Exercise 7.3 **First-Order-Resolution**

Given is the following formula already presented in exercise 6.4:

$$\forall y.Q(f(a), f(y)) \land \forall xy.(Q(y, f(y)) \to P(f(x), g(y, b))) \to \exists xyz.(P(x, y) \land P(f(a), g(x, b)) \land Q(x, z))$$

This time use the first-order resolution presented in the lecture to show its validity.

See Ex 6.4:

$C_1 = \{ Q(f(a), f(y_1)) \}$

$C_2 = \{ \neg Q(y_2, f(y_2)), P(f(x_2), g(y_2, b)) \}$

$C_3 = \{ \neg Q(x_3, z_3), \neg P(x_3, y_3), \neg P(f(a), g(x_3, b)) \}$

$C_1, C_2 \quad [f(a)/y_2, f(a)/y_1, x_4/x_2]$

$\leadsto C_4 = \{ P(f(x_4), g(f(a), b)) \}$

$C_1, C_3\ [\,f(a)/x_3,\ f(y_2)/z_3,\ y_5/y_3\,]$

$\leadsto C_5 = \{\neg P(f(a), y_5),\ \neg P(f(a), g(f(a), b))\}$

$C_4, C_5\ [\,a/x_4,\ g(f(a), b)/y_5\,]$

$\leadsto C_6 = \square$

Exercise 8.1 **Resolution restrictions**

Resolution w.r.t. a set of support: Let F be a formula in clause form. A subset T of F is called a *set of support (sos)* for F if $F \setminus T$ is satisfiable. Given a sos T of F, a *sos-deduction of the empty clause from* (F, T) is any deduction of the empty clause from F where never two clauses from $F \setminus T$ are resolved.

The idea is that as $F \setminus T$ is satisfiable, we can only deduce the empty clause by using at least one clause which is directly or indirectly obtained from the sos T via resolution ("T supports the deduction of the empty clause."). The sos-restriction therefore allows us to guide the resolution such that it does not "get stuck" within the resolvents of $F \setminus T$.

(a) Formally prove that sos-resolution is complete.

Linear resolution: one of the two clauses must be the resolvent produced in the previous step (no restriction for the first step).

$$\begin{tikzcd}
C_0 \arrow[r] & C_1 \arrow[r] & C_2 \arrow[r] & \cdots \arrow[r] & \square \\
B_0 \arrow[ur] & & B_1 \arrow[ul] & &
\end{tikzcd}$$

Would like to have that $C_0 \in T$, then also a sos-deduction.

Linear resolution: one of the two clauses must be the resolvent produced in the previous step (no restriction for the first step).

Theorem: Linear resolution is complete.

Proof: Let F be unsatisfiable.

$$F = \quad \{A\} \quad \{A, B, C\} \quad \{\neg A, \neg B, \neg C\} \quad \{\neg A, B\} \quad \{\neg B, C\}$$

Let $F' \subseteq F$ be a minimal unsatisfiable subset ( )

$$F' = \quad \{A\} \quad \{\neg A, \neg B, \neg C\} \quad \{\neg A, B\} \quad \{\neg B, C\}$$

We show: for clause C of F' there is a linear derivation of the empty clause starting with $\cancel{C}$. C_0

⇝ By def. of sos, $F \setminus T$ is satisfiable

⇝ For every unsat. core $F' \subseteq F$, we have $F' \not\subseteq F \setminus T$, i.e., $F' \cap T \neq \emptyset$.

⇝ So we can always find $C_0 \in F' \cap T \subseteq F'$.

(b) Consider the following directed graph:

$$\begin{tikzcd} b \arrow[r] & c \arrow[d] \arrow[dl] \\ a \arrow[u] & d \arrow[ul] \end{tikzcd}$$

We can represent the transitive closure of the graph as the following "knowledge base" H in clause form where a, b, c, d are constants, and X, Y, Z are variables (implicitly universally quantified):

$$H = \{\{E(a,b)\}, \{E(b,c)\}, \{E(c,d)\}, \{E(d,b)\}, \{E(c,a)\}\} \cup \{\{\neg T(X,Y), \neg T(Y,Z), T(X,Z)\}, \{\neg E(X,Y), T(X,Y)\}\}.$$

Use sos-resolution to show that a is transitively reachable from d.

