

CLAUDIEN  
**The Clausal Discovery Engine**  
**User's Guide 3.0**

*Luc Dehaspe, Wim Van Laer, Luc De Raedt*

*Report CW 239, September 3, 1996*

Department of Computing Science, K.U.Leuven

**Abstract**

This is the users's guide to CLAUDIEN, a tool for discovering regularities in relational databases. We present some elementary background, a full sample session and a detailed description of the components of CLAUDIEN. The stand-alone version of CLAUDIEN that corresponds to this guide is available at URL:

*[ftp : //ftp.cs.kuleuven.ac.be/pub/logic – prgm/ilp/clauidien/clauidien3.0](ftp://ftp.cs.kuleuven.ac.be/pub/logic-prgm/ilp/clauidien/clauidien3.0)*

The WWW home page of CLAUDIEN can be found at URL:

*[http : //www.cs.kuleuven.ac.be/cwis/research/ai/Research/clauidien–E.shtml](http://www.cs.kuleuven.ac.be/cwis/research/ai/Research/clauidien-E.shtml)*

**Keywords** : machine learning, knowledge discovery, relational databases

# **CLAUDIEN**

## The CLAUsal DIsccovery ENgine User's Guide

Version 3.0  $\beta$   
September 3, 1996

Luc Dehaspe

Wim Van Laer

Luc De Raedt

Department of Computer Science  
Katholieke Universiteit Leuven  
Celestijnenlaan 200A  
B-3001 Heverlee  
Belgium

{Luc.Dehaspe,Wim.VanLaer,Luc.DeRaedt}@cs.kuleuven.ac.be

©1995, Luc De Raedt, Luc Dehaspe, and Wim Van Laer

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>How to run CLAUDIEN</b>	<b>2</b>
2.1	System prerequisites and installation . . . . .	2
2.1.1	User environment . . . . .	2
2.1.2	Prolog by BIM . . . . .	2
2.2	Basic components . . . . .	3
2.3	Optional components . . . . .	3
2.4	How to stop CLAUDIEN . . . . .	4
<b>3</b>	<b>Getting started: a short tutorial</b>	<b>4</b>
<b>4</b>	<b>Settings</b>	<b>21</b>
4.1	Knowledge . . . . .	21
4.1.1	partial_models . . . . .	21
4.1.2	leave_out . . . . .	22
4.2	Language . . . . .	22
4.2.1	bias . . . . .	22
4.2.2	max_head . . . . .	22
4.2.3	max_body . . . . .	22
4.2.4	max_complexity . . . . .	22
4.2.5	range_restricted . . . . .	23
4.2.6	types . . . . .	23
4.2.7	modes . . . . .	23
4.2.8	call_handling . . . . .	23
4.2.9	test_language . . . . .	23
4.3	Quantifying validity . . . . .	24
4.3.1	scope . . . . .	24
4.3.2	local_transform . . . . .	24
4.3.3	non_trivial . . . . .	24
4.3.4	min_accuracy . . . . .	25
4.3.5	min_lower_accuracy . . . . .	25
4.3.6	min_coverage . . . . .	25
4.3.7	heuristic . . . . .	25
4.4	Semantic pruning . . . . .	26
4.4.1	min_refine_coverage . . . . .	26
4.4.2	fair . . . . .	26
4.5	Search . . . . .	26
4.5.1	search . . . . .	26
4.5.2	beam_size . . . . .	27
4.5.3	parallel . . . . .	27
4.6	Miscellaneous . . . . .	27
4.6.1	max_realtime . . . . .	27

4.6.2	talking . . . . .	27
4.7	Theorem proving . . . . .	28
4.7.1	non_redundancy . . . . .	28
4.7.2	compactness . . . . .	28
<b>5</b>	<b>Knowledge</b>	<b>28</b>
5.1	Background knowledge . . . . .	28
5.2	Initial theory . . . . .	28
5.3	Observations . . . . .	28
5.4	Initial theory . . . . .	29
<b>6</b>	<b>Language</b>	<b>29</b>
6.1	<i>dlab_template/1</i> . . . . .	29
6.2	<i>dlab_variable/3</i> . . . . .	29
6.3	<i>dlab_macro/2</i> . . . . .	30
6.4	<i>dlab_type/1</i> . . . . .	30
6.5	<i>dlab_mode/2</i> . . . . .	30
6.6	<i>dlab_call/3</i> . . . . .	30

# 1 Introduction

CLAUDIEN discovers regularities in data. Roughly speaking, in relational database terminology, CLAUDIEN searches a space of YES/NO SQL queries for queries that return YES when submitted to the relational database. More formally, in terms of first-order logic, we can formulate the task of CLAUDIEN as follows.

Given

- a set of observations  $O$ , such that  $o_i \in O$  is a conjunction of first order Horn clauses
- a background knowledge base  $B$ , with  $B$  a conjunction of first order Horn clauses
- a language  $L$ , with  $L$  a set of clauses

CLAUDIEN produces a maximally general conjunction of clauses  $H$ , such that

- $H$  contains all the clauses of language  $L$  that are true in the least Herbrand model of  $B \cup o_i$ , for all  $o_i \in O$ .
- $H$  does not contain logically redundant clauses, i.e. there is no  $c \in H$  such that  $H - c \models H$
- for all  $o_i \in O$ ,  $H$  is true in the minimal Herbrand model of  $B \cup o_i$ .

Testing whether a clause  $c$  is true in the least Herbrand model of  $B \cup o_i$  roughly corresponds to running the Prolog query  $? - head(c), not(body(c))$  with  $B$  and  $o_i$  loaded. If this query finitely fails, clause  $c$  is a valid solution. Often additional restrictions are imposed on clauses in  $H$  (cf. settings of CLAUDIEN).

A key observation underlying CLAUDIEN is that clauses  $c$  false in the least Herbrand model of the knowledge base  $B$  are overly general, i.e. that there exist substitutions  $\theta$  for which  $body(c)\theta$  is true and  $head(c)\theta$  is false in the model. As they are overly general they should be specialized. Applying standard Inductive Logic Programming (ILP) principles, we can use a refinement operator  $\rho$  (under  $\theta$ -subsumption) for this (see [Muggleton and De Raedt, 1994; Shapiro, 1983]). Combining this with artificial intelligence search techniques results in the following basic algorithm:

```
 $Q := \{false\}; H := \emptyset;$ 
while  $Q \neq \emptyset$  do
  delete  $c$  from  $Q$ 
  if  $c$  is true in all minimal models of  $KB$ 
  then add  $c$  to  $H$ 
  else add all refinements  $\rho(c)$  of  $c$  to  $Q$ 
  endif
endwhile
```

For more details on CLAUDIEN we refer to the following publications.

- a full overview: [De Raedt and Dehaspe, 1995]

- declarative bias formalism DLAB: [Dehaspe and De Raedt, 1995a; Dehaspe and De Raedt, 1996]
- problem setting: [De Raedt and Bruynooghe, 1993; De Raedt and Lavrač, 1993; Muggleton and De Raedt, 1994]
- PAC-learning results: [De Raedt and Džeroski, 1994]
- applications: [Dehaspe *et al.*, 1994]
- parallel version: [Dehaspe and De Raedt, 1995b]

The rest of this user's guide is organized as follows. In Section 2 the inputs and outputs of CLAUDIEN are briefly introduced as well as basic instructions for installing, starting and stopping the system. Section 3 then contains a detailed description of a simple example session. The last three manual-like sections cover CLAUDIEN's settings (Section 4), knowledge (Section 5), and language (Section 6) components.

