Handlungsplanung und Allgemeines Spiel

„Game Description Language (GDL)“

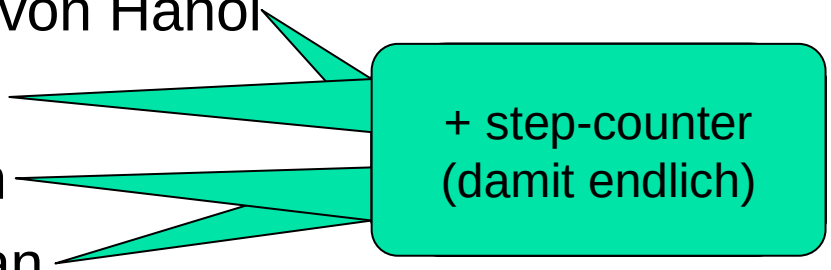
Peter Kissmann

General Game Playing

- Einschränkungen des derzeitigen Formalismus: Spiele sind
  - endlich
  - diskret
  - deterministisch
  - vollständige Information für alle Spieler
- Ein- und Mehrpersonenspiele
  - Spielerzahl endlich und fest (von Start bis Ende)
- Gleichzeitige und abwechselnde Züge
  - Immer ≥ 1 Zug pro Spieler
  - standardmäßig nur simultane Züge
  - nicht-simultan: sog. “noops”
    - Züge, in denen die nicht-aktiven Spieler nichts tun

Mögliche Spiele

- Vier Gewinnt
- Tic-Tac-Toe
- Solitaire
- Türme von Hanoi
- Halma
- Schach
- Sokoban
- etc.



Unmögliche Spiele

- Kartenspiele (Skat, Poker, Doppelkopf etc.) → unvollständiges Wissen
- Schiffe versenken → unvollständiges Wissen
- Würfelspiele (Kniffel, Backgammon, Mensch ärgere Dich nicht etc.) → Zufall (nicht deterministisch)
- Computerspiele → Zufall, unvollständiges Wissen, evtl. kontinuierlich, nicht endlich (?)
- Pen-and-Paper Rollenspiele → Zufall, unvollständiges Wissen, nicht endlich
- etc.

Themen Allgemeines Spiel

- Einführung
- Game Desciption Language (GDL)
- Spielen allgemeiner Spiele
- Heuristiken im allgemeinen Spiel und Verbesserungen
- Lösen allgemeiner Spiele
- Instanziierung
- Ausblick: Unvollständige Information und Zufall

Ablauf

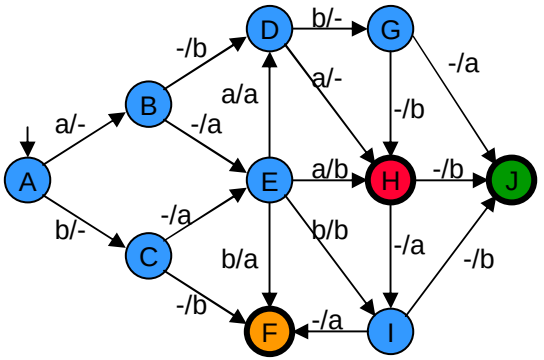
- Beschreibungsformen
- Spiel-Modell
- Logische Spielbeschreibung
- Elemente von GDL
- Wichtige Eigenschaften von GDL

Mögliche Beschreibungsformen

- Jede Form muss Möglichkeit bieten,
  - Legalität von Zügen,
  - Zustandsübergänge,
  - Terminierung und
  - Bewertungen zu bestimmen

Mögliche Beschreibungsformen

- Endliche Automaten
  - Problem: Riesige Zustandsräume
    - etwa 5000 in Tic-Tac-Toe
    - etwa 4,5 x ^10^{12} in Vier Gewinnt
    - etwa 10^{13} im 15-Puzzle
  - damit riesiger Speicherbedarf → nicht geeignet



Mögliche Beschreibungsformen

- Listen und Tabellen
  - haben gleiche Größe wie endliche Automaten
  - ebenfalls ungeeignet