SOS: $T_{SOS} = \{\{\neg T(d,a)\}\} \subseteq F := H \cup \{\neg T(d,a)\}$

$C_0 = \{\neg T(d,a)\}$ $\quad B_0 = \{\neg T(X,Y), \neg T(Y,Z), T(X,Z)\}$

$C_1 = \{\neg T(d,Y_1), \neg T(Y_1,a)\}$ $\quad B_1 = \{\neg E(X,Y), T(X,Y)\}$

$C_2 = \{\neg E(d,Y_1), \neg T(Y_1,a)\}$ $\quad B_2 = \{E(d,b)\}$

$$C_3 = \{\neg T(b,a)\}$$

Exercise 8.2

(a) On the webpage, you can find the otter-input files for the examples "ape" and "towers of hanoi" discussed in the lecture.

Try to extend the "ape" example analogously to the "towers of hanoi" example such that otter tells you how the ape can reach the banana. See also the remarks in the "ape" input file.

Recall Towers of Hanoi

solve(No, Src, Dst, Aux, Instr)

number of discs to move

from source to destination via auxiliary stack

by issuing the instructions

Idea:
- move No-1 discs from Src to Aux via Dst,
- then move largest/lowest disc from Src to Dst,
- then move No-1 discs from Aux to Dst via Src.

Constants :

$\underbrace{\text{left, right, middle,}}_{\text{disc stacks}}$ $\underbrace{\text{empty}}_{\text{list}}$, $\underbrace{0}_{\text{No discs}}$

Functions :

s( ) : successor function for encoding $\mathbb{N}$

move(,) : move topmost disc from 1st arg to 2nd arg

li(,) : list built by taking 1st arg as head and 2nd arg as tail

Predicates :

solve(, , , ,) : as described

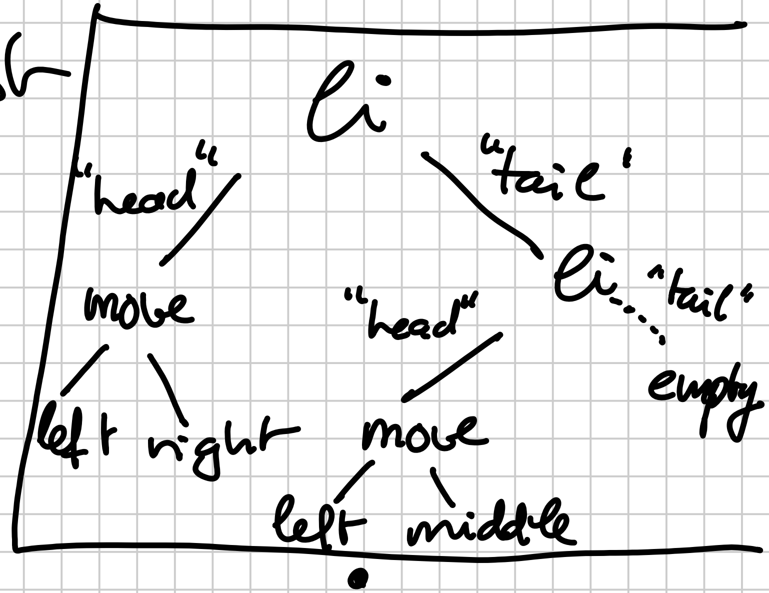
append(, ,) : true if 1st arg is the list obtained by appending the 3rd arg to the 2nd.

source ← destination

Fact: solve(0, S, D, A, empty). % always true

aux

% handling the instruction list



append(X, empty, X). %

result

append(li(H,R), li(H,T), L)

:- append(R, T, L).

% given a list li(H, T) with head H and tail T, append L to it, by appending L to T.

