APPENDIX

A SESSION WITH INDEX

INDEX is a prototype system for inductive data engineering, implemented in Quintus Prolog. Starting from a given extensional database relation it suggests attribute dependencies that could result in a meaningful decomposition. After the user has chosen a dependency the decomposition is carried out, also with some help from the user. The main algorithms have been described in chapter 8; below follows an annotated session, corresponding to the example in §33. The INDEX prompt is '?'; user input is in **bold**.

```
itksu3> index
? [train3].
 [consulting /a/kubsu2/usr5/flach/prolog/Deps/train3...]
 [train3 consulted 0.100 sec 1,608 bytes]
? get pos train.
? init ics train.
   1. hour
   2. mins
   3. direction
   4. type
   5. platform
Which dependencies? all.
? show all.
positive tuples:
     train(8,7,utrecht,intercity,5)
     train(8,9,tilburg,intercity,6)
     train(8,12,utrecht,stopping_train,4)
     train(8,16,tilburg,stopping_train,4)
     train(8,37,utrecht,intercity,5)
     train(8,39,tilburg,intercity,6)
     train(8,42,utrecht,stopping_train,4)
     train(8,46,tilburg,stopping_train,4)
     train(8,57,utrecht,intercity,6)
     train(9,7,utrecht,intercity,5)
     train(9,9,tilburg,intercity,6)
     train(9,37,utrecht,intercity,5)
     train(9,39,tilburg,intercity,6)
     train(9,42,utrecht,stopping_train,4)
     train(9,46,tilburg,stopping_train,4)
```

```
There are no negative tuples.
integrity constraints:
     train:[]->->[hour]
     train:[]->->[mins]
     train:[]->->[direction]
     train:[]->->[type]
     train:[]->->[platform]
     train:[]-->[hour]
     train:[]-->[mins]
     train:[]-->[direction]
     train:[]-->[type]
     train:[]-->[platform]
```

After starting INDEX the user reads a file consisting a relation scheme and a number of positive tuples into Prolog. The positive tuples are then read into the INDEX database, and the user tells INDEX to construct the strongest attribute dependencies (here generically called integrity constraints) for all attributes in the relation scheme.

```
? switch all.
switches ON:
     cwa
switches OFF:
     debug
     eval
     horn
? find ics.
-train(9,12,utrecht,stopping_train,4)
-train(8,9,utrecht,intercity,6)
-train(8,12,utrecht,intercity,4)
-train(8,9,tilburg,intercity,5)
-train(8,12,utrecht,stopping_train,5)
-train(8,7,utrecht,stopping_train,4)
-train(8,57,utrecht,intercity,5)
? show ics all.
integrity constraints:
     train:[mins]-->[direction,platform,type]
     train:[platform]-->[type]
     train:[mins]->->[direction]
     train:[mins]->->[hour]
     train:[mins]->->[platform]
     train:[mins]->->[type]
     train:[platform]->->[type]
```

The operation of INDEX can be controlled by setting a number of *switches*. The `cwa` switch tells INDEX to use the Closed World Assumption rather than querying the user about the indicated tuples (i.e. to perform non-incremental rather than incremental induction). The `debug` switch can be used to make INDEX more talkative. The `eval` switch tells INDEX to use heuristics to find dependencies that are almost satisfied, rather than to find the set of strongest dependencies that are satisfied; this switch is now off to illustrate the latter process, and will be switched on for the rest of the session. Finally, the `horn` switch can be used to display integrity constraints in Horn form, i.e. as definite clauses.

The above list of integrity constraints found by INDEX consists of all non-trivial attribute dependencies that hold for the given relation; apart from the fd *Platform*$\rightarrow$*Type* (and consequently the mvd *Platform*$\twoheadrightarrow$*Type*) these dependencies are contingent, caused by the circumstance that no two trains happen to leave at the same moment.

```
? switch eval.
eval is now on.
? del ics all.
? init ics train.
   1. hour
   2. mins
   3. direction
   4. type
   5. platform
Which dependencies? all.
? find ics.
? show ics all.
integrity constraints:
     train:[mins]-->[direction,platform,type]
     train:[platform]-->[type]
     train:[type,direction]-->[platform]
     train:[]->->[hour]
     train:[mins]->->[direction]
     train:[mins]->->[platform]
     train:[mins]->->[type]
     train:[platform]->->[type]
     train:[direction,type]->->[mins]
     train:[direction,platform]->->[mins]
     train:[type,direction]->->[platform]
? check ics train:[]->->[hour].
train:[]->->[hour] looks promising: sat 0.8
```