## 2 How to run CLAUDIEN

### 2.1 System prerequisites and installation

To use CLAUDIEN you need a Sun system running SunOS 4.1.3 or Solaris 2 and about 3Mb of disk space. Installation is then straightforward: create a home directory for CLAUDIEN and copy the distributed files in this directory (see also the README file included in the distribution).

#### 2.1.1 User environment

Each user has to set the value of the environment variable *CLAUDIEN\_DIR* to the CLAUDIEN home directory, and *CLAUDIEN\_BIM\_PROLOG\_COMPILER* to *YES* or *NO* (see Section 2.1.2). The user should also extend the path with *CLAUDIEN\_DIR/bin/bin*.

#### 2.1.2 Prolog by BIM

CLAUDIEN is implemented in Prolog by BIM, release 4.0.5. The CLAUDIEN system made available to the general public is a stand-alone version, hence it runs without the prolog by BIM compiler. However, before loading the user's application files, CLAUDIEN checks the environment variable *CLAUDIEN\_BIM\_PROLOG\_COMPILER*. If this variable has the value *YES*, CLAUDIEN compiles the application-specific files, otherwise, if the value is *NO*, the application files are interpreted.

Users of CLAUDIEN should be well aware that, especially with large applications, system performance might be seriously affected if application files are interpreted rather than compiled. Those interested in purchasing the Prolog by BIM compiler (separately) can contact BIM via Email at *prolog@bim.be*.

The table below should give some indication of how the compiler influences performance. The table contains the time (in cpu seconds) it took CLAUDIEN to complete the finite-element

mesh design application as it is included in the distribution of CLAUDIEN. We have tested on two machines a SPARCstation 2 and 20, once with, once without the prolog by BIM compiler.

	SPARCstation 2	SPARCstation 20
without Prolog by BIM compiler	105887 cpu sec	18299 cpu sec.
with Prolog by BIM compiler	18490 cpu sec	4864 cpu sec.

Notice CLAUDIEN runs about 5 times faster with the Prolog by BIM compiler. The same speedup can be obtained by moving from a SPARCstation 2 to a SPARCstation 20.

The problem that performance of CLAUDIEN depends on the presence of a Prolog by BIM compiler might be partly solved in the near future, when we make available to the public our CLAUDIEN version coupled to *Oracle7<sup>TM</sup>*.

## 2.2 Basic components

To run CLAUDIEN you have to create two files

`<applic>.kb` the input knowledge base containing background knowledge and observations

`<applic>.l` the language bias

where `<applic>` is the name of your *application*. From a UNIX shell, CLAUDIEN is then started by performing:

```
% cl <applic>
```

The solution hypothesis produced by CLAUDIEN is automatically written to an output file `<applic>.out`.

## 2.3 Optional components

You can also choose to separate the background knowledge and observations. You should then store the observations in `<applic>.kb` as before, and create an extra input file

```
<applic>.bg
```

where you put background knowledge.

Another optional input file is

```
<applic>.s
```

where you can specify non-default settings. If no `<applic>.s` is present, all settings get default values.

In a third extra input file

```
<applic>.t
```

you can write a theory that will be added to the theorem prover during initialisation of each run (illustrated below).

A final extension to the basic setup is added to support multiple experiments on the same dataset `<applic>.kb`. You can associate with each experiment a configuration name `<config>` and use this name to prefix the files that together determine the experimental setup: `<config><applic>.l`, `<config><applic>.s`, `<config><applic>.bg`, and `<config><applic>.t`. This feature can for instance be used to store inputs and outputs of different experiments in separate directories. In that case you simply have to choose a directory name (ending on `/`) for your configuration.

If you have a configuration `<config>`, you can start CLAUDIEN with the appropriate input files by typing:

```
% cl <applic> [<config>]
```

Notice `<config>` is optional, if it is not specified, it gets by default the empty string value.

## 2.4 How to stop CLAUDIEN

You can stop CLAUDIEN any time with ctrl-C. The interrupt signal is caught by CLAUDIEN and the system terminates gracefully.

# 3 Getting started: a short tutorial

In this section we guide you through a toy CLAUDIEN application `deptcw`. The idea is to give you a basic feeling of how you can make CLAUDIEN do what.

First, open a file `deptcw.kb` in which you will write some background knowledge and observations concerning four users (wimv, ldh, wiske, suske) of the computing facilities at your department.

```
begin(background).
  ml_system(claudien).
  ml_system(icl).
end(background).
```

```
begin(model(wimv)).
  male.
  assistant.
  implements(icl).
end(model(wimv)).
```

```
begin(model(ldh)).
  male.
  assistant.
  implements(claudien).
end(model(ldh)).
```

```

begin(model(wiske)).
    female.
    student.
    implements(chess).
end(model(wiske)).

begin(model(suske)).
    male.
    student.
    implements(minesweeper).
end(model(suske)).

```

The facts `begin(background)` and `end(background)` mark out the background knowledge. Likewise, the beginning and ending of each observation is indicated with the facts `begin(model(Id))` and `end(model(Id))`, where *Id* uniquely identifies the observation.

Important general remark:

*If you do not have a Prolog by BIM compiler, your input files will be interpreted on a line-by-line basis. Make sure each fact or rule in the <applic>.kb file starts on a new line.*

Save the file `deptcw.kb` and open a file `deptcw.l`. In this file you will define the hypothesis space to be searched by CLAUDIEN. For this you have to use the DLAB format (see [Dehaspe and De Raedt, 1995a; Dehaspe and De Raedt, 1996]).

```

dlab_template('0-len:[male,female,student,assistant]
<--
0-len:[male,female,student,assistant]')

```

The bodies and heads of the clauses in the language defined above contain any subset of the predicates `{male, female, student, assistant}`.

For this first run we will use only default settings and no separate file for background knowledge. So for the time being you do not have to create the files `deptcw.s` and `deptcw.bg`.

You do have created the `deptcw.kb` and `deptcw.l` files for your application `deptcw` and are ready to run CLAUDIEN. Issue the command

```
% cl deptcw
```

You should get approximately the following information on your screen.

```

*****
***** C L A U D I E N *****
*****
*
* Runtime version : 3.0
* Created on : Tue Apr 30 10:57:28 1996
*****

```

Prolog by BIM compiler available, user files will be compiled

Please give the name of the configuration for deptcw  
?

You have no configuration for *deptcw*, so you can simply press <ENTER> here.

Now loading files for application deptcw with configuration

```
**Initialising background knowledge**
  there is no backgroundfile deptcw.bg
**Initialising settings**
  set settings to their default value ... done
  loading deptcw.s ...
**Initialising language**
  loading deptcw.l ...
Initializing DLAB ...
dlab_template 1 successfully parsed by DLAB
Size language : 256
**Initialising knowledge base**
  loading deptcw.kb
    o there is no optimized file deptcw.kb.0.wic
    o using non-optimized deptcw.kb ...
    o multiple models found in deptcw.kb
    -> 4 models have been loaded
  preparing knowledge ... ml_compiling -c+ .deptcw.kb
done
```

Starting interactive session

\*\*\*\*\*

For list of commands : h/0 or help/0

claudien-deptcw>

The last line is the prompt *claudien-<applic>* that indicates CLAUDIEN is ready to accept your command. Try to call up the list of commands, some help for *help* and *claudien*, and display information with *show\_info/0*.

