Handlungsplanung und **Allgemeines Spiel**

„Evaluationsfunktionen “

Peter Kissmann

Themen Allgemeines Spiel

- Einführung
- Game Desciption Language (GDL)
- Spielen allgemeiner Spiele
- Evaluationsfunktionen im allgemeinen Spiel
- Verbesserungen für Alpha-Beta und UCT
- Lösen allgemeiner Spiele
- Instanziierung
- Ausblick: Unvollständige Information und Zufall

Problem Minimax-basierter Verfahren

- gut (optimal), wenn vollständige Suche möglich
- Aber: im allgemeinen Spiel in der Regel nicht möglich
- daher: Verwendung von Evaluationsfunktionen

Evaluationsfunktionen

- für klassische Spiele, oft vom Programmierer festgelegt / basierend auf Expertenwissen
  - etwa Bewertung von Figuren beim Schach
  - etwa Vorteil von Ecken in Reversi
- im allgemeinen Spiel nicht möglich
  - Evaluationsfunktionen müssen automatisch generiert werden

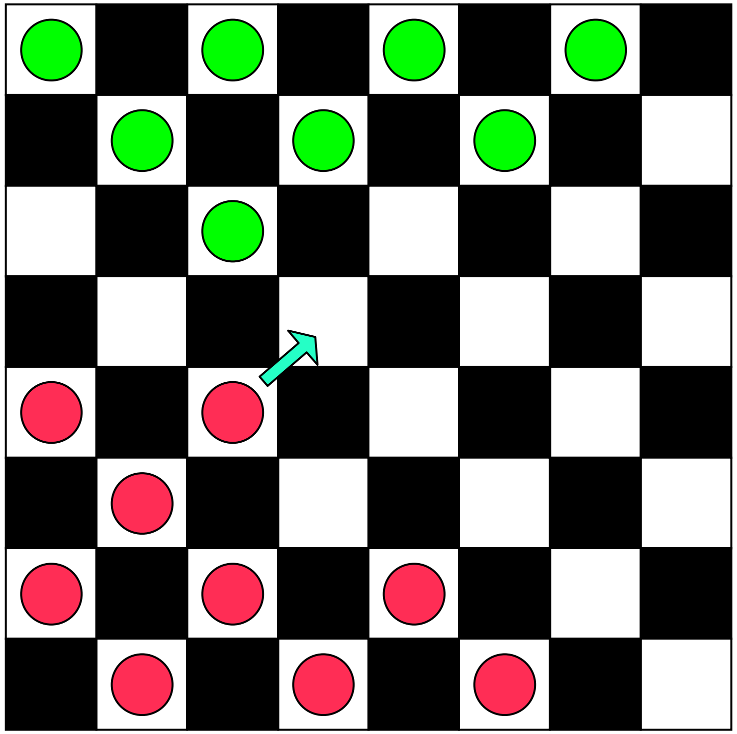
Evaluationsfunktionen

- einige allgemeine Ansätze funktionieren häufig, aber nicht immer
- etwa
  - Mobilität
  - Neuigkeit
  - Entfernung zum Ziel

Evaluationsfunktionen - Mobilität

- mehr Züge = besserer Zustand
- evtl. auch Einschränkung von Gegnerzügen gut
- oft: Zugzwang schlecht
  - Schach: Schachposition auflösen
  - Reversi: wenige Züge = geringe Kontrolle über Spielfeld
- aber:
  - schlecht in Dame (mit Zugzwang)

Evaluationsfunktionen - Mobilität



- Ergebnis: Gegner hat nur einen Zug
- Aber: eigener Spielstein geopfert

Evaluationsfunktionen - Inverse Mobilität

- Weniger Möglichkeiten zu haben, ist besser
- Auch: Gegner mehr zu tun zu geben ist besser
- Beispiel: nothello (wie Reversi (Othello), aber gewonnen, wenn weniger Steine als Gegner)
- Wie automatisch entscheiden, ob Mobilität oder inverse Mobilität?

