DLAB (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 DLAB is designed for first-order languages it can also be used to constrain propositional concept spaces. In an appendix we document a DLAB Prolog library available via anonymous ftp. The WWW-homepage of DLAB can be found at URL


Keywords: declarative language bias, machine learning, knowledge discovery
1 Introduction

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

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 $\text{DLAB}$ (Declarative LAnguage Bias) as a machine learning system component that allows for a straightforward specification of syntactic bias. $\text{DLAB}$ 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 $\text{DLAB}$ and other formalisms. Prior to that we present an overview of $\text{DLAB}$ in two stages. First, we discuss syntax, semantics and a refinement operator for $\text{DLAB}^\circ$, a subset of $\text{DLAB}$. We then extend $\text{DLAB}^\circ$ to full $\text{DLAB}$ and show both formalisms at work in the domain of finite element mesh design (see e.g. [7; 10]). A simplified implementation of $\text{DLAB}$ can be found in the text itself. In an appendix we document the more sophisticated $\text{DLAB}$ Prolog library which is available by ftp access.

2 $\text{DLAB}^\circ$ syntax

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

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

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

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

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

- \{say(\text{Hello}) \leftarrow \text{to\_world}\}
- \{false \leftarrow 0 \cdots 2 : [\text{male}(X), \text{female}(X)]\}
- \{2 \cdots 2 : [a(X), b(Y)] \leftarrow 1 \cdots 2 : [c(X), 0 \cdots 1 : [d(Y)]], 0 \cdots 1 : [n, 1 \cdots 2 : [o, 1 \cdots 1 : [p, q], r], s] \leftarrow \text{true}\}
3 $\mathcal{DLAB}^\oplus$ semantics

The generation of a language $L$ given a $\mathcal{DLAB}^\oplus$ grammar then basically consists of the (recursive) selection of all subsets of $L$ with length within range $\text{Min} \ldots \text{Max}$ from each $\mathcal{DLAB}^\oplus$ atom $\text{Min} \ldots \text{Max} : L$ 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 $h_1 \lor \ldots \lor h_n \leftarrow b_1 \land \ldots \land b_m$ will be written as $[h_1, \ldots, h_n] \leftarrow [b_1, \ldots, b_m]$.

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

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

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

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

$$\begin{align*}
\text{dlab1}(A) & \rightarrow [A], \{A \neq \text{Min} \ldots \text{Max} : L\}. \\
\text{dlab1}(\text{Min} \ldots \text{Max} : []) & \rightarrow \{\text{Min} \leq 0\}, []. \\
\text{dlab1}(\text{Min} \ldots \text{Max} : [L]) & \rightarrow \text{dlab1}(\text{Min} \ldots \text{Max} : L). \\
\text{dlab1}(\text{Min} \ldots \text{Max} : [A|L]) & \rightarrow \{\text{Max} > 0\}, \text{dlab1}(A), \text{dlab1}((\text{Min} - 1) \cdot (\text{Max} - 1) : L).
\end{align*}$$

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

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

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

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

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

$e_0(L) = 1$

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

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

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

A $\mathcal{DLAB}^\oplus$ atom which is not of the form $\text{Min} \ldots \text{Max} : L$ 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 \( \{L_1, \ldots, L_n\} \), for each \( k \) in range \( \text{Min} \ldots \text{Max} \), hence the summation \( \sum_{k=\text{Min}}^{\text{Max}} \). 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 (\( \{L_1, \ldots, L_n\} \)) with one or more marbles (\( d(s(L_i) \geq 1) \)), and only combinations that use at most one marble from each urn should be counted. Therefore we need \( e_k(s_1, \ldots, s_n) \), 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 \( e_k(s_2, \ldots, s_n) \). 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 \( e_{k-1}(s_2, \ldots, s_n) \), the number of \( k - 1 \) order combinations of elements from urns 2 to \( n \). □

A few illustrations of the definitions above should give a first idea of the expressive power of the relatively simple \( \text{Dlab}^\oplus \) formalism. Given a \( \text{Dlab}^\oplus \) atom \( \text{Min} \ldots \text{Max} : L \), we can distinguish the following cases of special interest:

- **all subsets**: \( \text{Min} = 0, \text{Max} = \text{length}(L) \)

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

\[
dlab_{\text{generate}}(DGRAM_1) = \{
\text{} \leftarrow \text{}
, \text{} \leftarrow \text{[male}(X)]
, \text{} \leftarrow \text{[female}(X)]
, \text{} \leftarrow \text{[female}(X), \text{male}(X)]
, \text{human}(X) \leftarrow \text{}
, \text{human}(X) \leftarrow \text{[male}(X)]
, \text{human}(X) \leftarrow \text{[female}(X)]
, \text{human}(X) \leftarrow \text{[female}(X), \text{male}(X)]
\}
\]

\[
dlab_{\text{size}}(DGRAM_1) = 8
\]

