Zusammenfassung der Lesarten (Kapitel 3) und Fragen aus vergangenen Abgabegesprächen.

Lesarten

Was beschreibt das Programm?

Wie wird das Programm ausgeführt?

Terminiert das Programm?

Wie effizient läuft das Programm ab?

Lesarten wozu?

- Um Programme zu verstehen und zu begründen, warum etwas der Fall ist oder nicht der Fall ist.

Informelle Lesart

vorfahre_von/2

- Prädikat kann als deutscher Satz gelesen werden.
- Bei komplexeren Prädikaten umständlich, da Prolog mit Variablen arbeitet, man in natürlicher Sprache aber verschiedene Konstrukte verwenden muss, um sich auf bereits erwähnte Dinge oder Personen zu beziehen.
- Wird auch bei langen Definitionen umständlich.

Deklarative Lesart

Was beschreibt das Programm?

- Beschreibt, was das Programm logisch bedeutet.
- Unabhängig davon, wie Prolog intern arbeitet.
- Programm kann wie logische Formeln analysiert werden.
- Definition eines Prädikats kann **spezialisiert oder verallgemeinert** werden.

Prädikat besteht aus Klauseln. **Jede Klausel beschreibt einen Teil der Lösungsmenge** des Prädikats.

Spezialisierungen

Wie?

1. **Streichen von Klauseln/Regeln,** um Lösungen aus der Lösungsmenge wegzunehmen. → Verbleibende Definition beschreibt nun nur mehr einen Teil der Lösungsmenge.
   z.B. Klauseln das Ziel false/0 hinzufügen.
2. **Weiteres Ziel hinzufügen.**

Verallgemeinerungen

Wie?

- Vernachlässigung eines Ziels (* vor das Ziel setzen).
- Mehr verallgemeinernde Regeln hinzufügen.

Schließende Lesart

Wie man Regeln lesen kann.

- Man liest zuerst den Rumpf (= Ziele des Regelkörpers) und vergisst den Regelkopf.
- Wir wollen aus dem Rumpf etwas folgern.
- NICHT wichtig zu verstehen, wie was abläuft, sondern WAS eine Regel bedeutet.

Beispiel: *Wir wissen:*

1. *Person ist Vorfahre eines Nachfahren.*
2. *Person ist ein Kind einer weiteren Person.*

... dann → ist diese weitere Person auch Vorfahre jenes Nachfahren (= Regelkopf).

Stimmt Prädikat nun nicht mit der Intention überein, ist die Definition entweder zu allgemein oder zu speziell:

1. **Zu allgemein:** Das Prädikat ist für die Anfrage erfüllt, die scheitern sollte.
2. **Zu speziell:** Das Prädikat scheitert für eine Anfrage, die erfüllt sein sollte.

Prozedurale Lesart

Wie wird das Programm ausgeführt?

- Beschreibt wie eine Schlussfolgerung erreicht wird (**Inferenz**).
- Um ein Ziel zu beweisen, selektiert Prolog eine Klausel desselben Prädikats und ersetzt das Ziel durch den Regelkörper der Klausel. Diese Ersetzung heißt Inferenz.
- Um ein Ziel (eine Anfrage) zu beweisen, ersetzt Prolog eine Anfrage durch eine Regel, wenn die Regel das gleiche Prädikat hat. Das ist die **Inferenz**.