Mögliche Beschreibungsformen

- Programme
  - zum Generieren gültiger Züge und Nachfolgerzustände
  - zur Evaluation der Terminierung und der Bewertungen
  - Welche Sprache? Java, Lisp, C, …?
  - Schwierig, wenn Spieler allgemeine Struktur des Spiels untersuchen möchte

Mögliche Beschreibungsformen

- Logik
  - existierende Interpreter / Compiler
  - einfacher zu analysieren als prozedurale Beschreibungen
  - heutzutage zumeist verwendet (“Game Description Language”)

Mögliche Beschreibungsformen

- Außer Logik darf nichts angenommen werden
  - keine Arithmetik
  - keine Physik
  - kein gesunder Menschenverstand
  - daher möglich, Spiele auch mit absurden Bezeichnern zu beschreiben
    - wird bei Meisterschaften zumeist auch explizit so gemacht

Logisches Folgern

- Minimalanforderungen:
  - Berechnung gültiger Züge
  - Berechnung von Effekten von Zügen
  - Wissen, wann Spiel endet
  - Wissen, wie viele Punkte erhalten wurden (wann welches Ziel erreicht wurde)
- weitere Möglichkeiten:
  - Vereinfachung
  - Optimierung
  - Erkennung spezieller Strukturen

Endliche Spiele

- Endliche Umgebung
  - Spiel-”Welt” mit endlich vielen Zuständen
  - 1 Initialzustand, ≥ 1 Terminalzustand
  - Festgelegte endliche Anzahl an Spielern
    - endlich viele Züge für jeden in jedem Zustand
    - ≥ 1 Zielzustand (= Terminalzustand mit Maximalbewertung) für jeden
- Kausales Modell
  - nur Züge ändern Umgebung
  - Synchrone Züge

Spiel-Modell

- Ein n-Personen Spiel ist ein Tupel mit
  - S - Menge von Zuständen
  - $A_1, \ldots, A_n$ - n Mengen von Zügen, eine für jeden Spieler
  - $l_1, \ldots, l_n$ - mit $l_i \subseteq A_i \times S$, legal Relationen (Bestimmung gültiger Züge)
  - $u: S \times A_1 \times \ldots \times A_n \rightarrow S$ - Update-Funktion
  - $s_1 \in S$ - Initialzustand
  - $t \subseteq S$ - Terminalzustände
  - $g_1, \ldots, g_n$ - mit $g_i \subseteq S \times \mathbb{N}$, goal Relationen (Bestimmung der Bewertungen)

Direkte Beschreibung

- Spiele endlich → möglich, Spielinformationen durch Listen (von Zuständen und Zügen) und Tabellen (für Gültigkeit, Update etc.) zu kommunizieren
- Problem: Größe der Beschreibung
- Lösungen:
  - modulare Umschreibung
  - kompakte Kodierung

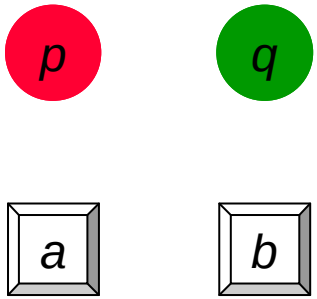
Zustände und Fluents

- Oft, Welt als Menge von Fluents, also Eigenschaften, die sich ändern können, etwa
  - “Position des schwarzen Königs”
  - “Schwarz kann eine Rochade durchführen”
- Züge betreffen Teilmenge dieser Fluents

Zustände und Fluents

- Zustände
  - repräsentieren alle Möglichkeiten, wie Welt aussehen kann
  - Zustandsanzahl exponentiell in Zahl der Fluents in der Welt
  - Tabellen für Züge etc. entsprechend groß
- Lösung:
  - Fluents direkt repräsentieren
  - Beschreiben, wie Züge einzelne Fluents ändern (statt gesamte Zustände)