- **all non-empty subsets**: \( \text{Min} = 1, \text{Max} = \text{length}(L) \)

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

3
\[ dl_{\text{generate}}(DGRAM_2) = \begin{align*}
&\{ [] \leftarrow [\text{male}(X)] \\
&\quad \leftarrow [\text{female}(X)] \\
&\quad \leftarrow [\text{female}(X),\text{male}(X)] \\
&\quad [\text{human}(X)] \leftarrow [\text{male}(X)] \\
&\quad [\text{human}(X)] \leftarrow [\text{female}(X)] \\
&\quad [\text{human}(X)] \leftarrow [\text{female}(X),\text{male}(X)]
\end{align*} \]

\[ dl_{\text{size}}(DGRAM_2) = 6 \]

- \textbf{exclusive or:} \( \text{Min} = \text{Max} = 1 \)

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

\[ dl_{\text{generate}}(DGRAM_3) = \begin{align*}
&\{ [] \leftarrow [\text{male}(X)] \\
&\quad \leftarrow [\text{female}(X)] \\
&\quad [\text{human}(X)] \leftarrow [\text{male}(X)] \\
&\quad [\text{human}(X)] \leftarrow [\text{female}(X)]
\end{align*} \]

\[ dl_{\text{size}}(DGRAM_3) = 4 \]

- \textbf{combined occurrence:} \( \text{Min} = \text{Max} = \text{length}(L) \)

\[ DGRAM_4 = \{ 1 \cdots 1 : [\text{human}(X)] \leftarrow \\
&\quad 0 \cdots 2 : [\text{female}(X),\text{isDaughter}(X)], \\
&\quad 2 \cdots 2 : [\text{male}(X),\text{isSon}(X)] \\
&\quad \} \]

\[ dl_{\text{generate}}(DGRAM_4) = \begin{align*}
&\{ [\text{human}(X)] \leftarrow [ ] \\
&\quad [\text{human}(X)] \leftarrow [\text{male}(X),\text{isSon}(X)] \\
&\quad [\text{human}(X)] \leftarrow [\text{female}(X),\text{isDaughter}(X)] \\
&\quad [\text{human}(X)] \leftarrow [\text{female}(X),\text{isDaughter}(X),
\quad \text{male}(X),\text{isSon}(X)]
\end{align*} \]

\[ dl_{\text{size}}(DGRAM_4) = 4 \]

4 \textbf{A }DLAB^\Theta \textbf{ refinement operator}

A refinement operator for \( DLAB^\Theta \) is based on the observation that clauses \( c \) in\n\( dl_{\text{generate}}(DGRAM) \) are defined by a sequence of subset selections from \( DLAB^\Theta \) atoms\noccurring in \( DGRAM \). If we enlarge one of these subsets then the clause \( c' \supseteq c \) defined by\nthe new sequence is a specialization of \( c \) under \( \theta \)-subsumption. If we somehow enlarge one\nsubset in a minimal way, then \( c' \) will be a refinement, i.e. a maximally general specialization
of $c^i$. To implement this idea we adapt the definite clause grammar $\text{dlab1}$ in Definition 4 in three steps.

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

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

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

For instance,

<table>
<thead>
<tr>
<th>$\text{DATOM} = 0 \cdots 2 : [\text{human}(X), 1 \cdots 1 : [\text{female}(X), \text{male}(X)]]$</th>
<th>$\text{dlab1}(\text{DATOM})$</th>
<th>$\text{DLAB}^Θ$ path of $C$ with regard to $\text{DATOM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[\ ]$</td>
<td>$[<em>,</em>]$</td>
<td>$[[<em>,</em>,\text{male}(X)]]$</td>
</tr>
<tr>
<td>$[\text{male}(X)]$</td>
<td>$[<em>,[</em>,\text{male}(X)]]$</td>
<td>$[\text{human}(X), [*]]$</td>
</tr>
<tr>
<td>$[\text{female}(X)]$</td>
<td>$[*,[\text{female}(X), *]]$</td>
<td>$[\text{human}(X), [*]]$</td>
</tr>
<tr>
<td>$[\text{human}(X)]$</td>
<td>$[\text{human}(X), [*]]$</td>
<td>$[\text{human}(X), [*, \text{male}(X)]]$</td>
</tr>
<tr>
<td>$[\text{human}(X), \text{male}(X)]$</td>
<td>$[\text{human}(X), [*, \text{male}(X)]]$</td>
<td>$[\text{human}(X), [\text{female}(X), *]]$</td>
</tr>
<tr>
<td>$[\text{human}(X), \text{female}(X)]$</td>
<td>$[\text{human}(X), [\text{female}(X), *]]$</td>
<td></td>
</tr>
</tbody>
</table>