claudien-deptcw> help

The following commands are available:

help/0	help/1	quit/0
prompt/0	claudien/0	show_settings/0
show_settings/1	show_info/0	show_language/0
load_settings/0	set_default_settings/0	set/2
optimise_kb_file/1	optimise_kb_file/0	save_theory/0
new_config/0	new_config/1	load_language/0

```
claudien-deptcw> help(help/1)
Synopsis : help(Command) -- h(Command)
--> print specific help for Command
```

```
claudien-deptcw> h(claudien)
Synopsis : claudien -- c
--> start the discovery process
```

```
claudien-deptcw> show_info
*****
***** C L A U D I E N *****
*****
* Version : 3.0
* Runtime : yes
* Runtime created on : Tue Apr 30 10:57:28 1996
* Prolog by BIM compiler available : YES
*
* Date :Tue Apr 30 11:10:06 1996
* User : ldh
* Dir : /home/ml/claudien/CV-3/EX/DEPTCW/
*
* Application : deptcw
* Configuration :
* Knowledgebase file : deptcw.kb -- loaded
* Background file : deptcw.bg -- not found
* Settings file : deptcw.s -- not found
* Language file : deptcw.l -- loaded
* Theory file : deptcw.t -- not found
*
* total number of models : 4
*
*** SETTINGS ***
*** knowledge ***
* partial_models : off (default)
* leave_out : false (default)
*** language ***
* bias : dlab (default)
* max_head : 10 (default)
* max_body : 10 (default)
* max_complexity : 100 (default)
* range_restricted : on (default)
* types : off (default)
* modes : off (default)
* call_handling : off (default)
* test_language : off (default)
*** quantifying validity ***
* scope : global (default)
* local_transform : off (default)
```

```

* non_trivial : on (default)
* min_accuracy : 1 (default)
* min_lower_accuracy : off (default)
* min_coverage : 1 (default)
* heuristic : mdl (default)
*** semantic pruning ***
* min_refine_coverage : 1 (default)
* fair : on (default)
*** search ***
* search : breadth (default)
* beam_size : 5 (default)
* parallel : 1 (default)
*** misc ***
* max_real_time : off (default)
* talking : 3 (default)
*** theorem proving ***
* non_redundancy : on (default)
* compactness : on (default)
*
*** DLAB GRAMMAR ***
dlab_template(' 0-len:[male,female,student,assistant]
              <--
              0-len:[male,female,student,assistant] ').
*
* Size language : 256
*****

```

```
claudien-deptcw>
```

With `show_info`, you get an overview of the current setup : the version of CLAUDIEN, the input files, the settings, the DLAB grammar, and the total size of the hypothesis space CLAUDIEN will search.

So now you know how to start the discovery process: type either *claudien* or the short command *c*. The following should appear on your screen (we interrupt the output with comments in *italic* where appropriate).

```

claudien-deptcw> c
**Initialisation**
  initialising theoremprover
  -> no theory file loaded, deptcw.t not found ... done

      false if true           [a(0),t(4),p(0),n(4),c(2),val(add to queue)]
New nodes: 1           Remaining nodes : 1
Number of solutions : 0           Consumed cpu : 0.05

```

CLAUDIEN *tells you it has inspected the most general clause false if true in your hypothesis space, and found that is was not a solution, but a candidate for further development. In a*

heuristic search, this clause would get a value. Since this is a breadth first search (cf. setting search), the value is simply 'add to queue'. The rest of the information in the list that follows 'false if true' denotes:

- a*(Acc) The accuracy (= Pos/Total).
- t*(Total) The number of observations (= Pos + Neg).
- p*(Pos) The number of observations where body implies head.
- n*(Neg) The number of observations where body is true and head is false.
- c*(Comp) The complexity, counted as the total number of variables, constants, functors, and predicate names.

So 1 new node has been added to your previously empty queue of candidates, 0 solutions have been found, and 0.05 cpu seconds have elapsed.

```

Clause being refined : false if true
  male if true           [a(0.75),t(4),p(3),n(1),c(2),val(add to queue)]
  female if true        [a(0.25),t(4),p(1),n(3),c(2),val(add to queue)]
  student if true       [a(0.5),t(4),p(2),n(2),c(2),val(add to queue)]
  assistant if true     pruned : not further refinable
  false if male         [a(0.25),t(4),p(1),n(3),c(2),val(add to queue)]
  false if female       [a(0.75),t(4),p(3),n(1),c(2),val(add to queue)]
  false if student      [a(0.5),t(4),p(2),n(2),c(2),val(add to queue)]
  false if assistant    [a(0.5),t(4),p(2),n(2),c(2),val(add to queue)]
New nodes: 7           Remaining nodes : 7
Number of solutions : 0           Consumed cpu : 0.15

```

The clause false if true is removed from the queue for further refinement. There are 8 refinements; none of them are solutions; 7 of them are added back to the queue; 1 is pruned. Clause assistant if true is pruned because DLAB considers it to be not further refinable. This has to do with the fact that with CLAUDIEN DLAB works in optimal mode. See [Dehaspe and De Raedt, 1995a] for more details.

```

Clause being refined : male if true
  male;female if true    +**** S O L U T I O N +****
  male;student if true   +**** S O L U T I O N +****
  male;assistant if true pruned : fairness violated
New nodes: 0           Remaining nodes : 6
Number of solutions : 2           Consumed cpu : 0.216667

```

The next clause selected for refinement is male if true. There are 3 refinements, 2 of which are solutions. Notice that solutions are not added to the queue. Neither is the third clause. The reason here is the violation of the fairness principle (cf. [De Raedt and Dehaspe, 1995]). If a refinement adds Added to the head of a clause  $Head \leftarrow Body$  then the fairness principle is violated if there is no observation  $O$  such that Body and Added are true w.r.t.  $O$ , and Head is false w.r.t.  $O$ . In other words, Added does not improve the validity of the clause. In the current case there is no observation in which the added assistant is true and male false.

Clause being refined : female if true  
 female;student if true [a(0.5),t(4),p(2),n(2),c(3),val(add to queue)]  
 female;assistant if true pruned : not further refinable  
 New nodes: 1 Remaining nodes : 6  
 Number of solutions : 2 Consumed cpu : 0.266667

Clause being refined : student if true  
 student;assistant if true \*\*\*\*\* S O L U T I O N \*\*\*\*\*  
 New nodes: 0 Remaining nodes : 5  
 Number of solutions : 3 Consumed cpu : 0.316667

Clause being refined : false if male  
 male if male pruned : tautology  
 female if male pruned : fairness violated  
 student if male [a(0.333333),t(3),p(1),n(2),c(2),val(add to queue)]  
 assistant if male pruned : not further refinable  
 false if male,female \*\*\*\*\* S O L U T I O N \*\*\*\*\*  
 false if male,student [a(0.75),t(4),p(3),n(1),c(3),val(add to queue)]  
 false if male,assistant [a(0.5),t(4),p(2),n(2),c(3),val(add to queue)]  
 New nodes: 3 Remaining nodes : 7  
 Number of solutions : 4 Consumed cpu : 0.416667

*male if male is pruned because it is a tautology: the same literal occurs at both sides of the implication sign.*

Clause being refined : false if female  
 male if female pruned : fairness violated  
 female if female pruned : tautology  
 student if female pruned : non-redundancy violated  
 assistant if female pruned : fairness violated  
 false if female,student pruned : fairness violated  
 false if female,assistant \*\*\*\*\* S O L U T I O N \*\*\*\*\*  
 New nodes: 0 Remaining nodes : 6  
 Number of solutions : 5 Consumed cpu : 0.45

*student if female is pruned because it is implied by the conjunction of the previously discovered solutions male;student if true, and false if male,female (it suffices to resolve on male), see also non-redundancy in [De Raedt and Dehaspe, 1995].*

*false if female,student is an illustration of a second way to violate the fairness principle. If a refinement adds Added to a the body of a clause Head ← Body then the fairness principle is violated if Body implies Added. This means Added does not bring you closer to a solution. Here female implies student.*