Buttons & Lights

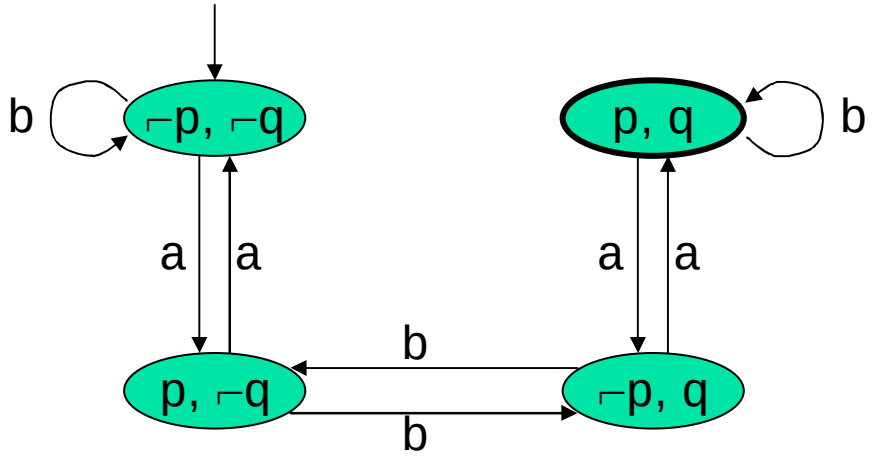


- Drücken von *a* ändert *p*
- Drücken von *b* vertauscht *p* und *q*
- Initial, *p* und *q* aus
- Ziel: *p* und *q* an

Automat für Buttons & Lights

Fluents: p, q

Züge: a, b



Spielbeschreibung für Buttons & Lights

- Gültige Züge

```
legal(robot, a)
legal(robot, b)
```

- Update

```
next(p) <= does(robot, a) ∧ ¬true(p)
         ∨ does(robot, b) ∧ true(q)
next(q) <= does(robot, a) ∧ true(q)
         ∨ does(robot, b) ∧ true(p)
```

- Terminierung und Ziel

```
terminal <= true(p) ∧ true(q)
goal(robot, 100) <= true(p) ∧ true(q)
```

Relationale Logik

- Objektvariablen: `X`, `Y`, `Z`
- Objektkonstanten: `a`, `b`, `c`
- Funktionskonstanten: `f`, `g`, `h`
- Relationskonstanten: `p`, `q`, `r`, `distinct`
- Logische Operatoren: ¬, ∧, ∨, `<=`, `<=>`
- Terme: `X`, `Y`, `Z`, `a`, `b`, `c`, `f(a)`, `g(a, X)`, `h(a, b, f(Y))`
- Relationale Sätze: `p(a, b)`
- Logische Sätze: `r(X, Y) <= p(X, Y) ∧ q(Y)`
- Standard Semantik (Prädikatenlogik)

Klausellogik

- Literale:
  - relationale Sätze
  - negierte relationale Sätze
- Klauseln:
  - Fakten: Relationale Sätze
  - Regeln: `Head <= Body`
    - `Head`: relationaler Satz
    - `Body`: Logischer Satz aus $\wedge$, $\vee$, Literal
- Standard Completion Semantik (“Negation as Failure”)

Completion

- Sei P eine endliche Klauselmenge
- Für eine Relation $p(X_1, \ldots, X_n)$ seien

$$p(t_{11}, \ldots, t_{1n}) \leftarrow B_1$$
$$\vdots$$
$$p(t_{m1}, \ldots, t_{mn}) \leftarrow B_m$$

  alle Klauseln in P mit Head p
- Die Completion von p in P ist die Formel

$$p(X_1, \ldots, X_n) \leftrightarrow X_1 = t_{11} \wedge \ldots \wedge X_n = t_{1n} \wedge B_1 \vee \ldots \vee X_1 = t_{m1} \wedge \ldots \wedge X_n = t_{mn} \wedge B_m$$