Beispiel:

```
verschwistert_mit(Kind1, Kind2) ←
    kind_von(Kind1, Elternteil),
    kind_von(Kind2, Elternteil).

← verschwistert_mit(joseph_II, Person).
```

Das Ziel der Anfrage wird durch den Regelkörper der Klausel verschwistert_mit/2 ersetzt. Dabei werden entsprechende Variablen gleichgesetzt also gebunden.

```
← kind_von(joseph_II, Elternteil), kind_von(Person, Elternteil).
```

Termination bzw. Nichttermination des Ziels:

Zusicherung		deklarativ	prozedural
← Ziel.	:-	wahr	es findet sich (zumindest) eine Lösung
⊬ Ziel.	:/-	falsch	endliches Scheitern, Termination
⊬ Ziel, false.		(falsch)	Termination
$\overset{\infty}{\leftarrow}$ Ziel.	:-&	wahr	wegen Endlosableitung keine Lösung
$\overset{\infty}{\not\vdash}$ Ziel.	:/-&	falsch	wegen Endlosableitung kein Scheitern
$\overset{\infty}{\not\vdash}$ Ziel, false.		(falsch)	Ziel terminiert nicht
$\overset{\$}{\leftarrow}$ Ziel.	:-$	wahr	aufwendige Lösung
$\overset{\$}{\not\vdash}$ Ziel.	:/-$	falsch	aufwendiges (aber endliches) Scheitern
$\overset{\$}{\not\vdash}$ Ziel, false.		(falsch)	aufwendige Termination

- **Endlosableitungen** werden durch (direkt oder indirekt) rekursive Regeln verursacht.

Beispiel:

~~vorfahre_von_2(Vorfahre, Nachfahre) ←~~ false,
~~kind_von(Nachfahre, Vorfahre)~~.
vorfahre_von_2(Vorfahre, Nachfahre) ←
vorfahre_von_2(Person, Nachfahre),
kind_von(Person, Vorfahre).

Endlosrekursion möglich?

- *Beschränkung nur auf jene Teile des Programms, die Rekursionen enthalten, deshalb wird erstes Prädikat gestrichen.*
- ***Variable Vorfahre** nicht im ersten Ziel verwendet → kann daher **keinen Einfluss** darauf haben, ob Endlosableitungen auftreten.*
- ***Variable Nachfahre** hat auch keine Bindungen, die die Endlosrekursion aufhalten könnten.*
- ***Antwort: Ja***

Beim Testen von Programmen können nur Programmierfehler gefunden werden, es kann nicht garantiert werden, dass keine Fehler mehr im Programm sind.

Prädikat false/0 ist ein vordefiniertes Prädikat, das falsch ist — also immer scheitert.

- Negative Zusicherung ***:/- Ziel, false.*** → **Ziel terminiert**.
- Es kann nicht immer zugesichert werden, ob Programm stets terminiert (z.B. Prädikate mit unendlich großen Lösungsmengen).

~~vorfahre_von(Vorfahre, Nachfahre) ←~~ false,
~~kind_von(Nachfahre, Vorfahre)~~.
vorfahre_von(Vorfahre, Nachfahre) ←
kind_von(Person, Vorfahre),
vorfahre_von(Person, Nachfahre).

- *Erst kind_von/2 bewiesen → terminiert.*
- *Ob **vorfahre_von/2 terminiert** hängt von den **Lösungen von kind_von/2** und der **Variable Vorfahre** ab.*
- *Termination unabhängig von Variable Nachfahre.*

:/-&

1. Unendlich große Mengen an Antwortsubstitutionen → Ziel darf nicht terminierten.
2. Probleme mit Terminationsverhalten → Antwortsubstitutionen endlich, aber Berechnung nicht in endlicher Zeit möglich.

Effizienzüberlegungen

- Größe der Lösungsmenge
- Anzahl der Inferenzen
- Termgröße

Größe der Lösungsmenge

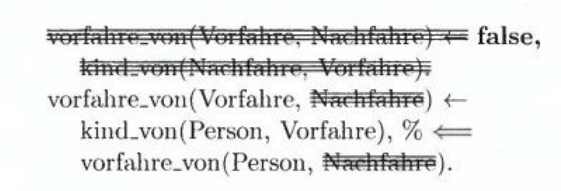
- Lösungsmenge wird durch jedes Ziel beschrieben.
- Je weniger Argumente, umso größer die Lösungsmenge, umso aufwendiger der Beweis.