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

$$\text{dlab2}(A, A) \longrightarrow [A, \{A \neq \text{Min} \cdots \text{Max} : L\}].$$ (5)
$$\text{dlab2}(\text{Min} \cdots \text{Max} : [], []) \longrightarrow \{\text{Min} \leq 0\}, [].$$ (6)
$$\text{dlab2}(\text{Min} \cdots \text{Max} : [L], [*Y]) \longrightarrow \text{dlab2}(\text{Min} \cdots \text{Max} : L, Y).$$ (7)
$$\text{dlab2}(\text{Min} \cdots \text{Max} : [A L], [X Y]) \longrightarrow \{\text{Max} > 0\}, \text{dlab2}(A, X),$$
$$\text{dlab2}(\text{Min} - 1 \cdots (\text{Max} - 1) : L, Y).$$ (8)

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

---

1. Depending on the $\text{DLAB}^Θ$ grammar, this refinement (under $θ$-subsumption) can be proper or not.
we have to locate a $P_i = \ast$ in $DP$ indicating the corresponding $D\text{LAB}^\Box$ atom $L_i$ is excluded during generation of $C'$, and then include $L_i$ during generation of supersets $C'$ of $C$. Define clause grammar $dlabs$ does that, and moreover returns the $D\text{LAB}^\Box$ path $DP'$ of $C'$ in the third argument position.

\[
dlabs(\text{Max} : A | [ ], [ ], [ ]) \rightarrow [ ]. \quad (9)
\]
\[
dlabs(\text{Max} : A | [A | L], *[Y], [X | Z]) \rightarrow \{ \text{Max} > 0 \}, dlabs(A, X),
\]
\[
dlabs(\text{Max} : A | [A | L], *[Y], [X | Z]) \rightarrow dlabs(\text{Max} - 1) : L, Y, Z). \quad (10)
\]
\[
dlabs(\text{Max} : A | [A | L], [P | Y], [Q | Z]) \rightarrow \{ P \neq \ast, \text{Max} > 0 \}, dlabs(A, P, Q),
\]
\[
dlabs(\text{Max} : A | [A | L], [P | Y], [Q | Z]) \rightarrow dlabs(\text{Max} - 1) : L, Y, Z). \quad (11)
\]
\[
dlabs(\text{Max} : A | [A | L], [X | Y], [X | Z]) \rightarrow \{ X \neq \ast, \text{Max} > 0 \}, dlabs(A, X),
\]
\[
dlabs(\text{Max} : A | [A | L], [X | Y], [X | Z]) \rightarrow dlabs(\text{Max} - 1) : L, Y, Z). \quad (12)
\]

Notice how in Rule (10) of $dlabs$ the previously excluded $A$ (cf. the $\ast$ in Arg2) is now included with the call of $dlabs(A, X)$. For instance,

\[
\begin{array}{|c|c|}
\hline
DATOM = 0 \cdot 3 : [\text{human}(X), \text{female}(X), \text{male}(X)] & C = [\text{female}(X)] \\
C' = dlabs(DATOM, DP, DP') & DP = \ast, \text{female}(X), \ast \\
[\text{human}(X), \text{female}(X), \text{male}(X)] & [\text{human}(X), \text{female}(X), \ast] \\
[\text{human}(X), \text{female}(X)] & [*; \text{female}(X), \text{male}(X)] \\
[\text{female}(X)] & [*; \text{female}(X), \ast] \\
\hline
\end{array}
\]

The rules in $dlabs$ 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 $dlabs$ is required such that it will generate only smallest supersets of $C$. Roughly stated, exactly one $\ast$ in the $D\text{LAB}^\Box$ 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 $P_i = \ast$ 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 $D\text{LAB}^\Box$ path $P_i'$ should contain the maximally allowed number of $\ast$'s. For this we need a modified version of $dlab2$, that, given a $D\text{LAB}^\Box$ atom $\text{Min} \cdot \text{Max} : A$, will only generate subsets of length $\text{Min}$. The first requirement is realized in $dlab$ by eliminating some recursive calls, the second by initialization the newly included $D\text{LAB}^\Box$ atom $A$ with $dlabi$ instead of $dlab2$.