- Falls m=0 ist die Completion $\neg p(X_1, \ldots, X_n)$

Completion für Buttons & Light

$$
\begin{aligned}
&\neg init(F)\\
&legal(P, A) \Leftrightarrow P=robot \wedge A=a\\
&\qquad\qquad\qquad \vee\ P=robot \wedge A=b\\
&next(F) \Leftrightarrow F=p \wedge (does(robot, a) \wedge \neg true(p)\\
&\qquad\qquad\qquad\qquad \vee\ does(robot, b) \wedge true(q))\\
&\qquad\qquad \vee\\
&\qquad\qquad F=q \wedge (does(robot, a) \wedge true(q)\\
&\qquad\qquad\qquad\qquad \vee\ does(robot, b) \wedge true(p))\\
&terminal \Leftrightarrow true(p) \wedge true(q)\\
&goal(P,V) \Leftrightarrow P=robot \wedge V=100 \wedge true(p) \wedge true(q)
\end{aligned}
$$

Knowledge Interchange Format (KIF)

- KIF ist Standard für Austausch von Wissen repräsentiert in relationaler Logik
- Syntax ist Präfix-Version der Standardsyntax
- einige Operatoren sind umbenannt (`not`, `and`, `or`)
- Case-Independent
- Variablen durch Präfix `?` dargestellt
- Semantik unverändert
- GDL nutzt KIF

Knowledge Interchange Format (KIF)

- damit wird

  ```
  t(X, Y) <= q(X) ∧ q(Y) ∧ ¬s(X, Y)
  ```

  zu

  ```
  (<= (t ?x ?y) (and (q ?x) (q ?y) (not (s ?x ?y))))
  ```

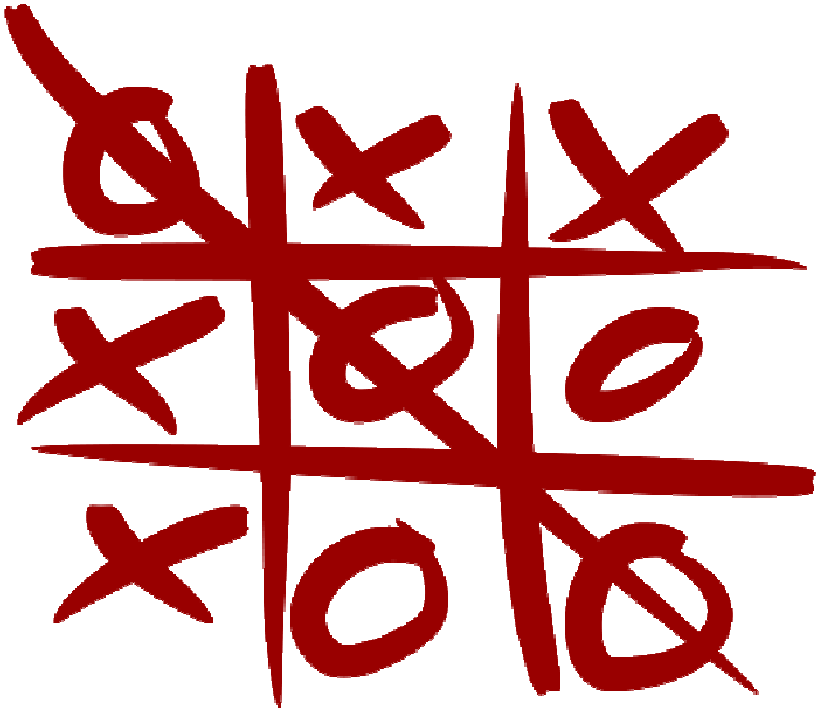
  oder, äquivalent, zu

  ```
  (<= (t ?x ?y) (q ?x) (q ?y) (not (s ?x ?y)))
  ```

Spiel-unabhängiges Vokabular

- Objektkonstanten
  - 0, 1, 2, 3, …
- Relationen
  - `(role player)` Rollenname im Spiel
  - `(true fluent)` wahr im aktuellen Zustand
  - `(init fluent)` wahr im Initialzustand
  - `(does player move)` gewählter Zug
  - `(next fluent)` wahr im Nachfolgeustand
  - `(legal player move)` gültige Züge
  - `terminal` Terminalzustand
  - `(goal player value)` Bewertung