The search for dependencies is performed again, this time with the `eval` switch for heuristic evaluation turned on. The reader can verify for herself that the dependencies that weren’t found previously, i.e. the ones that are almost but not completely satisfied, are the third fd *Type*,*Direction*$\rightarrow$*Platform*, the first mvd $\emptyset\twoheadrightarrow$*Hour*, and the last three mvds. The user chooses to take a closer look at the mvd $\emptyset\twoheadrightarrow$*Hour* and asks for its satisfaction degree, which is 0.8 (i.e. 3 exceptions out of 15 tuples; see below).

```
? decomp  train:[]->->[hour].
part1 --- :
     train(8,7,utrecht,intercity,5)
     train(8,9,tilburg,intercity,6)
     train(8,37,utrecht,intercity,5)
     train(8,39,tilburg,intercity,6)
     train(8,42,utrecht,stopping_train,4)
     train(8,46,tilburg,stopping_train,4)
     train(9,7,utrecht,intercity,5)
     train(9,9,tilburg,intercity,6)
     train(9,37,utrecht,intercity,5)
     train(9,39,tilburg,intercity,6)
     train(9,42,utrecht,stopping_train,4)
     train(9,46,tilburg,stopping_train,4)
part2 --- :
     train(8,12,utrecht,stopping_train,4)
     train(8,16,tilburg,stopping_train,4)
     train(8,57,utrecht,intercity,6)
Proceed? yes.
```

The user is satisfied with this satisfaction degree, and asks INDEX to construct a decomposition (note that INDEX finds out that the decomposition should be horizontal, since the dependency is violated). INDEX now constructs a horizontal decomposition and asks the user whether she wants to proceed. The decomposition is rather trivial in this example: since there are no antecedent attributes the antecedent partition trivially consists of the whole relation, and the second step of the algorithm separates the relation into two non-contradicting blocks, for which the reader only has to supply names. I will illustrate a more elaborate horizontal decomposition after this session has been completed.

```
tuples:
     train(8,7,utrecht,intercity,5)
     train(8,9,tilburg,intercity,6)
     train(8,37,utrecht,intercity,5)
     train(8,39,tilburg,intercity,6)
     train(8,42,utrecht,stopping_train,4)
     train(8,46,tilburg,stopping_train,4)
     train(9,7,utrecht,intercity,5)
     train(9,9,tilburg,intercity,6)
     train(9,37,utrecht,intercity,5)
     train(9,39,tilburg,intercity,6)
     train(9,42,utrecht,stopping_train,4)
     train(9,46,tilburg,stopping_train,4)
relation name? hourlytrain.
```

```
tuples:
     train(8,12,utrecht,stopping_train,4)
     train(8,16,tilburg,stopping_train,4)
     train(8,57,utrecht,intercity,6)
relation name? irregtrain.
Decompose irregtrain? no.
Decompose hourlytrain? yes.
attributes:
     hour
relation name? hour.
attributes:
     mins
     direction
     type
     platform
relation name? hourlytrain1.
```

The user recognises the first set of tuples as the hourly trains, and the remaining three as irregular trains. Since both of these new relations now satisfy the mvd $\emptyset \rightarrow\!\!\rightarrow \mathit{Hour}$, the user is asked whether she wants to construct the corresponding vertical decomposition for them. The user decides to do this only for the hourly trains, resulting in two new relations, one listing the possible hours, and one listing the values of the remaining attributes for each hourly train, as shown below.

```
? show all.
positive tuples:
     hour(8)
     hour(9)
     hourlytrain1(7,utrecht,intercity,5)
     hourlytrain1(9,tilburg,intercity,6)
     hourlytrain1(37,utrecht,intercity,5)
     hourlytrain1(39,tilburg,intercity,6)
     hourlytrain1(42,utrecht,stopping_train,4)
     hourlytrain1(46,tilburg,stopping_train,4)
     irregtrain(8,12,utrecht,stopping_train,4)
     irregtrain(8,16,tilburg,stopping_train,4)
     irregtrain(8,57,utrecht,intercity,6)
negative tuples:
     […see above…]
integrity constraints:
     train=[irregtrain,hourlytrain]
     hourlytrain=hour><hourlytrain1
```

```
? switch  horn.
horn is now on.
? show  ics  all.
integrity constraints:
train(A,B,C,D,E) :-
        (   irregtrain(A,B,C,D,E)
        ;   hourlytrain(A,B,C,D,E)
        ).
hourlytrain(A,B,C,D,E) :-
        hour(A),
        hourlytrain1(B,C,D,E).
? halt.
itksu3>
```

The composition rules are included as integrity constraints, and are displayed above in relational form and as clauses.