Clause being refined : false if student  
 male if student [a(0.5),t(2),p(1),n(1),c(2),val(add to queue)]  
 female if student [a(0.5),t(2),p(1),n(1),c(2),val(add to queue)]  
 student if student pruned : tautology  
 assistant if student pruned : fairness violated

```

false if student,assistant      +**** S O L U T I O N +****
New nodes: 2      Remaining nodes : 7
Number of solutions : 6      Consumed cpu : 0.516667

Clause being refined : false if assistant
male if assistant      pruned : non-redundancy violated
female if assistant      pruned : fairness violated
student if assistant      pruned : fairness violated
assistant if assistant      pruned : tautology
New nodes: 0      Remaining nodes : 6
Number of solutions : 6      Consumed cpu : 0.55

Clause being refined : female;student if true
female;student;assistant if true      pruned : non-redundancy violated
New nodes: 0      Remaining nodes : 5
Number of solutions : 6      Consumed cpu : 0.55

Clause being refined : student if male
student;assistant if male      pruned : non-redundancy violated
New nodes: 0      Remaining nodes : 4
Number of solutions : 6      Consumed cpu : 0.583333

Clause being refined : false if male,student
male if male,student      pruned : tautology
female if male,student      pruned : fairness violated
student if male,student      pruned : tautology
assistant if male,student      pruned : fairness violated
false if male,student,assistant      pruned : non-redundancy violated
New nodes: 0      Remaining nodes : 3
Number of solutions : 6      Consumed cpu : 0.6

Clause being refined : false if male,assistant
male if male,assistant      pruned : tautology
female if male,assistant      pruned : fairness violated
student if male,assistant      pruned : fairness violated
assistant if male,assistant      pruned : tautology
New nodes: 0      Remaining nodes : 2
Number of solutions : 6      Consumed cpu : 0.65

Clause being refined : male if student
male;female if student      pruned : non-redundancy violated
male;student if student      pruned : tautology
male;assistant if student      pruned : fairness violated
New nodes: 0      Remaining nodes : 1
Number of solutions : 6      Consumed cpu : 0.7

Clause being refined : female if student
female;student if student      pruned : tautology

```

```
female;assistant if student                pruned : fairness violated
New nodes: 0          Remaining nodes : 0
Number of solutions : 6          Consumed cpu : 0.716667
```

*The number of nodes remaining in the queue has been reduced to 0, so the discovery process stops.*

\*\*\* COMPACTING THEORY \*\*\*

```
retracted((false :- female , assistant)).
```

*As the compact setting is on (cf. [De Raedt and Dehaspe, 1995]), CLAUDIEN will now try to reduce the discovered theory by looking for additional implications between solutions in reversed order. Here CLAUDIEN found the clause false if female, assistant is implied by the combination of the later generated solution false if student,assistant, with the previously generated male;student if true and false if male,female.*

\*\*\*\*\* DISCOVERED CLAUSES \*\*\*\*\*

```
false :- male , female.
```

```
false :- student , assistant.
```

```
male ; female :- true.
```

```
male ; student :- true.
```

```
student ; assistant :- true.
```

*You get an overview of the discovered clauses.*

\*\*\*\*\* STATISTICS \*\*\*\*\*

```
* Stop date : Thu Mar 14 09:45:29 1996
```

```
* Type of ending : regular
```

```
* Consumed cpu : 0.8
```

```
*
```

```
* Nodes pruned for syntactic reasons : 13
```

```
*   + not range restricted : 0
```

```
*   + tautology : 10
```

```
*   + type declarations violated : 0
```

```
*   + mode declarations violated : 0
```

```
*   + maxhead exceeded : 0
```

```
*   + maxbody exceeded : 0
```

```
*   + clause too complex : 0
```

```
*   + not further refinable : 3
```

```
*
```

```
* Nodes pruned for semantic reasons : 26
```

```
*   + solution : 6
```

```
*   + non-redundancy violated : 6
```

```

*       + fairness violated :                14
*       + p+n under minimal solution coverage : 0
*       + p+n under minimal refinement coverage : 0
*       + p under minimal solution coverage : 0
*
* Nodes added to queue : 14
*       + no solution added to queue : 14
*       + solution added to queue : 0
*
* Size of search space : 256
* Total number of nodes visited : 53
* Percentage of search space visited : 20.7031 %
*
*****

```

*Finally, some statistics about the run are printed. You mainly get an overview of how many nodes are pruned for each of the 14 possible reasons and of how many nodes of the queue are added to the queue.*

Now have a look at the outputfile *deptcw.out* created by CLAUDIEN (if possible in another window, without closing this session).

```

/*
*****
***** C L A U D I E N *****
*****
* Version : 3.0
* Runtime : yes
*
* Date :Tue Apr 30 11:10:06 1996
* User : ldh
* Dir : /home/ml/clauidien/CV-3/EX/DEPTCW/
*
* Application : deptcw
* Configuration :
* Knowledgebase file : deptcw.kb -- loaded
* Background file : deptcw.bg -- not found
* Settings file : deptcw.s -- not found
* Language file : deptcw.l -- loaded
*
* total number of models : 4
*
*** SETTINGS ***
*** knowledge ***
* partial_models : off (default)
* leave_out : false (default)
*** language ***
* bias : dlab (default)
* max_head : 10 (default)

```

```

* max_body : 10 (default)
* max_complexity : 100 (default)
* range_restricted : on (default)
* types : off (default)
* modes : off (default)
* call_handling : off (default)
* test_language : off (default)
*** quantifying validity ***
* scope : global (default)
* local_transform : off (default)
* non_trivial : on (default)
* min_accuracy : 1 (default)
* min_lower_accuracy : off (default)
* min_coverage : 1 (default)
* heuristic : mdl (default)
*** semantic pruning ***
* min_refine_coverage : 1 (default)
* fair : on (default)
*** search ***
* search : breadth (default)
* beam_size : 5 (default)
* parallel : 1 (default)
*** misc ***
* max_real_time : off (default)
* talking : 3 (default)
*** theorem proving ***
* non_redundancy : on (default)
* compactness : on (default)
*
*** DLAB GRAMMAR ***
dlab_template(' 0-len:[male,female,student,assistant]
<--
0-len:[male,female,student,assistant] ').
*
* Size language : 256
*****

*/
rule(1,[accuracy(1),total(4),pos(4),neg(0),comp(3),cpu(0.166667),realtime(2)],
(male ; female :- true)).
rule(2,[accuracy(1),total(4),pos(4),neg(0),comp(3),cpu(0.2),realtime(2)],
(male ; student :- true)).
rule(3,[accuracy(1),total(4),pos(4),neg(0),comp(3),cpu(0.3),realtime(2)],
(student ; assistant :- true)).
rule(4,[accuracy(1),total(4),pos(4),neg(0),comp(3),cpu(0.366667),realtime(2)],
(false :- male , female)).
rule(5,[accuracy(1),total(4),pos(4),neg(0),comp(3),cpu(0.45),realtime(2)],
(false :- female , assistant)).