Tic-Tac-Toe



Tic-Tac-Toe Vokabular

- Objektkonstanten
  - `xplayer, oplayer` — Spieler
  - `x, o, b` — Markierungen
- Funktionen
  - `(cell number number mark)` — Fluent (Belegung von Spielbrettzelle)
  - `(control player)` — Fluent (Aktiver Spieler)
  - `(mark number number)` — Zug (Ziel der nächsten Markierung)
  - `noop` — Zug (nichts tun)
- Relationen
  - `(row number mark)` — 3 Markierungen mark horizontal
  - `(column number mark)` — 3 Markierungen mark vertikal
  - `(diagonal mark)` — 3 Markierungen mark diagonal
  - `(line mark)` — 3 Markierungen mark in einer Linie
  - `open` — freie Position auf Spielbrett

Relationen - role

- Definiert Namen der Spieler
- Beispiel Tic-Tac-Toe
  - `(role xplayer)`
  - `(role oplayer)`
  - Zwei Spieler: xplayer und oplayer
- Zur Erinnerung: In Nachricht über gewählte Züge immer xplayer als 1. Spieler und oplayer als 2. Spieler
- Darf nicht head einer Regel mit Body sein
  - Spieler sind von Anfang an festgelegt und ändern sich nicht

Relationen - true

- Repräsentiert ein Fluent das im aktuellen Zustand wahr ist
- Zustand = Menge aller Fluents, die wahr sind
  - nicht spezifizierte Fluents sind falsch
- Beispiel Tic-Tac-Toe:
  - `(true (control xplayer))`
  - xplayer aktiv im aktuellen Zustand
  - `(true (cell 1 1 b))` … `(true (cell 3 1 b))`
  - `(true (cell 3 2 x))`
  - `(true (cell 3 3 o))`
  - nur zwei Zellen belegt
    - (3, 2) mit x
    - (3, 3) mit o
- Darf nicht Head einer Regel sein

Relationen - init

- Analog zu true, nur im Initialzustand
- Beispiel Tic-Tac-Toe
  - `(init (control xplayer))`
  - xplayer beginnt
  - `(init (cell 1 1 b)) … (init (cell 3 3 b))`
  - Spielbrett leer (b = blank)
- Darf nur Head einer Regel sein
- Darf nicht von true, does, next, legal, goal oder terminal abhängen

Relationen - does

- Wird entsprechend gewählter Züge gesetzt
  - genau ein Zug für jeden Spieler
- Nötig, um Nachfolgezustand zu berechnen
  - Zur Erinnerung: Nur Züge übertragen, nicht gesamte Zustände
- Beispiel Tic-Tac-Toe:
  - `(does xplayer (mark 1 2))`
  - `(does oplayer noop)`
  - xplayer markiert Zelle (1, 2) während oplayer nichts macht
- Darf nicht Head einer Regel sein
- Darf nicht von legal, goal oder terminal abhängen (und umgekehrt)

Relationen - next

- Nötig zur Berechnung des Nachfolgezustands (Update)
  - hängt häufig von gewählten Zügen ab
- Fluents, für die next wahr ist sind im nächsten Zustand in der true Relation wahr
- Beispiel Tic-Tac-Toe
  -
```
(<= (next (control xplayer))
    (true (control oplayer)))
```
  - Wechsel des aktiven Spielers
  -
```
(<= (next (cell ?x ?y x))
    (true (cell ?x ?y b))
    (does xplayer (mark ?x ?y)))
```
  - Setzen von x an Position (?x, ?y)
- Darf nur Head von Regeln sein

