Appendix A Declarative Bias

The use of *declarative bias*, which allows users to declaratively specify the search space of possible clauses to be explored while learning, is common in ILP systems such as PROGOL (Muggleton 1995), TILDE (Blockeel and De Raedt 1998), CLAUDIEN (De Raedt and Dehaspe 1997), ALEPH (Srinivasan 2001), etc. When the space is potentially huge, it plays an important role in restricting the search to finite and meaningful clauses. For our purposes, we adapt the bias declarations from (De Raedt 2008). In *DiceML*, the bias consists of four types of declarations, i.e., type, mode, rand, and rank declarations. We describe them in turn with examples:

Types: All functors are accompanied by *type declarations* of the form $type(func(t_1, \cdots, t_n))$, where t_i denotes the type of the i-th argument, i.e., the domain of the variable. For instance, consider the type declarations in Figure A 1. Since the first argument of `hasAcc/2` should be different type than the argument of `freq/1`, the clause

```
age(C)∼ gaussian(30, 2.1) ← mod(X,(hasAcc(C,A),freq(C)≅X),low).
```

is not type conform, but the following clause is:

```
age(C)∼ gaussian(30, 2.1) ← mod(X,(hasAcc(C,A),freq(A)≅X),low).
```

Modes: We also employ modes, which is standard in ILP, for each attribute. Modes specify the form of literal b_i in the body of the clause $h \sim \mathcal{D}_\phi \leftarrow b_1, \ldots, b_n, \mathcal{M}_\psi$. A *mode declaration* is an expression of the form $mode(a_1, aggr, (r(m_1, \ldots, m_j), a_2(m_k)))$, where m_i are different modes associated with variables of functors, $aggr$ is the name of aggregation function, r is the link relation, and a_i are attributes. The expression specifies the candidate aggregation functions considered while learning clauses for the attribute a_1. If the link relation is absent, then the aggregation function is not needed, so the mode declaration reduces to the form $mode(a_1, none, a_2(m_k)))$. The modes m_i can be either *input* (denoted by "+") or *output* (denoted by "−"). The input mode specifies that at the time of calling the functor the corresponding argument must be instantiated, the output mode specifies that the argument will be instantiated after a successful call to the functor. Consider the mode declarations in Figure A 1. The clause

```
age(C)∼ gaussian(30, 2.1) ← mod(X,(cliLoan(C,L1),status(L2)≅X),appr).
```

is not mode conform since the first argument of `cliLoan/2`, i.e., the variable `C` does not satisfy the output mode and the variable `L2` does not satisfy the input mode. The following clause, however, satisfies the mode:

```
age(C)∼ gaussian(30, 2.1)← mod(X,(cliLoan(C1,L1),status(L1)≅X),appr).
```

Rand Declarations: They are used to define the type of random variables (i.e., discrete or continuous) and to specify the domain of discrete random variables.

Rank Declarations: As we have already seen in Section 3.2, the second validity condition of the DC program requires the existence of a rank assignment $\prec$ over predicates of the program. Hence, we introduce these declarations, to specify the rank assignment

% Type declarations
```
type(client(c)).
type(loan(l)).
type(account(a)).
type(hasAcc(c,a)).
type(hasLoan(c,l)).
type(age(c)).
type(creditScore(c)).
type(loanAmt(l)).
type(status(l)).
type(savings(a)).
type(freq(a)).
```

% Mode declaration
```
mode(age,none,creditScore(+)).
mode(age,sum,(hasAcc(+,-),savings(+))).
mode(age,avg,(hasAcc(+,-),savings(+))).
mode(age,mod,(hasAcc(+,-),freq(+))).
mode(age,max,(cliLoan(+,-),loanAmt(+))).
mode(age,mod,(cliLoan(-,-),status(+))).
mode(status,none,loanAmt(+)).
mode(status,mod,(hasLoan(-,+),age(+))).
⋮
```

% Rank declaration
```
rank([age,creditScore,loanAmt,
status,savings,freq]).
```

% Random variable declaration
```
rand(age,continuous,[]).
rand(creditScore,continuous,[]).
rand(loanAmt,continuous,[]).
rand(status,discrete,[appr,pend,decl]).
rand(savings,continuous,[]).
rand(freq,discrete,[low,high]).
```

% Transformed tables
```
client(ann).
loan(l_20).
account(a_10).
age(ann) ~ val(33).
creditScore(john) ~ val(700).
savings(a_10) ~ val(3050).
freq(a_10) ~ val(high).
loanAmt(l_20) ~ val(20050).
hasAcc(ann,a_11).
hasLoan(a_11,l_20).
⋮
```

% Background knowledge
```
age(carl) ~ gaussian(40,5.1).
cliLoan(C,L)←hasAcc(C,A),hasLoan(A,L).
```

Figure A 1. An example of input to *DiceML*, which consists of a transformation of the spreadsheet in Table 1, along with background knowledge and declarative bias.

over attributes. While learning distributional clauses for a single attribute, the rank declaration is not used, it is crucial while learning DC programs.

Example Appendix A.1. An example of the input to *DiceML* is shown in Figure A 1, where Table 1 is converted into facts, and background knowledge is expressed using distributional clauses. The first clause in the background knowledge shown in the bottom-right of the figure states that the age of `carl` follows a Gaussian distribution, and the second clause states that if a client has an account in the bank and the account is linked to a loan account, then the client also has a loan.