```

```

rule(6,[accuracy(1),total(4),pos(4),neg(0),comp(3),cpu(0.516667),realtime(3)],
      (false :- student , assistant)).
/** COMPACTING THEORY **/

retracted((false :- female , assistant)).

/***** STATISTICS *****/
* Stop date : Thu Mar 14 09:45:29 1996
* Type of ending : regular
* Consumed cpu : 0.8
*
* Nodes pruned for syntactic reasons : 13
*   + not range restricted : 0
*   + tautology : 10
*   + type declarations violated : 0
*   + mode declarations violated : 0
*   + maxhead exceeded : 0
*   + maxbody exceeded : 0
*   + clause too complex : 0
*   + not further refinable : 3
*
* Nodes pruned for semantic reasons : 26
*   + solution : 6
*   + non-redundancy violated : 6
*   + fairness violated : 14
*   + p+n under minimal solution coverage : 0
*   + p+n under minimal refinement coverage : 0
*   + p under minimal solution coverage : 0
*
* Nodes added to queue : 14
*   + no solution added to queue : 14
*   + solution added to queue : 0
*
* Size of search space : 256
* Total number of nodes visited : 53
* Percentage of search space visited : 20.7031 %
*
*****/

```

This file is ready to be loaded in Prolog in case you want to use the output of CLAUDIEN for some further purpose. The initial block repeats the setup information (obtained with the *show\_info* command from the CLAUDIEN prompt). Then you get the rules discovered by CLAUDIEN. Each rule is represented as a fact *rule/3*:

- *Arg1* is the sequence number
- *Arg2* lists some information

**accuracy/1** The accuracy ( $= Pos/Total$ ).

**total/1** The number of observations ( $= Pos + Neg$ ).

**pos/1** The number of observations where body implies head.

**neg/1** The number of observations where body is true and head is false.

**comp/1** The complexity, counted as the total number of variables, constants, functors, and predicate names.

**cpu/1** The cpu time it took to find this rule (in seconds).

**realtime/1** The real time it took to find this rule (in seconds).

- *Arg3* is the discovered rule

If the *compact* setting was on, you also get a list of retracted rules. Notice you have to remove retracted rules (in this case rule 5) manually from the list of *rule/3* facts. Finally you get the statistics of the CLAUDIEN run.

From the CLAUDIEN prompt you can now conduct new experiments. Remember to take a backup copy of your *.out* file first if you do not want it to be overwritten.

After each discovery run you may decide to save the theory by typing **save\_theory**. If you do that the discovered theory is saved in `<config><applic>.t`. This file will be loaded automatically during initialisation of the next run, and the theory it contains will be added to the theoremprover.

```
claudien-deptcw> save_theory
writing theory to file deptcw.t ...
done
claudien-deptcw>
```

The file `deptcw.t` now contains the following:

```
male ; female :- true .
male ; student :- true .
student ; assistant :- true .
false :- male , female .
false :- student , assistant .
```

If you now reduce the talkativity of CLAUDIEN, and rerun, you should get...

```
claudien-deptcw> set(talking,0)
claudien-deptcw> c
**Initialisation**
initialising theoremprover
-> loading theory from file deptcw.t ... done

*** COMPACTING THEORY ***
```

\*\*\*\*\* DISCOVERED CLAUSES \*\*\*\*\*

\*\*\*\*\* STATISTICS \*\*\*\*\*

```
* Stop date : Thu Mar 14 17:10:29 1996
* Type of ending : regular
* Consumed cpu : 0.95
*
* Nodes pruned for syntactic reasons : 27
*   + not range restricted : 0
*   + tautology : 10
*   + type declarations violated : 0
*   + mode declarations violated : 0
*   + maxhead exceeded : 0
*   + maxbody exceeded : 0
*   + clause too complex : 0
*   + not further refinable : 3
*
* Nodes pruned for semantic reasons : 12
*   + solution : 0
*   + non-redundancy violated : 12
*   + fairness violated : 14
*   + p+n under minimal solution coverage : 0
*   + p+n under minimal refinement coverage : 0
*   + p under minimal solution coverage : 0
*
* Nodes added to queue : 14
*   + no solution added to queue : 14
*   + solution added to queue : 0
*
* Size of search space : 256
* Total number of nodes visited : 53
* Percentage of search space visited : 20.7031 %
*
*****
```

claudien-deptcw>

Notice no clauses are listed under **DISCOVERED CLAUSES**, but in the statistics you can see that the number of redundant nodes has increased. Accordingly, the **deptcw.out** file will contain no rules.

We will now illustrate how the configuration name can be used to organise your experiments. First create a file **deptcw2.kb** identical to **deptcw.kb** but without the *begin(background) ... end(background)* block. Then create two subdirectories **setup1/** and **setup2/**, for each of the two experiments we will conduct. Copy **deptcw.1** into **setup1/deptcw2.1**, and open a file **setup1/deptcw2.s** in which you write the following deviations from the default settings:

talking(0).

Start CLAUDIEN with application name *deptcw2* and the first configuration *setup1/*.

```
% c1 deptcw2 setup1/
*****
***** C L A U D I E N *****
*****
*
* Runtime version : 3.0
* Created on : Tue May 14 14:15:45 1996
*****

Now loading files for application deptcw2 with configuration setup1/
**Initialising background knowledge**
  there is no backgroundfile setup1/deptcw2.bg
**Initialising settings**
  set settings to their default value ... done
  loading setup1/deptcw2.s ...
**Initialising language**
  loading setup1/deptcw2.l ...
Initializing DLAB ...
dlab_template 1 successfully parsed by DLAB
Size language : 256
**Initialising knowledge base**
  loading deptcw2.kb
    o there is no optimized file deptcw2.kb.0.wic
    o using non-optimized deptcw2.kb ...
    o multiple models found in deptcw2.kb
    -> 4 models have been loaded
  preparing knowledge ... ml_compiling -c+ .deptcw2.kb
done

Starting interactive session
*****

For list of commands : h/0 or help/0

claudien-setup1/deptcw2>
```

Now run the discovery process (**claudien**) and save the theory (**save\_theory**) as before. To prepare the second experiment, you should:

- copy the file *setup1/deptcw2.t* to *setup2/deptcw2.t*;
- copy the file *setup1/deptcw2.s* to *setup2/deptcw2.s*;
- open a file *setup2/deptcw2.l* in which you write the following language:

```

dlab_template('0-len: [male,female,student,assistant,
                implements(Program),ml_system(Program)]
            <--
            0-len: [male,female,student,assistant,
                implements(Program),ml_system(Program)]').

```

- open a file *setup2/deptcw2.bg* in which you write:

```

ml_system(claudien).
ml_system(icl).

```

Now start and run CLAUDIEN again.

```

% cl deptcw2 setup2/
*****
***** C L A U D I E N *****
*****
*
* Runtime version : 3.0
* Created on : Tue May 14 14:15:45 1996
*****

Now loading files for application deptcw2 with configuration setup2/
**Initialising background knowledge**
loading setup2/deptcw2.bg ...
ml_compiling -c+ setup2/deptcw2.bg
**Initialising settings**
set settings to their default value ... done
loading setup2/deptcw2.s ...
**Initialising language**
loading setup2/deptcw2.l ...
ml_compiling -c+ setup2/deptcw2.l
Initializing DLAB ...
dlab_template 1 successfully parsed by DLAB
Size language : 4096
**Initialising knowledge base**
loading deptcw2.kb
o there is no optimized file deptcw2.kb.0.wic
o using non-optimized deptcw2.kb ...
o multiple models found in deptcw2.kb
-> 4 models have been loaded
preparing knowledge ... ml_compiling -c+ .deptcw2.kb
done

Starting interactive session
*****

For list of commands : h/0 or help/0