Evaluationsfunktionen - Neuigkeit

- Änderung des Zustands vorteilhaft
- Vorteile:
  - große Änderungen verhindern, festzustecken
  - wenn man nicht weiß, was man machen soll, vielleicht gut, etwas “gerichteten Zufall” einzuführen
- Nachteile:
  - Zustandsänderung, wenn eigene Figuren geopfert
  - Unklar, ob Neuigkeit generell für irgendwen gut

Evaluationsfunktion - Zieldistanz

- Je näher dran, eigenes Ziel (goal) zu erfüllen, desto besser
- Beispiel: Tic-Tac-Toe
  -
```
(<= (goal xplayer 100)
    (or (and (true (cell ?c 1 x)) (true (cell ?c 2 x))
             (true (cell ?c 3 x)))
        (and (true (cell 1 ?r x)) (true (cell 2 ?r x))
             (true (cell 3 ?r x)))
        (and (true (cell 1 1 x)) (true (cell 2 2 x))
             (true (cell 3 3 x)))
        (and (true (cell 1 3 x)) (true (cell 2 2 x))
             (true (cell 3 1 x)))))
```

Evaluationsfunktionen - Zieldistanz

eval(`(goal xplayer 100)`) nach `(does xplayer (mark 2 2))`
\> eval(`(goal xplayer 100)`) nach `(does xplayer (mark 1 1))`
\> eval(`(goal xplayer 100)`) nach `(does xplayer (mark 1 2))`

Vorlesungs-Ablauf

- Evaluationfunktionen für Alpha-Beta
  - nach [Kuhlmann et al., 2006]
  - nach [Schiffel & Thielscher, 2007]
- Weitere nützliche Eigenschaften durch Simulationen

Evaluationsfunktion [Kuhlmann et al., 2006]

- Strukturen identifizieren
- von Strukturen zu Features
- von Features zu Evaluationsfunktionen
- Verteilte Suche

Strukturen

- wichtige Strukturen:
  - Zähler
  - Spielbretter
  - bewegliche Figuren
  - …
- Finden durch
  - syntaktischen Vergleich
  - Simulation

Strukturen - Nachfolgerrelation

- Syntax:
  - `(<succ> <el1> <el2>)`
    `(<succ> <el2> <el3>)`
    `(<succ> <el3> <el4>)`
    etc.
- Spielbeschreibung kann mehrere Nachfolgerelationen enthalten:
  - Zähler (1, 2, 3, …)
  - benachbarte x-Koordinaten
  - benachbarte y-Koordinaten
  - etc.