I will now illustrate a more elaborate horizontal decomposition of the same relation, imposed by the fd *Type,Direction→Platform*. This fd has only one exception, as indicated by its satisfaction degree.

```
? check  ics  train:[type,direction]-->[platform].
train:[type,direction]-->[platform] looks promising:
     sat 0.933333
? decomp  train:[type,direction]-->[platform].
segment1 --- :
     train(8,9,tilburg,intercity,6)
     train(8,12,utrecht,stopping_train,4)
     train(8,16,tilburg,stopping_train,4)
     train(8,39,tilburg,intercity,6)
     train(8,42,utrecht,stopping_train,4)
     train(8,46,tilburg,stopping_train,4)
     train(9,9,tilburg,intercity,6)
     train(9,39,tilburg,intercity,6)
     train(9,42,utrecht,stopping_train,4)
     train(9,46,tilburg,stopping_train,4)
segment2 --- part1 --- :
     train(8,7,utrecht,intercity,5)
     train(8,37,utrecht,intercity,5)
     train(9,7,utrecht,intercity,5)
     train(9,37,utrecht,intercity,5)
segment2 --- part2 --- :
     train(8,57,utrecht,intercity,6)
Proceed? yes.
```

In this non-minimal horizontal decomposition, segments roughly correspond to antecedent blocks, and parts correspond to non-contradicting blocks within antecedent blocks. The first segment actually consists of several antecedent blocks, which can be seen from the fact that there are several *Type*,*Direction*-values within this segment — it consists of all non-contradicting antecedent blocks: i.e. the Tilburg intercity trains, the Utrecht stopping trains, and the Tilburg stopping trains. Only the fourth antecedent block, consisting of the Utrecht intercity trains, contradicts the fd and is split into two non-contradicting blocks.

The user can now decide to keep all these blocks separate (this would be a non-minimal decomposition), or to combine the first segment with either the first or the second part of the second segment, by assigning a name to each of these blocks.

```
tuples:
     train(8,9,tilburg,intercity,6)
     train(8,12,utrecht,stopping_train,4)
     train(8,16,tilburg,stopping_train,4)
     train(8,39,tilburg,intercity,6)
     train(8,42,utrecht,stopping_train,4)
     train(8,46,tilburg,stopping_train,4)
     train(9,9,tilburg,intercity,6)
     train(9,39,tilburg,intercity,6)
     train(9,42,utrecht,stopping_train,4)
     train(9,46,tilburg,stopping_train,4)
relation name? regtrain.
tuples:
     train(8,7,utrecht,intercity,5)
     train(8,37,utrecht,intercity,5)
     train(9,7,utrecht,intercity,5)
     train(9,37,utrecht,intercity,5)
relation name? regtrain.
tuples:
     train(8,57,utrecht,intercity,6)
relation name? irregtrain.
```

The user chooses the minimal decomposition with the smallest number of exceptions. She proceeds by vertically decomposing the relation of regular trains.

```
Decompose irregtrain? no.
Decompose regtrain? yes.
attributes:
     type
     direction
     platform
relation name? platform.
```

```
attributes:
     hour
     mins
     direction
     type
relation name? **regtrain1.**
? **show all.**
positive tuples:
     platform(intercity,tilburg,6)
     platform(intercity,utrecht,5)
     platform(stopping_train,tilburg,4)
     platform(stopping_train,utrecht,4)
     regtrain1(8,7,utrecht,intercity)
     regtrain1(8,9,tilburg,intercity)
     regtrain1(8,12,utrecht,stopping_train)
     regtrain1(8,16,tilburg,stopping_train)
     regtrain1(8,37,utrecht,intercity)
     regtrain1(8,39,tilburg,intercity)
     regtrain1(8,42,utrecht,stopping_train)
     regtrain1(8,46,tilburg,stopping_train)
     regtrain1(9,7,utrecht,intercity)
     regtrain1(9,9,tilburg,intercity)
     regtrain1(9,37,utrecht,intercity)
     regtrain1(9,39,tilburg,intercity)
     regtrain1(9,42,utrecht,stopping_train)
     regtrain1(9,46,tilburg,stopping_train)
     irregtrain(8,57,utrecht,intercity,6)
negative tuples:
     [...]
integrity constraints:
     train=[irregtrain,regtrain]
     regtrain=platform><regtrain1
? **switch horn.**
horn is now on.
? **show ics all.**
integrity constraints:
train(A,B,C,D,E) :-
        (   irregtrain(A,B,C,D,E)
        ;   regtrain(A,B,C,D,E)
        ).
regtrain(A,B,C,D,E) :-
        platform(D,C,E),
        regtrain1(A,B,C,D).
```

The user might now proceed to induce dependencies for `regtrain1` such as $\emptyset \twoheadrightarrow$*Hour*, and separate the regular trains into hourly trains and non-hourly trains.