```

```

claudien-setup2/deptcw2> claudien
**Initialisation**
  initialising theoremprover
  -> loading theory from file setup2/deptcw2.t ... done

*** COMPACTING THEORY ***

retracted((false :- female , implements(B) , ml_system(B))).

***** DISCOVERED CLAUSES *****

false :- student , implements(A) , ml_system(A).

ml_system(A) :- assistant , implements(A).

***** STATISTICS *****
* Stop date : Tue May 14 14:36:17 1996
* Type of ending : regular
* Consumed cpu : 1.05
*
* Nodes pruned for syntactic reasons : 114
*   + not range restricted : 34
*   + tautology : 77
*   + type declarations violated : 0
*   + mode declarations violated : 0
*   + maxhead exceeded : 0
*   + maxbody exceeded : 0
*   + clause too complex : 0
*   + not further refinable : 3
*
* Nodes pruned for semantic reasons : 103
*   + solution : 3
*   + non-redundancy violated : 36
*   + fairness violated : 64
*   + p+n under minimal solution coverage : 0
*   + p+n under minimal refinement coverage : 0
*   + p under minimal solution coverage : 0
*
* Nodes added to queue : 61
*   + no solution added to queue : 61
*   + solution added to queue : 0
*
* Size of search space : 4096
* Total number of nodes visited : 278
* Percentage of search space visited : 6.78711 %
*

```

```
*****
```

```
claudien-setup2/deptcw2>
```

The file *setup2/deptcw2.out* should now contain the rules

```
rule(1,[accuracy(1),total(4),pos(4),neg(0),comp(6),cpu(0.48),realtime(1)],
      (false :- female , implements(Program) , ml_system(Program))).
rule(2,[accuracy(1),total(4),pos(4),neg(0),comp(6),cpu(0.54),realtime(1)],
      (false :- student , implements(Program) , ml_system(Program))).
rule(3,[accuracy(1),total(2),pos(2),neg(0),comp(5),cpu(0.58),realtime(1)],
      (ml_system(Program) :- assistant , implements(Program))).
```

```
/** COMPACTING THEORY */
```

```
retracted((false :- female , implements(A) , ml_system(A))).
```

At this point you should take some time to play around with the language, the knowledge base (e.g. make it less sexist) and the settings (e.g. turn off *non\_redundancy*). Each time rerun CLAUDIEN and see what happens...

Have fun!

## 4 Settings

With the command `show_settings` you get an overview of all settings used by CLAUDIEN. To change the default value of a setting `<name>` to `new_value` you can either type the command `set(<name>,<new_value>)` (e.g. `set(talking,2)`), or you can create a file `<config><applic>.s` where you write the Prolog fact `<name>(<new_value>)`. (e.g. `talking(2)`). To get a list of possible values for a setting, simply try a nonsensical one:

```
claudien-setup2/deptcw2> set(search,nonsense)
>>> ERROR user: search: nonsense must be either best, breadth, depth, or beam
```

We now describe each setting in more detail.

### 4.1 Knowledge

#### 4.1.1 partial\_models

use: *partial\_models(V)*

constraint: *V* must be either on or off

default: *V = off*

description: If on, CLAUDIEN handles partial models. In that case full clausal logic can be used to describe observations. (This feature is currently not publicly available.)

### 4.1.2 `leave_out`

use: `leave_out(Test)`

constraint: *Test* should be the body of a Prolog clause

default: `Test = false`

description: CLAUDIEN will skip all observations in which *Test* succeeds. This feature can be useful for conducting leave-N-out experiments.

## 4.2 Language

### 4.2.1 `bias`

use: `bias(B)`

constraint: `atom(B)`

default: `B = dlab`

description: *B* is the declarative language bias formalism used (currently only DLAB).

### 4.2.2 `max_head`

use: `max_head(N)`

constraint: *N* must be a positive number

default: `N = 10`

description: the maximal number of literals allowed in the head of a solution hypothesis

### 4.2.3 `max_body`

use: `max_body(N)`

constraint: *N* must be a positive number

default: `N = 10`

description: the maximal number of literals allowed in the body of a solution hypothesis

### 4.2.4 `max_complexity`

use: `max_complexity(N)`

constraint: *N* must be a positive number

default: `N = 100`

description: the maximal complexity of solution hypothesis defined as the total number of symbols (variables, constants, predicate names, functor names)

#### 4.2.5 range\_restricted

use: *range\_restricted(V)*

constraint:  $V$  must be either on or off

default:  $V = on$

description: If on, only range restricted clauses are considered. A clause  $c$  is range restricted if and only if  $vars(head(c)) \subset vars(body(c))$ .

#### 4.2.6 types

use: *types(V)*

constraint:  $V$  must be either on or off

default:  $V = off$

description: If on, only type conform clauses are considered (see Section 6.4). The use of this feature is strongly discouraged. Whenever possible type restrictions should be imposed via DLAB templates.

#### 4.2.7 modes

use: *modes(V)*

constraint:  $V$  must be either on or off

default:  $V = off$

description: If on, only mode conform clauses are considered (see Section 6.5). The use of this feature is strongly discouraged. Whenever possible mode restrictions should be imposed via DLAB templates.

#### 4.2.8 call\_handling

use: *call\_handling(V)*

constraint:  $V$  must be either on or off

default:  $V = off$

description: If on, literals  $'\#'/1$  occurring in clauses generated by DLAB are transformed as described in Section 6.6 before they are evaluated.

#### 4.2.9 test\_language

use: *test\_language(V)*

constraint:  $V$  must be either on or off

default:  $V = off$

description: If on, clauses are not evaluated (nor pruned) but simply shown. This feature can be useful to test the DLAB specifications.

### 4.3 Quantifying validity

To describe the settings for quantifying validity we here use the terminology introduced in [De Raedt and Dehaspe, 1995], Sections 4.5 and 4.6.

#### 4.3.1 scope

use:  $scope(V)$

constraint:  $V$  must be either global or local

default:  $V = global$

description: If the scope is set to global, CLAUDIEN will count the number of observations in which a clause is true, in order to determine its validity.

In case there is only one observation, it makes more sense to set scope to local, such that CLAUDIEN will count positive and negative substitutions.

#### 4.3.2 local\_transform

use:  $local\_transform(I)$

constraint:  $I$  must be either off or a strictly positive number; in the latter case scope should be set to local (see above).

default:  $I = off$

description: If scope is set to local, it is often convenient to transform a clause

$$p_1, \dots, p_m \leftarrow q_1, \dots, q_n$$

into the following logically equivalent form

$$p_1, \dots, p_m, \neg q_{I+1}, \dots, \neg q_n \leftarrow q_1, \dots, q_I$$

before constructing positive and negative substitutions. Notice the first  $I$  literals are kept in the body of the clause, the others are moved to the head. By appropriately choosing  $I$  it is possible that meaningful entities are counted.

#### 4.3.3 non\_trivial

use:  $non\_trivial(V)$

constraint:  $V$  must be either on or off

default:  $V = off$

description: Let  $c$  be  $p_1, \dots, p_m \leftarrow q_1, \dots, q_n$ . The clause  $c$  is non-trivial with regard to a background theory  $B$  and a set of observations  $O$  if and only if either  $m > 0$  and there exists an observation  $o \in O$  and a substitution  $\theta$  such that  $(q_1 \wedge \dots \wedge q_n)\theta$  is true in  $M(B \cup o)$ , or,  $m = 0$  and for all  $k$  there exists a

substitution  $\theta$  and an observation  $o$  such that  $(q_1 \wedge \dots \wedge q_{k-1} \wedge q_{k+1} \wedge q_n)\theta$  is true in  $M(B \cup o)$ .

If *non\_trivial* is on, clauses are excluded that trivially hold from the hypotheses. Without non-triviality, one can always postulate implications, provided that the condition part never holds.

#### 4.3.4 min\_accuracy

use: *min\_accuracy*( $R$ )

constraint:  $R$  must be a real number between 0 and 1

default: 1

description: If set to 1, CLAUDIEN will only accept solutions that are valid on all data. By lowering the accuracy (often called ‘confidence’) threshold, this strong requirement can be relaxed. In general all rules are accepted as solutions for which the ratio *valid\_data/all\_data*  $\geq R$ . A specific formula depends on the scope and *non\_trivial* settings.