Strukturen - Zähler

- Fluent, das bei jedem Schritt inkrementiert wird
- Syntax:
  - ```
    (<= (next (<counter> ?<var2>))
        (true (<counter> ?<var1>))
        (true (<succ> ?<var1> ?<var2>)))
    ```
    ```

Strukturen - Zähler

- Nutzen:
  - oft als step-counter, um Spiel endlich zu halten:
    - Spiel terminiert nach x Schritten
  - kann bei Simulation entfernt werden, damit
    - oft weniger Zustände (mehr Duplikate)
    - höhere Chance, guten Zielzustand zu finden

Strukturen - Spielbrett

- Syntax:
  - 3-wertiges Fluent
  - zwei Parameter für Koordinaten
  - ein Parameter für Belegung
- Annahme:
  - jedes 3-wertige Fluent beschreibt Spielfeld
- Spielfeldposition kann nicht doppelt belegt sein
  - Koordinaten: Input-Parameter
  - Belegung: Output-Parameter
- Spielbrett kann geordnet sein
  - wenn Koordinaten über Nachfolgerelation geordnet

Strukturen - Marker und Figuren

- Belegung von Spielbrett ist Marker
- wenn Marker stets an nur einer Position → Figur

Strukturen finden durch Simulation

- Spielbretter:
  - 3-wertig, 2 Input-, 1 Output-Parameter
  - simuliere einige Schritte
  - prüfe, ob für (angenommene) Input-Parameter auf (angenommenem) Output-Parameter stets nur eine Belegung erfüllt
    - wenn nicht, entsprechendes keine Input-Parameter
    - falls keine Kombination Input-Parameter, Fluent kein Spielbrett
- Marker / Figuren:
  - Annahme: Jede Belegung von Output-Parameter von Spielbrett ist Figur
  - prüfe (bei Simulation), ob (angenommene) Figur nur auf einem Feld
    - wenn nicht, ist es Marker

Von Strukturen zu Features

- Feature:
  - numerischer Wert
  - berechnet aus Spielzustand
  - potenziell mit Bewertung in Terminalzustand korreliert
  - Beispiel: geordnetes Spielbrett
    - berechne x- und y-Koordinaten aller Figuren
    - entsprechen natürlichen Zahlen gemäß der Nachfolgerrelation
    - damit möglich, Manhattan-Distanz zwischen Figuren zu berechnen
  - Beispiel: ungeordnetes Spielbrett
    - zähle Anzahl an Markern

Von Strukturen zu Features

Identifizierte Strukturen	Generierte Features
Geordnetes Spielbrett mit Figuren	x-Koordinaten jeder Figur
	y-Koordinaten jeder Figur
	Manhattan-Distanz zwischen jedem Figurenpaar
	Summe der paarweisen Manhattan-Distanzen
Spielbrett ohne Figuren	Anzahl Marker jeden Typs
Mengenangabe	entsprechender Wert

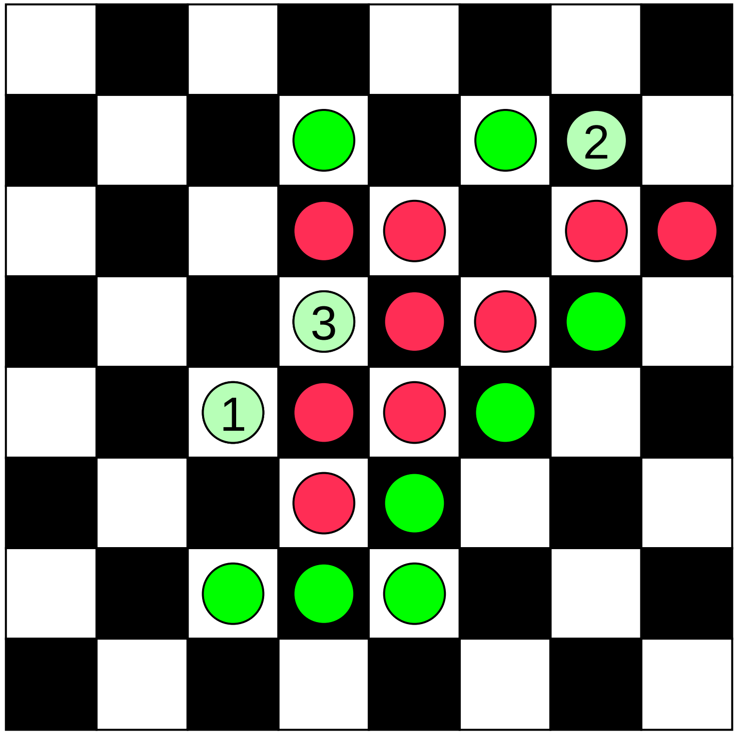
Von Features zu Evaluationsfunktionen

- in spezialisierten Spielern:
  - Evaluationsfunktion als gewichtete Kombination von Features
    - Gewichte manuell festgelegt
    - daher schwierig im allgemeinen Spiel
- hier:
  - erzeuge Menge potenzieller Evaluationsfunktionen
    - jede als Maximierung oder Minimierung einzelnen Features
    - Max. und Min, damit Möglichkeit, bei Suizid-Spielen gut zu spielen

Von Features zu Evaluationsfunktionen