Relationen - next

- Wichtig: Frame explizit modelliert
  - Fluents, für die next nicht wahr ist, gelten auch nicht im Nachfolgezustand
  - daher nötig, Unverändertes explizit zu erwähnen
- Beispiel Tic-Tac-Toe:
  - ```
    (<= (next (cell ?x ?y b))
        (true (cell ?x ?y b))
        (does ?player (mark ?m ?n))
        (or (distinct ?x ?m) (distinct ?y ?n)))
    ```
  - Zelle (?x, ?y) bleibt leer, wenn
    - eine andere Zelle (?m ?n) markiert wird und
    - ?x ≠ ?m oder ?y ≠ ?n gelten
    ```

Relationen - legal

- Bestimmung gültiger Züge
  - mindestens ein Zug für jeden Spieler in jedem (nicht-terminalen) Zustand
- Beispiel Tic-Tac-Toe:
  -
```
(<= (legal ?player (mark ?x ?y))
    (true (cell ?x ?y b))
    (true (control ?player)))
```
  - Spieler ?player kann Zelle (?x, ?y) nur markieren wenn
    - Zelle leer und
    - ?player am Zug

Relationen - legal

- Zur Erinnerung:
  - alle Spiele simultan
  - abwechselnde Züge explizit modelliert
- Beispiel Tic-Tac-Toe
  - `(<= (legal xplayer noop)`
    `(true (control oplayer))`
  - xplayer kann nichts machen wenn oplayer am Zug ist
  - noop könnte auch beliebig anders heißen

Relationen - terminal

- Definiert Terminalzustände
- Beispiel Tic-Tac-Toe
  - `(<= terminal`
    `(role ?player)`
    `(line ?player))`
  - Spieler ?player hat eine Linie von 3 seiner Symbole erreicht
  - `(<= terminal`
    `(not open))`
  - es gibt keine freie Zelle mehr

Relationen - goal

- Definiert Zielzustände für jeden Spieler
  - ordnet diesen Zuständen Bewertungen zu
  - Bewertungen ganzzahlig von 0 (schlecht) bis 100 (top)
  - jeder Terminalzustand muss eindeutige Bewertung für jeden Spieler haben
- Beispiel Tic-Tac-Toe:
  -
```
(<= (goal ?player 100)
    (line ?player))
```
  - bestes Ziel für jeden Spieler: Linie von 3 eigenen Symbolen

Relationen - goal

- Bewertungen können binär sein
  - nur 0 oder 100
- Bewertungen können invers sein
  - Nullsummenspiele: (0, 100), (100, 0), evtl. (50, 50)
- Bewertungen können identisch für mehrere Spieler sein
  - Kooperative Spiele
- Bewertungen können beliebig anders verteilt sein

Tic-Tac-Toe - Rollen

```
(role xplayer)
(role oplayer)
```

Tic-Tac-Toe - Initialzustand

```
(init (cell 1 1 b))
(init (cell 1 2 b))
(init (cell 1 3 b))
(init (cell 2 1 b))
(init (cell 2 2 b))
(init (cell 2 3 b))
(init (cell 3 1 b))
(init (cell 3 2 b))
(init (cell 3 3 b))
(init (control xplayer))
```

Tic-Tac-Toe - gültige Züge

```
(<= (legal ?player (mark ?x ?y))
    (true (cell ?x ?y b))
    (true (control ?player))
(<= (legal xplayer noop)
    (true (control oplayer)))
(<= (legal oplayer noop)
    (true (control xplayer)))
```

Tic-Tac-Toe - Update

```
(<= (next (cell ?x ?y x))
    (does xplayer (mark ?x ?y)))
(<= (next (cell ?x ?y o))
    (does oplayer (mark ?x ?y)))
(<= (next (cell ?x ?y ?mark))
    (true (cell ?x ?y ?mark))
    (distinct ?mark b))
(<= (next (cell ?x ?y b))
    (true (cell ?x ?y b))
    (does ?player (mark ?m ?n))
    (or (distinct ?x ?m) (distinct ?y ?n)))
```

Tic-Tac-Toe - Update

```
(<= (next (control xplayer))
    (true (control oplayer)))
(<= (next (control oplayer))
    (true (control xplayer)))
```

Tic-Tac-Toe - Terminierung und eigene Relationen

```
(<= terminal (or (line x) (line o)))
(<= terminal (not open))

(<= (line ?mark) (row ?row ?mark))
(<= (line ?mark) (column ?column ?mark))
(<= (line ?mark) (diagonal ?mark))

(<= open (true (cell ?x ?y b)))
```

Tic-Tac-Toe - eigene Relationen

```
(<= (row ?row ?mark)
    (true (cell ?row 1 ?mark))
    (true (cell ?row 2 ?mark))
    (true (cell ?row 3 ?mark)))