#### 4.3.5 min\_lower\_accuracy

use: *min\_lower\_accuracy*( $R$ )

constraint:  $R$  must be either off or a real number between 0 and 1; in the latter case,  $R$  should be less than the *min\_accuracy* setting (see above).

default:  $R = \text{off}$

description: If not off, CLAUDIEN will consider clauses with validity above  $R$  as solutions. Unless however their validity is also above the “hard” *min\_accuracy* threshold, these clauses will not be pruned away. Thus, more specific (and more accurate) descendants may be discovered later on during the search.

#### 4.3.6 min\_coverage

use: *min\_coverage*( $N$ )

constraint:  $N$  must be a strictly positive number

default:  $N = 1$

description: The coverage (often called ‘support’) of a rule refers to the number of cases in which it non-trivially applies, e.g. the number of observations for which the body of the clause succeeds. CLAUDIEN will only accept solutions that cover at least  $N$  cases.

#### 4.3.7 heuristic

use: *heuristic*( $V$ )

constraint:  $V$  must be either mdl or laplace

default:  $V = mdl$

description: If a heuristic search method is chosen (best or beam), a certain formula is used to rank candidates for further refinements (candidates with lower scores are evaluated earlier). With heuristic mdl (based on the minimal description length principle) this formula is  $p/(c+n)$ , where  $p$  accounts for the positive substitutions or interpretations,  $n$  for the negative ones, and  $c$  for the complexity of the clause defined as the total number of symbols (variables, constants, predicate names, functor names). With heuristic laplace:  $(p+1)/(p+n+2)$ .

## 4.4 Semantic pruning

### 4.4.1 min\_refine\_coverage

use:  $min\_refine\_coverage(N)$

constraint:  $N$  must be a strictly positive number

default:  $N = 1$

description: The coverage (often called 'support') of a rule refers to the number of cases in which it non-trivially applies, e.g. the number of observations for which the body of the clause succeeds. CLAUDIEN will prune candidates that cover less than  $N$  cases.

### 4.4.2 fair

use:  $fair(V)$

constraint:  $V$  must be either on or off

default:  $V = on$

description: If on, CLAUDIEN will prune clauses that violate the fairness principle.

If a refinement adds *Added* to the head of a clause  $Head \leftarrow Body$  then the fairness principle is violated if there is no observation  $O$  such that *Body* and *Added* are true w.r.t.  $O$ , and *Head* is false w.r.t.  $O$ . In other words, *Added* does not improve the validity of the clause.

If a refinement adds *Added* to the **body** of a clause  $Head \leftarrow Body$  then the fairness principle is violated if *Body* implies *Added*. This means *Added* does not bring you closer to a solution.

Pruning in these two cases is 'safe' if the language is fair. A language  $\mathcal{L}$  is fair if and only if  $\forall c_1, c_2, c_3 \in \mathcal{L}$  and substitution  $\theta_2$ ,  $c_1 \subset c_2$  and  $c_2\theta_2 \subset c_3 \rightarrow c_1\theta_2 \cup (c_3 - c_2\theta_2) \in \mathcal{L}$  (see also [De Raedt and Dehaspe, 1995], Section 4.2.3).

## 4.5 Search

### 4.5.1 search

use: *search(V)*

constraint: *V* must be best, breadth, depth, or beam

default: *V = breadth*

description: Selects one of four possible strategies for searching the space of candidate solutions.

### 4.5.2 beam\_size

use: *beam\_size(N)*

constraint: *N* must be a strictly positive number

default: *N = 5*

description: If search is set to *beam*, *beam\_size* determines the size of the beam.

### 4.5.3 parallel

use: *parallel(N)*

constraint: *N* must be a strictly positive number

default: *N = 1*

description: Determines the degree of parallelism. (Currently, in the public version only degree one can be used).

## 4.6 Miscellaneous

### 4.6.1 max\_real\_time

use: *max\_real\_time(R)*

constraint: *R* must be either off or a positive real number

default: *R = off*

description: If not off, CLAUDIEN will be interrupted after *R* seconds (real time) have elapsed.

### 4.6.2 talking

use: *talking(N)*

constraint: *N* must be either 0,1, 2, or 3

default: *N = 3*

description: Influences the amount of information CLAUDIEN shows on the screen (0 is almost no information). This setting has no effect on the amount of information written to the `<config><applic>.h` output file.

## 4.7 Theorem proving

### 4.7.1 non\_redundancy

use: *non\_redundancy(V)*

constraint: *V* must be either on or off

default: *V = on*

description: If on, CLAUDIEN will prune clauses that are logically entailed by the background knowledge and previously discovered solutions.

### 4.7.2 compactness

use: *compactness(V)*

constraint: *V* must be either on or off

default: *V = on*

description: If on, CLAUDIEN will after the discovery process has terminated go through the solutions in reverse order and eliminate solutions that are logically entailed by the background knowledge and solutions discovered later on.

## 5 Knowledge

CLAUDIEN accepts three types of knowledge: background knowledge, observations (examples), and an initial theory to be added to the theorem prover. Only the observations are obligatory.

### 5.1 Background knowledge

Background knowledge should be put in the beginning of file `<applic>.kb` or in a separate file `<config><applic>.bg`. In the former case, if multiple observations are specified in `<applic>.kb` (see the next Section), the beginning and ending of background knowledge should be indicated with the facts `begin(background).` and `end(background)..`

Important general remark:

*If you do not have a Prolog by BIM compiler, your input files will be interpreted on a line-by-line basis. Make sure each fact or rule in the <applic>.bg file starts on a new line.*

### 5.2 Initial theory

### 5.3 Observations

In case there are multiple observations, the beginning and ending of each observation should be indicated with the facts `begin(model(Id)).` and `end(model(Id)).`, where *Id* uniquely identifies the observation.

If there is only one observation these indications can be omitted. In fact, if the first clause CLAUDIEN detects in the `<applic>.kb` file is not of type `begin(model(Id)).` or `begin(background).`, the system will assume only one observation is specified. In that case do not forget to set the *scope* setting to *local* (see Section 4.3.1).

Important general remark:

*If you do not have a Prolog by BIM compiler, your input files will be interpreted on a line-by-line basis. Make sure each fact or rule in the <applic>.kb file starts on a new line.*

## 5.4 Initial theory

In a third extra input file `<config><applic>.t` you can write a full clausal theory that will be added to the theorem prover during initialisation of each run.

A clause of the form  $A_1, \dots, A_m \leftarrow B_1, \dots, B_n$  should be written as

$(m > 0, n > 0)$   $A_1; \dots; A_m : -B_1, \dots, B_n.$

$(m = 0)$   $false : -B_1, \dots, B_n.$

$(n = 0)$   $A_1; \dots; A_m : -true.$

## 6 Language

The search bias is defined in the file `<config><applic>.l`, with the predicates *dlab\_template/1*, *dlab\_variable/3*, *dlab\_macro/2*, *dlab\_type/1*, and *dlab\_mode/1*, and *dlab\_call/3*.

The terminology concerning the declarative language bias formalism DLAB is explained in [Dehaspe and De Raedt, 1995a; Dehaspe and De Raedt, 1996; De Raedt and Dehaspe, 1995].

### 6.1 dlab\_template/1

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

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

### 6.2 dlab\_variable/3

Each DLAB 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.

### 6.3 *dlab\_macro/2*

A DLAB macro is represented as a fact or a rule *dlab\_macro(\_SubString, \_NewSubString)*. In the first stage of initialization DLAB will scan the DLAB templates for substrings *\_SubString* and replace these with *\_NewSubString*.