- $V(s)$: skaliertes Feature in [0, 1] in Zustand s
- Evaluationsfunktion:
  - Maximierende Funktion:
    - $E(s) = 1 + R^- + (R^+ - R^- - 2) * V(s)$
  - Minimierende Funktion:
    - $E(s) = 1 + R^- + (R^+ - R^- - 2) * (1 - V(s))$
  - mit:
    - R^-: minimaler erreichbarer Gewinn
    - R^+: maximaler erreichbarer Gewinn
  - Damit: $E: S \rightarrow [R^- + 1, R^+ - 1]$
  - echter Gewinn besser als jede Evaluation, echter Verlust schlechter als jede Evaluation

Beispiel: Reversi (Othello)



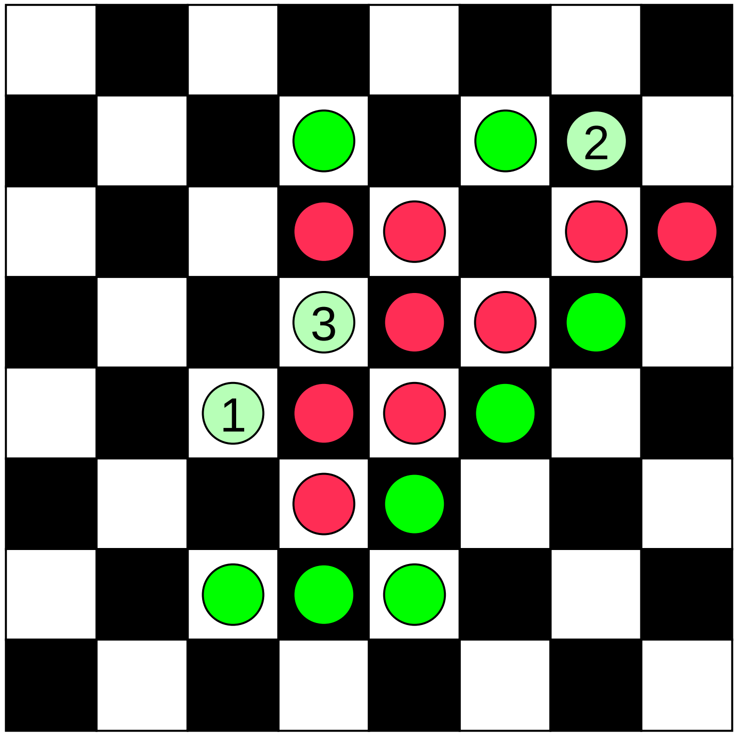
Aktueller Zustand:
#Marker(grün): 8
V(s) = #Marker(grün) / 64 = 0,125
E(s) = 1 + 98 * V(s) = 13,25

Fall 1:
#Marker(grün): 12
V(s) = #Marker(grün) / 64 = 0,1875
E(s) = 1 + 98 * V(s) = 19,375

Fall 2:
#Marker(grün): 10
V(s) = #Marker(grün) / 64 = 0,15625
E(s) = 1 + 98 * V(s) = 16,3125

Fall 3:
#Marker(grün): 14
V(s) = #Marker(grün) / 64 = 0,21875
E(s) = 1 + 98 * V(s) = 22,4375

Beispiel: Suizid-Reversi (nOthello)



Aktueller Zustand:
#Marker(grün): 8
V(s) = #Marker(grün) / 64 = 0,125
E(s) = 1 + 98 * (1 - V(s)) = 86,75

Fall 1:
#Marker(grün): 12
V(s) = #Marker(grün) / 64 = 0,1875
E(s) = 1 + 98 * (1 - V(s)) = 80,625

Fall 2:
#Marker(grün): 10
V(s) = #Marker(grün) / 64 = 0,15625
E(s) = 1 + 98 * (1 - V(s)) = 83,6875

Fall 3:
#Marker(grün): 14
V(s) = #Marker(grün) / 64 = 0,21875
E(s) = 1 + 98 * (1 - V(s)) = 77,5625

Verteilte Suche

- Problem:
  - mehrere Evaluationsfunktionen
  - einige evtl. gut geeignet, andere nicht
  - wie entscheiden, welche zu verwenden
  - Entscheidung kann sich auch im Laufe des Spiels ändern
- Mögliche Lösung:
  - Hinweise aus Zielbeschreibung generieren
- Aber:
  - kann beliebig komplex werden

Verteilte Suche

- Lösung:
  - ein Master-Prozess
  - Menge von Slave-Prozessen
  - Master
    - informiert Slaves über Zustandsänderungen
    - weist jedem Prozess eine Evaluationsfunktion zu
  - Slave
    - schickt besten bisher gefundenen Zug
  - Master
    - wählt "besten" von allen Prozessen gefundenen Zug
    - schickt diesen an GameController

Verteilte Suche

- mehr Prozesse als Evaluationsfunktionen → manche Evaluationsfunktionen auf mehreren Slaves
  - mag zu Expansion gleicher Bereiche führen
  - aber: gleichbewertete Züge werden zufällig ausgewählt → unterschiedliche Slaves sollten oft unterschiedliche Züge wählen
- zusätzlich: Slave(s), die vollständige Suche durchführen

Verteilte Suche

- Schwierigkeit: Welches ist “bester” Zug?
  - wenn vollständige Suche Lösung gefunden hat, diese verfolgen
  - sonst:
    - Züge, die aus tieferen Suchen resultierten, präferiert
  - Alternative:
    - Zielbedingung mit in Entscheidung einfließen lassen (aber wie?)

Evaluationsfunktionen [Schiffel & Thielscher, 2007]

- Evaluationsfunktion auf Basis von Terminal- und Zielbeschreibung
  - Berechnung des Erfülltheitsgrades