\[
dlabr(\text{Min} \cdot \text{Max} : A | L, *[Y], [X | Y]) \rightarrow \{ \text{not}(\text{dlab_optimal, member}(E, Y), E \neq \ast) \},
\]
\[
\{ \text{Max} > 0 \}, dlabi(A, X),
\]
\[
dlabr(\text{Min} \cdot \text{Max} : A | L, *[Y], [X | Y]) \rightarrow \quad dlabs(\text{Min} \cdot \text{Max} \cdot L, Y, Z). \quad (14)
\]
\[
dlabr(\text{Min} \cdot \text{Max} : A | L, *[Y], [X | Y]) \rightarrow \quad dlabs(\text{Min} \cdot \text{Max} : L, Y, Z). \quad (15)
\]
\[ \text{dlabr}(\text{Min} \cdot \text{Max} : [A|L], [X|Z], [Y|Z]) \quad \rightarrow \quad \{ X \neq *, \text{Max} > 0 \}, \text{dlabr}(A, X, Y), \]
\[ \text{dlabr}((\text{Min} - 1) \cdot (\text{Max} - 1) : L, Z). \] (16)

\[ \text{dlabr}(\text{Min} \cdot \text{Max} : [A|L], [X|Y], [X|Z]) \quad \rightarrow \quad \{ X \neq *, \text{Max} > 0 \}, \text{dlabr}(A, X), \]
\[ \text{dlabr}((\text{Min} - 1) \cdot (\text{Max} - 1) : L, Y, Z). \] (17)

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

\[ \text{dlabi}(0 \cdot \cdot \cdot : [], []) \quad \rightarrow \quad \emptyset. \] (19)

\[ \text{dlabi}((\text{Min} \cdot \cdot : [A|L], [X|Y]) \quad \rightarrow \quad \text{dlabi}(A, X), \]
\[ \text{dlabi}((\text{Min} - 1) \cdot \cdot : L, Y). \] (20)

\[ \text{dlabi}((\text{Min} \cdot \cdot : [], [*|Y]) \quad \rightarrow \quad \text{dlabi}(\text{Min} \cdot \cdot : L, Y). \] (21)

Notice that Rule 14 of \text{dlabr} contains an extra initial condition:

\[ \text{not(}\text{dlab_optimal}, \text{member}(E, Y), E \neq *) \]

A call to \text{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,

<table>
<thead>
<tr>
<th>( \text{DATOM} \quad = \quad 0 \cdot 3 : [\text{human}(X), \text{female}(X), \text{male}(X)] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C \quad = \quad [\text{female}(X)] )</td>
</tr>
<tr>
<td>( DP = [*, \text{female}(X), *} )</td>
</tr>
<tr>
<td>\textbf{dlab_optimal}</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>true</td>
</tr>
</tbody>
</table>

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-versa².

We can now formulate the definition of a \( \mathcal{DLAB}^\circ \) refinement operator based on the twelve definite clause grammar rules of \text{dlabr}, \text{dlabi}, and \text{dlabr2}.

**Definition 8** (\text{dlab_refine}(\text{DINFO}, c)) Given

- \( \mathcal{DLAB}^\circ \) **template** \( HA \leftarrow BA \),
- **clause** \( c = H \leftarrow B \), with \( c \in \text{dlab_generate}([HA \leftarrow HB]) \)

²In fact, both measures merely prevent the same couple of \( \mathcal{DLAB}^\circ \) 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 \( n > 1 \) \( \mathcal{DLAB}^\circ \) paths, e.g. \( [\text{male}(X)] \) given \( \mathcal{DLAB}^\circ \) atom \( 1 \cdot 1 : [\text{male}(X), \text{male}(X), \text{male}(X)] \) \( (n = 3) \), \( \mathcal{DLAB}^\circ \) is likely to generate this clause \( n \) times. Part of the responsibility for optimality is thus left to the \( \mathcal{DLAB}^\circ \) user.
• \( HP \) a \( \mathcal{DLAB}^\circ \) path of \( H \) with regard to \( HA \),
• \( BP \) a \( \mathcal{DLAB}^\circ \) path of \( B \) with regard to \( BA \),
• \( DINFO = (HA, HP, BA, BP) \),

If \( \text{lab}_{\text{optimal}} = \text{false} \)
\[
\text{lab}_{\text{refine}}(DINFO, c) = \text{lab}_{\text{refh}}(DINFO, c) \cup \text{lab}_{\text{refb}}(DINFO, c)
\]

If \( \text{lab}_{\text{optimal}} = \text{true} \)
\[
\text{lab}_{\text{refine}}(DINFO, c) = \text{lab}_{\text{refh}}((HA, HP, BA, BP), H \leftarrow B) = \\
\{((HA, HP', BA, BP'), H' \leftarrow B)|H' = \text{labr}(HA, HP, HP')\}
\]
\[
\text{lab}_{\text{refb}}((HA, HP, BA, BP), H \leftarrow B) = \\
\{((HA, HP, BA, BP'), H \leftarrow B')|B' = \text{labr}(BA, BP, BP')\}
\]