### 6.4 *dlab\_type/1*

To specify types, add for all literals *L* that might occur in a solution, a ground fact *dlab\_type(L)* to the language file (e.g. *type(works\_at(employee, company))*). A clause *c* is then type conform if and only if a grounding substitution  $\theta$  can be found for *c* by unifying each literal *L* with the single argument of a *type/1* fact.

The use of this feature is strongly discouraged. Whenever possible type restrictions should be imposed via DLAB templates. If type restrictions are defined with *dlab\_type/1*, clauses will be first generated (by DLAB) and then tested (and possibly rejected). With DLAB templates on the other hand you can prevent unwanted clauses from being generated at all, which will obviously speed up the discovery process.

### 6.5 *dlab\_mode/2*

To specify modes for a literal *L* that might occur in a solution, add a ground fact *dlab\_mode(L)* to the language file. Thereby set each argument to *i* (for input) if you want CLAUDIEN to verify whether this argument is instantiated at the time of calling, or set it to *o* (for output) if you do not care whether it is instantiated.

The use of this feature is strongly discouraged. Whenever possible mode restrictions should be imposed via DLAB templates. As with *dlab\_type/1* (see Section 6.4) it is more efficient to prevent mode violating clauses from being generated.

### 6.6 *dlab\_call/3*

In the default case, clauses generated by DLAB are immediately evaluated. For instance, `dlab_template('ok <-- 0-len:[len-len:[q(X), f(X,Y)]]')`.

will generate at some point the clause

```
ok if q(X),f(X,Y)
```

and this clause will be tested.

However, CLAUDIEN contains a rather low-level, Progol [Muggleton, 1995]-inspired feature that can be used to instantiate some variables before the clause is tested. The predicate in which these variables occur should then be specified as the first argument of a predicate *'#/1* in the DLAB template. Suppose in the example above the *Y* variable should be instantiated, then the DLAB template should be modified as follows:

```
dlab_template('ok <-- 0-len:[len-len:[q(X), #(f(X,Y)]]')
```

The predicate *dlab\_call/3* should then be used to specify how the variables will be instantiated. Currently the user has to write all the necessary Prolog code. Arg1 of *dlab\_call/3* corresponds to the argument of *#/1* in the DLAB template, Arg2 and Arg3 will be instantiated to respectively the head and body of the clause at the time of calling.

Returning to our example, suppose we would like to instantiate *Y* to all values for which the body succeeds in at least one observation. We accomplish this with:

```
dlab_call(f(_X, _Y), _Head, _Body):-
    Crossings = [_Y],
        % _Y is the variable we want to instantiate
    varlist(f(_Head, _Body), VarList),
        % VarList is the list of all variables in the clause
    eqsubtract$utilities(VarList, Crossings, Foo),
        % Foo is the list of all variables but _Y
    setof(Crossings, Foo^(select_model$utilities,_Body), Instantiations),
        % all Instantiations of _Y for which _Body succeeds are gathered
    member$utilities(Crossings, Instantiations).
        %Y is unified with one the Instantiations
```

In the example above module qualification *\$utilities* is used where claudien code is reused. You can write your own definitions in the language file if necessary. You do have to use *select\_model\$utilities* before accessing information in your *.kb* file.

If the setting *call\_handling* is on, CLAUDIEN will not evaluate

```
ok if q(X),#(f(X,Y))
```

but all clauses that can be found on backtracking by instantiating *Y*. In the *dlab\_call* above, *member* is the predicate that will be backtracked upon.

For instance with the following data in a *.kb* file

```
f(2,14).
f(4,18).
f(1,9).
```

```
q(1).
q(2).
```

you will get the following output with settings *test\_language(on), talking(3), scope(local)*

```
...
    ok if true                [val(test language)]
New nodes:  1      Remaining nodes :  1
Number of solutions :  0      Consumed cpu :  0.63

Clause being refined : ok if true
    ok if q(X),f(X,9)        [val(test language)]
    ok if q(X),f(X,14)       [val(test language)]
New nodes:  2      Remaining nodes :  2
Number of solutions :  0      Consumed cpu :  0.67
```

Notice you here get all  $Y$  values for  $f(X, Y)$ , for which the body succeeds. Through modification of *dlab\_call/3*, you can get one  $Y$  value, the average of all  $Y$  values, the highest, lowest, etc.

We are working on this number handling feature to make it more efficient and user-friendly. In the mean time, be aware of the following constraints:

- when CLAUDIEN encounters and expands a  $\#/1$  predicate in the DLAB template all refinements of this clause are pruned; in other words, use this feature only *at the end* of clauses;
- the DLAB size value does not take into account the *call* expansions, as these are generated dynamically;
- complex calculations (e.g. regression) might seriously slow down CLAUDIEN; take into account these calculations have to be repeated for every clauses tested by CLAUDIEN.

## Acknowledgements

Since Luc De Raedt first conceived and implemented a prototype for CLAUDIEN, a lot of people have been involved in its further development. Discussions with Maurice Bruynooghe, Hilde Adé, Gunther Sablon, Hendrik Blockeel, Sašo Džeroski, Nada Lavrač, Stephen Mugleton and Peter Flach proved to be very fruitful. Bojan Dolsak, Sašo Džeroski, Ashwin Srinivasan, Rüdiger Wirth, Peter Brockhausen, Walter Daelemans, Arno Knobbe, and Pieter Adriaans generously provided data used in experiments. Patrick Weemeeuw provided advice on the parallel implementation of CLAUDIEN. Finally, we received a lot of feedback from the early users of CLAUDIEN: Hendrik Blockeel, Sašo Džeroski, Bjarte Østvold, Chris Borgelt, as well as a (large) number of master's students.

This work is part of the ESPRIT projects no. 6020 and 20237 on Inductive Logic Programming and Inductive Logic Programming II. Wim Van Laer and Luc De Raedt are supported by the Belgian National Fund for Scientific Research.

## References

- [De Raedt and Bruynooghe, 1993] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann, 1993.
- [De Raedt and Dehaspe, 1995] L. De Raedt and L. Dehaspe. Clausal discovery. Forthcoming, 1995.
- [De Raedt and Džeroski, 1994] L. De Raedt and S. Džeroski. First order  $jk$ -clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.
- [De Raedt and Lavrač, 1993] L. De Raedt and N. Lavrač. The many faces of inductive logic programming. In J. Komorowski, editor, *Proceedings of the 7th International Symposium*

*on Methodologies for Intelligent Systems*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1993. invited paper.

- [Dehaspe and De Raedt, 1995a] L. Dehaspe and L. De Raedt. DLAB: a declarative language bias for concept learning and knowledge discovery engines. Technical Report CW-214, Department of Computer Science, Katholieke Universiteit Leuven, October 1995.
- [Dehaspe and De Raedt, 1995b] L. Dehaspe and L. De Raedt. Parallel inductive logic programming. In Y. Kodratoff, G. Nakhaeizadeh, and G. Taylor, editors, *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, pages 112–117, Heraklion, Crete, Greece, 1995.
- [Dehaspe and De Raedt, 1996] L. Dehaspe and L. De Raedt. DLAB: A declarative language bias formalism. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems (ISMIS96)*. Springer-Verlag, 1996. to appear.
- [Dehaspe *et al.*, 1994] L. Dehaspe, W. Van Laer, and L. De Raedt. Applications of a logical discovery engine. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 291–304, Sankt Augustin, Germany, 1994. Gesellschaft für Mathematik und Datenverarbeitung MBH.
- [Muggleton and De Raedt, 1994] S. Muggleton and L. De Raedt. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
- [Muggleton, 1995] S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13, 1995.
- [Shapiro, 1983] E.Y. Shapiro. *Algorithmic Program Debugging*. The MIT Press, 1983.