(<= (column ?column ?mark)
    (true (cell 1 ?column ?mark))
    (true (cell 2 ?column ?mark))
    (true (cell 3 ?column ?mark)))
```

Tic-Tac-Toe - eigene Relationen

```
(<= (diagonal ?mark)
    (true (cell 1 1 ?mark))
    (true (cell 2 2 ?mark))
    (true (cell 3 3 ?mark)))
(<= (diagonal ?mark)
    (true (cell 1 3 ?mark))
    (true (cell 2 2 ?mark))
    (true (cell 3 1 ?mark)))
```

Tic-Tac-Toe - Bewertungen

```
(<= (goal xplayer 100)
    (line x))
(<= (goal xplayer 50)
    (not (line x))
    (not (line o))
    (not open))
(<= (goal xplayer 0)
    (line o))
```

Tic-Tac-Toe - Bewertungen

```
(<= (goal oplayer 100)
    (line o))
(<= (goal oplayer 50)
    (not (line x))
    (not (line o))
    (not open))
(<= (goal oplayer 0)
    (line x))
```

Endliche Ableitbarkeit (Sicherheit)

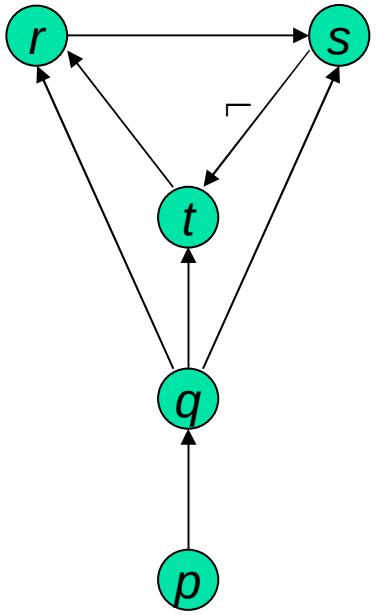
- Eine Klausel ist sicher, gdw. jede Variable der Klausel in positivem Literal im Body auftaucht
- Sichere Regel:
  - `(<= (r ?x ?y) (p ?x ?y) (not (q ?x ?y)))`
- Unsichere Regel:
  - `(<= (r ?x ?y) (p ?x ?y) (not (q ?x ?z)))`
- In GDL müssen alle Regeln sicher sein
  - damit: alle Fakten (Regeln ohne Body) variablenfrei

Abhängigkeitsgraph

- Abhängigkeitsgraph für Menge von Klauseln ist gerichteter Graph mit
  - Knoten sind Relationen in Head und Body der Klausel
  - Kante von Knoten *p* nach *q*, wenn *p* in Body einer Klausel mit Head *q*
- Menge von Klauseln rekursiv, wenn Abhängigkeitsgraph Kreis enthält
  - sonst nicht-rekursiv

Abhängigkeitsgraph

```
(<= (q ?x)
    (p ?x))
(<= (r ?x ?y)
    (q ?x) (t ?x ?y))
(<= (s ?x ?y)
    (r ?x ?y) (q ?y))
(<= (t ?x ?y)
    (q ?x)
    (q ?y)
    (not (s ?x ?y)))
```



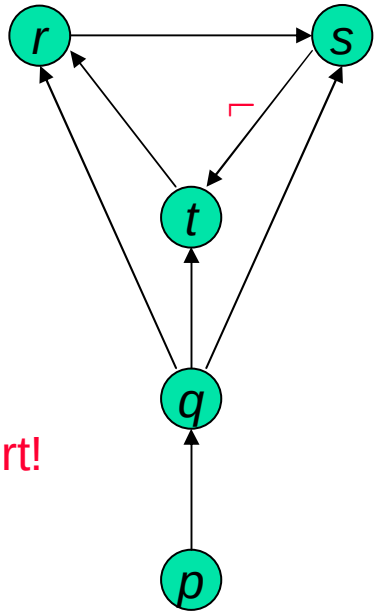
Endliche Ableitbarkeit (Stratifizierte Negation)

- Menge von Regeln stratifiziert, wenn kein rekursiver Kreis mit Negation in Abhängigkeitsgraph
- In GDL alle Regeln stratifiziert

Abhängigkeitsgraph

