Aufgabe 7-1 Forward- and Backward-Reasoning

Discuss the difference between forward and backward reasoning by considering the phenomenon pressure to buy (deutsch: Kaufzwang).

Lösungsvorschlag:

A forward-rule could be:

\[ \text{shop-has}(X, Y) \Rightarrow \text{buy}(Y) \]

The corresponding backward-rule would be

\[ \text{buy}(Y) \Leftarrow \text{shop-has}(X, Y) \]

The forward-rule would trigger for every shop \( X \) which has some \( Y \) to sell, that you buy \( Y \). This causes an unlimited pressure to buy.

The backwards-rule would only be applied if you actually want to buy some \( Y \). It checks all shops \( X \) if they have \( Y \). The first shop with \( Y \) is chosen.

Aufgabe 7-2 Closed-World Assumption

Discuss possibilities to introduce the Closed-World Assumption into the Tableau Calculus for \( \mathcal{ALC} \). What would be the consequences.

Lösungsvorschlag:

Closed-World Assumption is typically incorporated through negation-by-failure. That is, in order to negate a statement, it is tried to prove it positively, and if this fails, the negation is assumed to be proved.

In \( \mathcal{ALC} \) you have a T-Box, an A-Box and a claim to be proved. This is negated, and the Tableaux calculus is then supposed to close all branches. If a branch remains open, then the proof test failed first. Now one can try to close the open branch by negation-by-failure in order to prove a negation for a statement \( x : C \) or \( x \ chicks \) occurring in the branch. Closed-World Assumption is normally only valid for very specific assumptions, e.g. for timetables.

For example: \( \text{bus273 drives-at 10:00 o’clock} \). Suppose the open branch contains, for example, \( \text{bus273 drives at 11:00 am} \). Then one might try to prove its negation. To do this you start a new tableau with the T-Box and the A-Box and \( \text{bus273 \neg drives-at 11:00 am} \). If this contradiction proof fails, the unnegated version \( \text{bus273 drives at 11:00 am} \) cannot be proved. Therefore you can conclude by Closed-World Assumption that actually \( \text{bus273 \neg drives-at 11:00 am} \) holds, and thus close the open branch in the first tableau.

The open branch in the second tableau could, however, be closed by the Closed-World Assumption again, so that a third tableau would be needed, etc.

Therefore the method might not terminate.
Aufgabe 7-3  Monotonic Logic
Does the closed-world assumption cause a monotonic or a non-monotonic logic?

Lösungsvorschlag:
A logic is monotonic when adding a new fact to the assumptions keeps the the previously
derived facts still valid.
With Closed-World Assumption, however, adding a fact, for example, *bus273-at 11 o’clock*, a
negation-by-failure proof attempt which previously failed might now succeed. Therefore it is
now longer justified to assume the negation. Therefore, adding a new fact can invalidate the
facts so far derived.

Aufgabe 7-4  Datalog, Range-Restriction
What would change in the implementation of a Datalog system if the requirement that all
variables in the head of a rule also occur in the body of the rule (Range Restriction)?

Lösungsvorschlag:
One could derive new facts that contain variables. One would then have to search all previous
facts in order to find and delete any instances.
For example, if $p(X)$ is derived, then you should delete $p(a)$ and $p(b)$.
Moreover, the matching algorithm would be more complex for the body-literals.
Example: Rule
$p(X) : - q(a, X)$.
Now if a fact $q(Y, Y)$ with variable $Y$ appears, then the matching algorithm has to bind $X$ to $a$.
This all is possible, but reduces the efficiency.

Aufgabe 7-5  Graph-Coloring
Try to code the graph-coloring problem (with 3 colors) in Datalog, and give two reasons why
this must fail.

Lösungsvorschlag:
An Approach could be:

\[
\begin{align*}
\text{red}(X) &: - \text{neighbour}(X, Y), \neg\text{red}(Y) \\
\text{green}(X) &: - \text{neighbour}(X, Y), \neg\text{green}(Y) \\
\text{yellow}(X) &: - \text{neighbour}(X, Y), \neg\text{yellow}(Y) \\
\text{neighbour}(a, b) &: - \\
\text{neighbour}(a, c) &: - \\
\ldots &: -
\end{align*}
\]

The first reason why this has failed is that it is a search problem where there can be one, several
or even no solution at all. That is, the search can run into a dead end where backtracking is
necessary. This is not built into Datalog.

The second reason is that the clauses are not stratified. For example, in order to prove $\neg\text{red}(b)$,
the whole program would be restarted with the aim of proving that $\text{red}(b)$ can not be derived.
This immediately leads to an endless loop.
Aufgabe 7-6  Contradictions in Datalog

Is it possible to find contradictions in Datalog, and if not, what would have to be changed to make this possible. What would be the consequences?

Lösungsvorschlag:

In Datalog, only positive facts can be derived. There is no contradiction.

Negation in the rule heads could be allowed. This could generate contradictions. In order to find this, one would have for a newly derived fact $F$ to search through all other facts for $\neg F$. This is feasible, but elaborate.

If one wants in addition negation-by-failure, one needs a second negation sign for the negation-by-failure negation.

Aufgabe 7-7  OPS 5

Explain what the following OPS-5 program does. (http://www.99-bottles-of-beer.net/language-ops5-2208.html)

(literalise Count bottles)
(literalise SecondLine)

(startup
  (make Count \^bottles 99)
)

(p moreBeer
  (Count \^bottles {<beerLeft> > 0})
  -(SecondLine)
-->
  (writeln <beerLeft>| bottle of beer on the wall, | <beerLeft> | bottle of beer.|
  (modify 1 \^bottles (compute <beerLeft>-1))
  (make SecondLine)
)

(p moreBeerSecondLine
  (Count \^bottles > 1)
  (SecondLine)
-->
  (writeln |Take one down and pass it around, | <beerLeft> | bottles of beer on the wall.|
  (remove 2)
)

(p oneMoreBeerSecondLine
  (Count \^bottles 1)
  (SecondLine)
-->
  (writeln |Take one down and pass it around, | <beerLeft> | bottle of beer on the wall.|
  (remove 2)
)

(p lastBeerSecondLine
  (Count \^bottles 0)
Aufgabe 7-8  Prolog

What happens when the following Prolog program

\[
p(Z,f(Z)).  
q(X) :- p(X,X).  
\]

is called with a query \( q(Y) \). ? Explain the result.

Lösungsvorschlag:

The answer is \( Y = f(Y) \), which is not really correct, because \( p(f(Y),f(Y)) \) should become equal to \( p(Z,f(Z)) \), which is not possible.

By efficiency reasons one accepts this in Prolog, i.e. one omits the so-called occurs check (whether \( Y \) occurs in \( f(Y) \) is not tested).

However, by omitting the occurs-check one can implement cyclic graph-data structures.