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

**Definition 9 \((\text{lab}_{\text{initialize}}(DGRAM))\) .** Let \( DGRAM \) be a \( \mathcal{DLAB}^\circ \) grammar, then the following function returns the top nodes in the refinement lattice:

\[
\text{lab}_{\text{initialize}}(DGRAM) = \{\text{lab}_{\text{refh}}(\text{lab}_{\text{refb}}(DINFO, \square))| \\
(HA \leftarrow BA) \in DGRAM, \\
DINFO = (0 \cdot 1 : [HA], [\ast], 0 \cdot 1 : [BA], [\ast])\}
\]

The procedure \( \text{lab}_{\text{generate_df}}(DGRAM) \) is added in Figure 1 as an illustration of how \( \text{lab}_{\text{initialize}} \) and \( \text{lab}_{\text{refine}} \) 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 \( \text{lab}_{\text{optimal}} = \text{true}^3 \)

\[
(HA, [[\ast, \ast]], BA, [[c(X), \ast]]), \text{false} \leftarrow c(X) \\
(HA, [[a(X), \ast]], [[], []], a(X) \leftarrow c(X) \\
(HA, [[a(X), b(X)]], [[], []], a(X), b(X) \leftarrow c(X) \\
(HA, [[\ast, \ast]], BA, [[c(X), d(X)]], \text{false} \leftarrow c(X), d(X) \\
(HA, [[a(X), \ast]], BA, [[c(X), d(X)]], a(X) \leftarrow c(X), d(X) \\
(HA, [[a(X), b(X)]], BA, [[c(X), d(X)]], a(X), b(X) \leftarrow c(X), d(X)
\]

\[^3\text{We use the following abbreviations:}\]

\[
HA = 0 \cdot 1 : [0 \cdot 2 : [a(X), b(X)] \\
BA = 0 \cdot 1 : [1 \cdot 2 : [c(X), d(X)]
\]

8
procedure \texttt{dlab\_generate\_df(DGRAM)}
\hspace*{0.5cm}inputs : \texttt{DGRAM}: DLAB\textsuperscript{8} grammar

for all \((\texttt{DINFO}, c) \in \texttt{dlab\_initialize(DGRAM)}\) do
\texttt{dlab\_generate\_df(0, (DINFO, c))}
endfor

procedure \texttt{dlab\_generate\_df(DEPTH, (DINFO, c))}
\hspace*{0.5cm}inputs : \texttt{DEPTH}: depth, \(\texttt{(DINFO, c)}\): node

indent \texttt{DEPTH}, write \((\texttt{DINFO, c)}\), begin new line
for all \((\texttt{DINFO, c) \in \texttt{dlab\_refine(DINFO, c)}}\) do
\texttt{dlab\_generate\_df(DEPTH + 1, (DINFO, c))}
endfor
endprocedure

endprocedure

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

\[(HA, [[*, b(X)]]], BA, [[c(X), d(X)]]], b(X) \leftarrow c(X), d(X)\]
\[(HA, [[*, *]], BA, [[*, d(X)]]], \text{false} \leftarrow d(X)\]
\[(HA, [[a(X), *]], BA, [[*, d(X)]]], a(X) \leftarrow d(X)\]
\[(HA, [[a(X), b(X)]]], BA, [[*, d(X)]]], a(X), b(X) \leftarrow d(X)\]
\[(HA, [[*, b(X)]]], BA, [[*, d(X)]]], b(X) \leftarrow d(X)\]

5 Extended DLAB\textsuperscript{8}: DLAB

Although DLAB\textsuperscript{8} grammars as they are introduced in the previous section are purely declarative, their readability rapidly deteriorates as their complexity rises. In an extended version 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 DLAB and conversion functions that map grammars from the DLAB to the DLAB\textsuperscript{8} format.

Definition 10 (DLAB grammar) DGRAM is a DLAB grammar if and only if \(\text{DGRAM} = (\text{DTEMPLS}, \text{DVARS})\), where DTEMPLS is a set of DLAB templates, and DVARS is a set of DLAB variables.

Definition 11 (DLAB variable) DVAR is a DLAB variable if and only if \(\text{DVAR} = \text{dlab\_variable}(P_0, \text{Min} \cdot \text{Max}, [P_1, \ldots, P_n])\), where Min and Max are integers with \(0 \leq \text{Min} \leq \text{Max} \leq n\), and for all \(P_i \in \{P_0, \ldots, P_n\}\), \(P_i\) is a predicate symbol or a function symbol.
function ConvertToDLAB\textsuperscript{\textcopyright 1}

\textbf{inputs} : \((D\textsc{temps}, D\textsc{vars})\): DLAB grammar

\textbf{outputs} : equivalent DLAB grammar \((D\textsc{temps}', 0)\)

