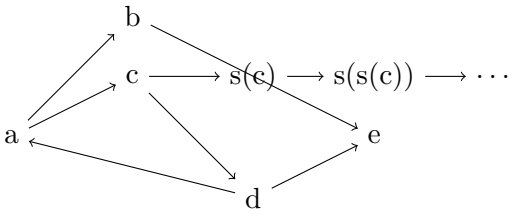
Hinweis: Die Lösungen sind per Email an ha-lp.mmis@uni-rostock.de bis spätestens **12. Juli 2019 um 23:59 Uhr (CEST)** abzugeben.

- Geben Sie Ihren Quelltext als Prolog-Datei mit dem Namen (Die Gruppennummer immer entsprechend anpassen) `Gruppe-01-HA4.pl` ab.
- Zusätzliche Erläuterungen können als PDF (maschinengeschrieben) abgegeben werden.
- Stellen Sie alle anderen Lösungen kurz und präzise in einem PDF dar.
- Achten Sie darauf, dass Ihre Prolog-Datei erfolgreich in SWI-Prolog (wie im Rechnerpool installiert) geladen werden kann. Lösungsideen können Sie im Kommentar oder im PDF skizzieren.
- Die Bearbeitung muss in Gruppen von 3 bis 5 Personen erfolgen.
- Für alle Gruppenmitglieder ist der Name, die Matrikelnummer und der Studiengang in allen Dokumenten anzugeben, d.h. sowohl in der Email, dem PDF-Dokument als auch als Kommentar im Quelltext.

1. (5 Punkte) Gegeben sei folgende Definition eines gerichteten Graphen g_1 (Abbildung rechts) in Prolog:

```
1 next(g1, a, b).
2 next(g1, a, c).
3 next(g1, b, e).
4 next(g1, c, s(c)).
5 next(g1, s(C), s(s(C))).
6 next(g1, c, d).
7 next(g1, d, a).
8 next(g1, d, e).
```



Der erste Parameter identifiziert den beschriebenen Graphen, der zweite und dritte Parameter die Verbunden Knoten in diesem Graphen.

Sei weiterhin folgendes rekursives Programm zum Finden aller Wege im Graphen G gegeben:

```
1 path(G, S, Z, P) :- path(G, S, Z, [], P).
2 path(G, S, Z, _, [S, Z]) :- next(G, S, Z).
3 path(G, S, Z, V, [S|P]) :- next(G, S, T), \+ member(T, V), path(G, T, Z, [S|V], P).
```

(Die Variable `V` beinhaltet eine Liste aller besuchten Knoten und verhindert, dass ein Knoten mehrfach besucht wird).

   a) Welche Wege gibt es im Graphen von `a` nach `e` (ohne Knoten mehrfach zu besuchen)?

   b) Welche Lösungen findet Prolog mit dem gegebenen Programm und der Anfrage `path(g1, a, e, P).`? Warum werden mit Prologs Beweisstrategie nicht alle Lösungen gefunden?

2. (20 Punkte) In der Vorlesung wurde ein einfacher Meta-Interpreter für Prolog vorgestellt:

```
1 prove(true) :- !.
2 prove(member(E, L)) :- !, member(E, L). % noetig, weil member built-in ist
3 prove((Goal1, Goal2)) :- !, prove(Goal1), prove(Goal2).
4 prove(Goal) :- clause(Goal, Body), prove(Body).
```

Dieser Meta-Interpreter simuliert die Beweisstrategie von Prolog, unterstützt aber keine Meta-Prädikate wie `\+` (entspricht dem `not`).

a) Erweitern Sie den Meta-Interpreter um eine zusätzliche Regel, sodass er Negation as Failure mit dem Prädikat `\+` unterstützt. Vergewissern Sie sich, dass ihr Programm mit der Anfrage `prove(path(g1, a, e, P)).` die gleichen Ergebnisse wie Prolog aus Aufgabe 1b) liefert liefert.

b) Erweitern Sie die einzelnen Regeln des Meta-Interpreters um eine Ausgabe des jeweils aktuellen Ziels, der betrachteten Regeln und der aktuellen Suchtiefe. Beispielausgabe:

```
?- prove(path(a, e, P)).
0 try rule: path(g1,a,e,_8432) -> path(g1,a,e,[],_8432)
1 try rule: path(g1,a,e,[],[a,e]) -> next(g1,a,e)
1 try rule: path(g1,a,e,[],[a|_8696]) -> next(g1,a,_8704),
                            \+member(_8704,[]),path(g1,_8704,e,[a],_8696)
2 goals: next(g1,a,_8704),\+member(_8704,[]),path(g1,_8704,e,[a],_8696)
2 try rule: next(g1,a,b) -> true
2 goals: \+member(b,[]),path(g1,b,e,[a],_8696)
2 NAF: member(b,[])
2 try rule: path(g1,b,e,[a],[b,e]) -> next(g1,b,e)
3 try rule: next(g1,b,e) -> true
P = [a, b, e]
```

Nutzen Sie, ähnlich wie im Vorlesungsskript 6, Folie 7, das Prädikat `write/1` zur Ausgabe von Text.

c) Aktuell verwendet der Meta-Interpreter, genau wie Prolog, die Strategie der Tiefensuche. Modifizieren Sie den Meta-Interpreter, sodass die Tiefensuche nur bis zu einer bestimmten Tiefe `Max` sucht. Das neue Prädikat soll die Form `prove(Max, Tiefe, G)` haben, wobei `Tiefe` am Anfang immer auf 0 gesetzt wird. Welche Lösungen finden Sie mit `prove(20, 0, path(g1, a, e, P)).` jetzt?

3. (10 Punkte) Gegeben sei folgendes Puzzle:

> *Drei Missionare stehen mit drei Kannibalen am Nordufer eines Flusses. Um ihn zu überqueren können sie ein Boot benutzen, das aber höchstens drei Personen zugleich trägt. Bei der Überquerung dürfen niemals Kannibalen in der Überzahl sein, weder im Boot, noch am einen oder anderen Ufer, da sonst die Gefahr besteht, dass sie ihren alten Sitten folgen und ein Festmahl bereiten. Wie kommen die Missionare mit ihren Gästen heil ans andere Ufer?*

Lösen Sie das Puzzle mithilfe der oben angegebenen Pfadsuche `path` indem Sie das `next`-Prädikat entsprechend definieren. Benutzen Sie zur Darstellung des aktuellen Zustandes einen Term der Form `s(MN,KN,MS,KS,Boot)` um die Anzahl der Missionare am Nordufer (`MN`), die Anzahl der Kannibalen am Nordufer (`KN`) die Anzahlen am Südufer und die Position des Bootes (`nord` bzw. `sued`) zu spezifizieren. Vermeiden Sie eine explizite Auflistung aller Zustände, sondern nutzen Sie Regeln der Form

```
next(kannibalen, s(MN, KN, MS, KS, nord), s(MN2, KN2, MS2, KS2, sued)) :-
    % Boot faehrt von nord nach sued
    % wie veraendern sich die anderen Variablen?
```

4. **Zusatzaufgabe** (10 Punkte) Ändern Sie Ihr `next`-Prädikat so, dass

```
1 next(kannibalen(BootKap), s(MN1,KN1,MS1,KS1,B1), s(MN2,KN2,MS2,KS2,B2)) :- ...
```

als Parameter `BootKap` die Kapazität des Bootes mit angegeben werden kann und lösen Sie das Kannibalenproblem für 4, d.h., ermitteln Sie die Lösung für:

```
1 ?- connected(kannibalen(3), s(4,4,0,0,nord), s(0,0,4,4,sued), P).
```