Anzahl der Inferenzen

Anzahl der erforderlichen Inferenzen abschätzen ähnlich wie Terminationsüberlegungen:

- Nichtrekursive Regeln vernachlässigen.

Beispiel:



Ziel ←vorfahre_von(joseph_II, Nachfahre).

- *Für **kind_von/2** so viele Lösungen, wie es Kinder von Joseph II gibt.*
- *Für die nächste Inferenz ist das erste Argument gebunden, das zweite wird nur durchgereicht (keinen Einfluss).*

→Anzahl der benötigten Inferenzen hängt davon ab, wie viele Nachfahren Joseph II hat.

Termgröße

Ausführungszeit eines Ziels verhält sich oft in etwa proportional zur Größe der verwendeten Terme.

Fragen aus vergangenen Abgabegesprächen

Unterschied zwischen deklarativer und prozeduraler Lesart

1. Hauptunterschied

- Die **deklarative Lesart** sagt uns, **was das Programm bedeutet** – also die logische Bedeutung der Regeln.
- Die **prozedurale Lesart** sagt uns, **wie das Programm ausgeführt wird** – also welche Schritte Prolog macht, um eine Lösung zu finden.

2. Termination (Wann das Programm stoppt)

- **Deklarativ gesehen** kann ein Prolog-Programm völlig korrekt sein, also die richtigen logischen Aussagen enthalten.
- **Aber prozedural gesehen** kann es sein, dass Prolog nie zu einem Ergebnis kommt, weil die Regeln in einer ungünstigen Reihenfolge stehen oder sich selbst unendlich oft aufrufen.

Abgabegespräch 2025: Er hat gefragt, was die deklarative und prozedurale Lesart gemeinsam haben. Es reicht nicht die deklarative Lesart zu erklären und zu sagen, dass die prozedurale Lesart noch Termination und Nicht-Termination berücksichtigt.

Beispiel: Zusicherung :- bruder_von(alex, alex).

Warum scheitert Zusicherung?

- Definition analysieren.

Wie macht man Zusicherung geltend?

- zusätzliches Faktum: bruder_von(alex, alex).
- zusätzliche Regel: bruder_von(B, B).
- dif Ziel entfernen.
- Zusicherung negativ machen.
- Statt eines Funktionssymbols eine Variable einfügen (:- bruder_von(alex, A).).

Abgabegespräch 2025: Er wollte allgemein wissen, wie die Begründung für eine gescheiterte Zusicherung aussieht (Definition mit zum Teil durchgestrichenen Zeilen). Dann was die durchgestrichenen Zeilen bedeuten. Dass damit aufgezeigt wird, wo die Zusicherung scheitert, ist falsch. Er meinte es gibt 3 Arten/Gründe wie/warum Sachen durchgestrichen werden. – Was er damit meinte, hat er leider nicht gesagt.

Lösungsmenge vs. Lösungssequenz

Die **Lösungsmenge** ist die Menge aller möglichen gültigen Lösungen einer Anfrage.

Die **Lösungssequenz** ist die Reihenfolge, in der Prolog die Lösungen während der Suche berechnet und ausgibt.

- Prolog produziert nur Lösungssequenzen.
- D.h. dass gelegentlich **redundante Lösungen** generiert werden.
- Nur störend, wenn sie unendlich oft in der Lösungssequenz vorkommen.
- Entstehen meist durch Variablen, die im Regelrumpf vorkommen (**existentielle** oder **interne Variablen**).

Wie kann man anhand einer Lösungsmenge entscheiden, ob die Lösungssequenz unendlich oder endlich ist?

- Wenn die **Lösungsmenge endlich** ist, muss die **Lösungssequenz endlich** sein.
- **Lösungsmenge unendlich**:
  1. Es gibt eine endliche Darstellung der unendlichen Lösungsmenge: **Lösungssequenz endlich**.
     z.B. die Menge aller einelementigen Mengen.
  2. Sonst **Lösungssequenz unendlich**.