\footnotesize

\textbf{DTEMS}' := DTEMS

CONTINUE := true

\textbf{while} CONTINUE \textbf{do}

\hspace{1em} find \(P(t_1, \ldots, t_n)\) in \(D\textsc{temps}'\), with \(0 \leq n\),

\hspace{2em} for which \(\text{dlab\_variable}(P, \text{Min} \cdots \text{Max}, [P_1, \ldots, P_m]) \in \text{Dvars}\)

\hspace{1em} \textbf{if} found

\hspace{2em} \textbf{then} replace \(P(t_1, \ldots, t_n)\) in \(D\textsc{temps}'\) with

\hspace{3em} \(\text{Min} \cdots \text{Max} : [P_1(t_1, \ldots, t_n), \ldots, P_m(t_1, \ldots, t_n)]\)

\hspace{1em} \textbf{else} CONTINUE := false

\textbf{endwhile}

endfunction

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{Removal of DLAB variables from DLAB grammars}
\end{figure}

\textbf{Definition 12 (DLAB template)} \(D\textsc{temp}\) is a DLAB template if and only if \(D\textsc{temp} = HA \leftarrow BA\), where \(HA\) and \(BA\) are DLAB atoms.

\textbf{Definition 13 (DLAB atom)} \(DATOM\) is a DLAB atom if and only if

- \(DATOM = P(t_1, \ldots, t_n)\), where \(P\) is a predicate symbol \(t_1, \ldots, t_n\) \((0 \leq n)\) is a sequence of \(n\) DLAB terms, or
- \(DATOM = \text{Min} \cdots \text{Max} : L\), where \(\text{Min}\) and \(\text{Max}\) are integers with \(0 \leq \text{Min} \leq \text{Max} \leq \text{length}(L)\), and \(L\) is a list of DLAB atoms.

\textbf{Definition 14 (DLAB term)} \(D\textsc{term}\) is a DLAB term if and only if

- \(D\textsc{term}\) is a variable symbol, or
- \(D\textsc{term} = F(t_1, \ldots, t_n)\), where \(F\) is a function symbol and \(t_1, \ldots, t_n\) \((0 \leq n)\) is a sequence of \(n\) DLAB terms, or
- \(D\textsc{term} = \text{Min} \cdots \text{Max} : L\), where \(\text{Min}\) and \(\text{Max}\) are integers with \(0 \leq \text{Min} \leq \text{Max} \leq \text{length}(L)\), and \(L\) is a list of DLAB terms.

The function convert\textsubscript{\textcopyright} to\textsubscript{\textcopyright} dlab\_minus\textsubscript{\textcopyright}(DGRAM) required for converting grammars from DLAB to the DLAB\textsuperscript{\textcopyright} 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$\oplus$2
  inputs : (DTEMS, DVARS): DLAB grammar
  outputs : equivalent DLAB grammar without subsets on termlevel

DTEMS' := DTEMS
CONTINUE := true
while CONTINUE do
  find $P(t_1,\ldots,t_i,\text{Min} \cdot \text{Max} : [L_1,\ldots,L_n],t_{i+2},\ldots,t_m)$ in DTEMS',
  with for all $t_j \in \{t_1,\ldots,t_i\}, t_j \neq A \cdot B : C$
  if found
    then replace $P(t_1,\ldots,t_i,\text{Min} \cdot \text{Max} : [L_1,\ldots,L_n],t_{i+2},\ldots,t_m)$ in DTEMS' with $\text{Min} \cdot \text{Max} : [P(t_1,\ldots,t_i,L_1,t_{i+2},\ldots,t_m),\ldots,P(t_1,\ldots,t_i,L_n,t_{i+2},\ldots,t_m)]$
  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 ConvertToDLAB$\oplus$2(ConvertToDLAB$\oplus$1(DGRAM)) = (DGRAM$\oplus$, \emptyset), then DGRAM$\oplus$ is the DLAB$\oplus$ grammar equivalent to DGRAM.

6 DLAB$\oplus$ and DLAB at work in the FE mesh domain

We demonstrate the power of DLAB$\oplus$ and DLAB$^4$ 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$\oplus$ is a first DLAB$\oplus$ gram-

$^4$As a minor additional extension we will also allow DLAB atoms of the type $\text{Min} \cdot \text{len} : L$ or $\text{len} \cdot \text{len} : L$, where len is a constant symbol that abbreviates length($L$).
A next version extends the search space defined by $MeshDGRAM_1$ with rules that include characteristics of a second edge in the structure.
$\text{MeshDGRAM}_2 = (\text{MeshDTEMP}_2, \text{MeshDVARS}_2)$

\[
\text{MeshDTEMP}_2 = \\
\{ \text{mesh}(E, \text{resolution}) \\
\quad \leftarrow \\
\quad \begin{array}{l}
\quad 1 \cdots \text{len} : [\text{type}(E), \text{boundary}(E), \text{loading}(E), \\
\quad \quad \text{mesh}(E2, \text{resolution}), \text{topology}(E, E2), \\
\quad \quad \text{type}(E2), \text{boundary}(E2), \text{loading}(E2) \\
\quad ] \\
\quad \end{array} \\
\}
\]

$\text{MeshDVARS}_2 = \text{MeshDVARS}_1 \cup \{ \text{dlab\_variable(topology, 1 \cdots 1, [opposite, neighbour])} \}$

$\text{dlab\_size}(\text{MeshDGRAM}_2) = 4.91 \times 10^7$

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.