% moving discs

source

solve(s(N), S, D, A, I)

succ

```
:- solve(N, S, A, D, I1)
,  solve(N, A, D, S, I2)
,  append(I, I1, li(m(S,D), I2)).
```

```
?- solve( s(s(s(s(0)))), left, right, middle, I).
```

Ape - Chair - Banana

Predicate: situation (A , C , B , I).

position of ape , chair , banana ; Instruction list.

„append" as before

situation (posA , posC , posB , e).

done (I) |- situation (onchair , c , c , I1)

|- append (I , I1 , li (grabBanana , empty)).

situation (A , C , B , I)

|- situation (Aold , C , B , I1)

|- append (I , I1 , li (move (Aold , A) , e)).

(b) A farmer accompanied by a large radish, a sheep, and a dog needs to cross a river. Unfortunately, he only has a raft which can accommodate at most two of the four objects mentioned (recall, it's a large radish). To state the obvious, only the farmer can steer the raft. The problem, of course, is that the dog is quite hungry and would love to eat the sheep, while the sheep is more than willing to feast on the radish. Thus, he needs to find a way to cross the river in such a way that at no point of time the dog is left alone with the sheep, and analogously for the sheep and the radish.

Analogously to the "ape" example, use otter to find a sequence of commands to guide the farmer and his belongings safely across the river.

situation(left, left, left, left, empty).

Farmer Dog Sheep Radish history/ instruction list

done(I) |- situation(right, ..., right, I).

% possible moves: - farmer changes sides alone using his raft

% - farmer takes exactly one obj. with him

```
% farmer moves alone
situation(F, D, S, R, I)
 |- situation(Fold, D, S, R, I1)
 |- append(I, I1, li(moveAlone(Fold, F),
                     empty))
 |- safe(F, D, S, R).
% farmer moves the dog
situation(F, F, S, R, I)
 |- situation(Fold, Fold, S, R, I1) |- safe(F, F, S, R)
 |- append(I, I1, li(moveSheep(Fold, F), empty)).
```

```
safe(F, D, S, R)
  :- sheepSafe(F, D, S, R)
  :- radishSafe(F, D, S, R).

sheepSafe(F, D, F, R).
sheepSafe(F, D, S, R)
  :- separated(D, S).

separated(left, right).

separated(right, left).
```

;

:

;

;

(c) Otter is accompanied by mace, a tool which enumerates all finites structures for a given formula until a model (or counterexample) is found.

Try to use mace in order to find a solution to the problem described in (b).

Exercise 8.3

Express the join $R_1 \ltimes_{i=j} R_2$ (as defined in the slides) by means of selection σ and projection π.

$$\begin{aligned}
\sigma_{B(\mathbf{x}')}(R) &= R(\mathbf{x}) \wedge B(\mathbf{x}') \quad \text{mit } \mathbf{x}' \subseteq \mathbf{x} \\
\pi_{\mathbf{x}'}(R) &= \exists\, \mathbf{x}''\, R(\mathbf{x}) \quad \text{mit } \mathbf{x}' \subseteq \mathbf{x},\ \mathbf{x}'' = \mathbf{x} \setminus \mathbf{x}' \\
(R_1 \cup R_2) &= R_1(\mathbf{x}) \vee R_2(\mathbf{x}) \\
(R_1 - R_2) &= R_1(\mathbf{x}) \wedge \neg R_2(\mathbf{x}) \\
(R_1 \times R_2) &= R_1(\mathbf{x}) \wedge R_2(\mathbf{y}) \\
(R_1 \ltimes_{i=j} R_2) &= \exists z \;\; R_1(x_1, \ldots, x_{i-1}, z, x_{i+1}, x_n) \wedge \\
&\qquad\quad R_2(y_1, \ldots, y_{j-1}, z, y_{j+1}, y_m)
\end{aligned}$$

$$\pi_{x \cup y - \{x_i, y_j\}}\left(\sigma_{x_i = y_j}\left(R_1(x) \times R_2(y)\right)\right) =: R_1 \bowtie_{i=j} R_2$$