```
(<= (q ?x)
    (p ?x))
(<= (r ?x ?y)
    (q ?x) (t ?x ?y))
(<= (s ?x ?y)
    (r ?x ?y) (q ?y))
(<= (t ?x ?y)
    (q ?x)
    (q ?y)
    (not (s ?x ?y)))
```

nicht stratifiziert!



Endliche Ableitbarkeit

- Wenn Menge von Regeln Klausel

$$
\begin{array}{l}
(\texttt{<=}\ (p\ s_1\ \ldots\ s_m) \\
\quad (b_1\ t_{11}\ \ldots\ t_{1l}) \\
\quad \vdots \\
\quad (q\ v_1\ \ldots\ v_k) \\
\quad \vdots \\
\quad (b_n\ t_{n1}\ \ldots\ t_{nl}))
\end{array}
$$

  enthält, so dass p und q in Kreis des Abhängigkeitsgraphen auftreten, muss für jedes $i \in \{1, \ldots, k\}$ gelten:

  - v_i variablenfrei oder
  - v_i eines von $s_1, \ldots, s_m$ oder
  - v_i tritt in einem $t_{j1}, \ldots, t_{jl}$ auf, so dass b_j nicht auf Kreis mit p im Abhängigkeitsgraph liegt ($1 \leq j \leq n$)

Vollständigkeit

- Spielbeschreibungen sind logisch unvollständig
  - Züge der Spieler nicht eindeutig spezifiziert
- Jede Spielbeschreibung enthält vollständige Definitionen für
  - Gültigkeit von Zügen
  - Terminierung
  - Bewertung
  - Update

  durch Relationen true und does
- Jeder Spieler kann
  - gültige Züge berechnen
  - prüfen, welche Züge gültig
  - prüfen, ob Spiel beendet
  - gegeben Züge aller Spieler, Nachfolgezustand berechnen

Definition: Terminierung

- Eine Spielbeschreibung in GDL terminiert, wenn alle unendlichen Sequenzen gültiger Züge vom Initialzustand nach endlich vielen Schritten einen Terminalzustand erreichen
- Jedes Spiel muss terminieren
  - unendliche Spielverläufe nicht vorgesehen
  - oft Step-Counter, damit Spiel endlich

Definition: Spielbarkeit

- Eine Spielbeschreibung in GDL ist spielbar, gdw. jeder Spieler in jedem nicht-terminalen Zustand, der vom Initialzustand aus erreichbar ist, mindestens einen gültigen Zug hat
- Beispiel Schach:
  - Wenn ein Spieler nicht mehr ziehen kann, ist eine Patt-Situation erreicht
  - Patt ist aber ein Terminalzustand (Remis)
- Jedes Spiel muss spielbar sein

Definition: Monotonie

- Eine Spielbeschreibung in GDL ist monton, gdw. jeder Spieler genau eine Bewertung in jedem vom Initialzustand aus erreichbaren Zustand hat und die Bewertungen niemals geringer werden

Definition: Gewinnbarkeit

- Eine Spielbeschreibung in GDL ist stark gewinnbar, gdw. es für irgendeinen Spieler eine Sequenz individueller Züge gibt die zu einem Terminalzustand führt in dem die Bewertung für diesen Spieler maximal (= 100) ist.
- Eine Spielbeschreibung in GDL ist schwach gewinnbar, gdw. es für jeden Spieler eine Sequenz gemeinsamer Züge gibt die zu einem Terminalzustand führt in dem die Bewertung für diesen Spieler maximal ist

Definition: Wohlgeformtes Spiel

- Eine Spielbeschreibung in GDL ist wohlgeformt (engl. well-formed), wenn es terminiert und monoton, spielbar sowie schwach gewinnbar ist
- Damit müssen wohlgeformte Einpersonenspiele sogar stark gewinnbar sein
- In Meisterschaften sollten Spiele stets wohlgeformt sein
  - Monotonie dürfte aber häufig nicht gegeben sein
  - Bewertungen zumeist nur in Terminalzuständen