$\text{MeshDGRAM}_3 = (\text{MeshDTEMP}_3, \text{MeshDVARS}_2)$

\[
\text{MeshDTEMP}_3 = \\
\{ \text{mesh}(E, \text{resolution}) \\
\quad \leftarrow \\
\quad \begin{array}{l}
\quad \text{len} \cdots \text{len} : [1 \cdots \text{len} : [\text{type}(E), \text{boundary}(E), \text{loading}(E)], \\
\quad \quad 0 \cdots \text{len} : [\text{len} \cdots \text{len} : [\text{topology}(E, E2), \\
\quad \quad \quad 1 \cdots \text{len} : [\text{mesh}(E2, \text{resolution}), \\
\quad \quad \quad \quad \text{type}(E2), \text{boundary}(E2), \text{loading}(E2) \\
\quad \quad \quad ] \\
\quad \quad \quad ] \\
\quad \quad \quad ] \\
\quad \end{array} \\
\}
\]

$\text{dlab\_size}(\text{MeshDGRAM}_3) = 3.26 \times 10^7$
7 Relation of DLAB to other syntactic bias formalisms

We conclude with a brief situation of DLAB against some alternative declarative 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 DLAB are the clausemodels proposed in [1]. Clausemodels are expressions of the form Head ← Body, BodySet whose conversion to DLAB 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) ← P(Y), Q(X, Z), \{parent({X, Z}, Y)}\}

corresponds to DGRAM:

\[
DGRAM_5 = (DTEMPS_5, DVARS_5)
\]

\[
DTEMPS_5 =
\{grandfather(X, Y) ← \text{len} \cdot \text{len} : [P(Y), Q(X, Z),
0 \cdot \text{len} : \{\text{parent}(1 \cdot \text{len} : [X, Z], Y)]
\}
\]

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

Generally speaking, clausemodels are special cases of DLAB templates in which the choice of Min and Max is restricted. In fact, due to these constraints, none of the previous example DLAB 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 DLAB templates.

---

5 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 DLAB, in the sense that the same parameters trivially define (a series of) DLAB grammars.
Antecedent description grammars of Cohen [5]

An antecedent description grammar, as used in GRENDL, is in essence a definite clause grammar that generates the antecedents of clauses in $L$. 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.

\[
goal\_formula(\text{illegal}(A, B, C, D, E, F)).
\]

\[
body(\text{illegal}(A, B, C, D, E, F)) \rightarrow \text{rels}(A, B, C, D, E, F)
\]

\[
\text{rels}(A, B, C, D, E, F) \rightarrow [].
\]

\[
\text{rel}(A, B, C, D, E, F) \rightarrow \text{pred}(X, Y)
\]

where \( \text{member}(X, [A, B, C, D, E, F]), \text{member}(Y, [A, B, C, D, E, F]) \).

\[
\text{pred}(X, Y) \rightarrow [X = Y].
\]

\[
\text{pred}(X, Y) \rightarrow [\neg X = Y].
\]

\[
\text{pred}(X, Y) \rightarrow [\text{adj}(X, Y)].
\]

\[
\text{pred}(X, Y) \rightarrow [\neg \text{adj}(X, Y)].
\]

\[
\text{pred}(X, Y) \rightarrow [\text{less}\_\text{than}(X, Y)].
\]

\[
\text{pred}(X, Y) \rightarrow [\neg \text{less}\_\text{than}(X, Y)].
\]

For this example, we can write down an equivalent antecedent description grammar, using $\text{dlabl}$ (see Definition 4)

\[
goal\_formula(\text{illegal}(A, B, C, D, E, F)).
\]

\[
body(\text{illegal}(A, B, C, D, E, F)) \rightarrow \text{dlabl}(BA_6).
\]

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

\[
DGRAM_6 = (DTEMP_6, DVAR_6)
\]

\[
DTEMP_6 = \{ \text{illegal}(A, B, C, D, E, F) \leftarrow \text{pred}(1 \cdot \text{len} : [A, B, C, D, E, F],
1 \cdot \text{len} : [A, B, C, D, E, F]) \}
\]

\[
DVAR_6 = \text{dlab\_variable}(\text{pred}, 0 \cdot \text{len}, [=, \neq, \text{adj}, \neg \text{adj}, \text{less}\_\text{than}, \neg \text{less}\_\text{than}])
\]

In general however a conversion of antecedent description grammars to $\text{DLAB}$ is not always possible\(^6\). As suggested by the example, $\text{DLAB}$ is a special case of antecedent description grammars.

\(^6\)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 \textit{dlab1}. In other words, using \textsc{DLAB}, we work with a fixed and hidden definite clause grammar \textit{dlab1} that takes our \textsc{DLAB} grammar as its single argument. In that sense \textsc{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 \textit{dlab\_size(DGRAM)} combinatorics problem.

**References**


A The DLAB library

In this section we introduce a distributed Prolog\(^7\) library\(^8\) with predicates for manipulating DLAB data structures. In this section we introduce the relevant predicates in this library.

The main data structure manipulated by DLAB predicates is the uncompressed node (UNode), which roughly corresponds to the DLAB\(^6\) 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 DLAB predicates further described below.

\texttt{dlab_initialization/1}

\texttt{dlab\_initialization\( (_\text{Optimality}) \)}

\texttt{arg1: ground: atom}

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

\texttt{dlab\_root\_cnode/1}

\texttt{dlab\_root\_cnode\( (_\text{CNode}) \)}

\texttt{arg1: free: atom}

\(^7\)Currently ProLog by BIM 4.0.5 and SICStus Prolog 2.1 are supported.

\(^8\)This library is available at \texttt{ftp://ftp.cs.kuleuven.ac.be/pub/logic - prgm/ilp/dlab}.
Returns in \(\text{arg}1\) the compressed node corresponding to the empty clause. Uncompress this node and refine both body and head to obtain the top UNode in the hypothesis space.

\[\text{dlab\_uncompress\_node}/2\]

\[\text{dlab\_uncompress\_node}(_\text{CNode}, \text{unode}(_\text{Nr}, _\text{HeadUNode}, _\text{BodyUNode}))\]

arg1: ground: atom
arg2: free: term

Returns in UNode \(\text{arg}2\) the uncompressed equivalent to CNode \(\text{arg}1\). The UNode is a term \text{unode}/3. \(\text{Nr}\) is the sequence number of a template in the DLAB grammar. \(\text{HeadUNode}\) and \(\text{BodyUNode}\) are DLAB paths w.r.t. to this template.

\[\text{dlab\_compress\_node}/2\]

\[\text{dlab\_compress\_node}(\text{unode}(_\text{Nr}, _\text{HeadUNode}, _\text{BodyUNode}), _\text{CNode})\]

arg1: ground: term
arg2: free: atom

Returns in CNode \(\text{arg}2\) the compressed equivalent to UNode \(\text{arg}1\).

\[\text{dlab\_decode\_node}/3\]

\[\text{dlab\_decode\_node}(_\text{UNode}, _\text{Decoded}, _\text{GroundDecoded})\]

arg1: ground: term
arg2: free: term
arg3: free: term

Decodes the UNode in \(\text{arg}1\) to \(\text{arg}2\) and \(\text{arg}3\). \(\text{Decoded}\) is of the form \(\text{HeadList} : \text{BodyList}\), where \(\text{HeadList}\) (\(\text{BodyList}\)) is a list of head (body) literals of the clause identified by \(\text{arg}1\). \(\text{GroundDecoded}\) is of the same form, but here variables of \(\text{Decoded}\) are instantiated with the names they have in the original \text{dlab\_template}. \(\text{GroundDecoded}\) is useful for output of rules in a more readable format.

\[\text{dlab\_refine\_node}/3\]

\[\text{dlab\_refine\_node}(_\text{UNode}, _\text{Place}, _\text{RefinedUNode})\]

arg1: ground: term
arg2: any: atom
arg3: free: term

Refines the UNode in \(\text{arg}1\) at place \(\text{arg}2\). The result of refinement in returned in UNode \(\text{arg}3\). \(\text{Place}\) is either body or head.

\[\text{dlab\_refinable}/2\]

\[\text{dlab\_refinable}(_\text{Place}, _\text{UNode})\]

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 $\text{DLAB}$ library

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

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

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

Each $\text{DLAB}$ variable is represented as a fact $\text{dlab\_variable}(_P, _M i n - _M a x, _{\text{List\_Names}})$. An important constraint here is that $_P$ and the members of list $_{\text{List\_Names}}$ should be atoms. Moreover, $_P$ should be an atom that need not be quoted.

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

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

\begin{verbatim}
dlab_template('mesh(E,resolution)
  \leftarrow
  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,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]).
\end{verbatim}