  - Werte von Terminal- und Zielauswertungen kombiniert, damit Terminalzustand vermieden, bis Ziel erfüllt
  - Terminalauswertung hat
    - negativen Einfluss, falls Zielauswertung niedrigen Wert liefert
    - positiven Einfluss, sonst

Evaluationsfunktionen

- Idee:
  - nutze Fuzzy-Logik
  - Weise Fluents Werte zwischen 0 und 1 zu, abhängig von Wahrheitswert
  - Nutze Standard T-Norm und T-Conorm, um Wahrheitsgrad komplexer Formeln zu bestimmen

Einschub: Fuzzy-Logik

- keine Booleschen Werte {0, 1}, sondern mehrwertig [0, 1]
- wie komplexere Formeln verknüpfen, etwa $a \wedge b$ für $a, b \in [0, 1]$?
- Lösung: verallgemeinerter Konjunktions-Operator → T-Norm
- Eigenschaften T-Norm:
  - $T: [0, 1] \times [0, 1] \rightarrow [0, 1]$
  - Assoziativität: $T(a, T(b, c)) = T(T(a, b), c)$
  - Kommutativität: $T(a, b) = T(b, a)$
  - Monotonie: $T(a, c) \leq T(b, c)$, falls $a \leq b$
  - Neutrales Element 1: $T(a, 1) = a$
  - Nullelement 0: $T(a, 0) = 0$

Einschub: Fuzzy-Logik

- Gegenstück: verallgemeinerter Disjunktions-Operator → T-Conorm (auch S-Norm)
- Dual zu T-Norm:
  - 1 - S(a, b) = T(1 - a, 1 - b)
  - 1 - T(a, b) = S(1 - a, 1 - b)
- Eigenschaften der T-Conorm:
  - S: [0, 1] x [0, 1] → [0, 1]
  - Assoziativität: S(a, S(b, c)) = S(S(a, b), c)
  - Kommutativität: S(a, b) = S(b, a)
  - Monotonie: S(a, c) ≤ S(b, c), falls a ≤ b
  - Neutrales Element 0: S(a, 0) = a
  - Nullelement 1: S(a, 1) = 1

Einschub: Fuzzy-Logik

Häufige T-Normen und T-Conormen			
$T_{min}(a, b)$	$\min\{a, b\}$	$S_{max}(a, b)$	$\max\{a, b\}$
$T_{Lukasiewicz}(a, b)$	$\max\{0, a + b - 1\}$	$S_{Lukasiewicz}(a, b)$	$\min\{a + b, 1\}$
$T_{prod}(a, b)$	$a \cdot b$	$S_{sum}(a, b)$	$a + b - a \cdot b$
$T_{-1}(a, b)$	a, falls b = 1 b, falls a = 1 0, sonst	$S_{-1}(a, b)$	a, falls b = 0 b, falls a = 0 1, sonst

Evaluationsfunktionen

- Problem Standard T-Norm:
  - Resultat ≤ kleineres der Elemente
  - damit Resultat = 0, falls mind. ein Element nicht erfüllt
- Beispiel (Blocksworld):
  - Zielformel: `(and (on a b) (on b c) (ontable c))`
  - Evaluation soll Anzahl erfüllter Teilziele widerspiegeln
  - wenn nur `(on a b)` fehlt, wird Formel zu 0 ausgewertet, obwohl Ziel fast erreicht

Evaluationsfunktionen

- Problem: kann nicht zwischen Zuständen unterscheiden, die beide bestimmte Formel erfüllen
- Beispiel: Reversi
- Ziel: Mehr Steine als Gegner
  - gute Heuristik: Je mehr eigene Steine, desto besser
  - aber: `(greater ?whitepieces ?blackpieces)` immer 1, egal ob weiß 1 oder 20 Steine mehr als schwarz

Evaluationsfunktionen

- Behebung:
  - erfüllte Fluents: Wert p
  - nicht erfüllte Fluents: Wert $1 - p$
  - $0{,}5 < p < 1$

Evaluationsfunktionen

- Neues Problem bei T_{prod}
  - Konjunktion mit vielen Elementen gegen 0, auch wenn alle Elemente wahr
  - Disjunktion mit vielen Elementen gegen 1, auch wenn alle Elemente falsch
- Behebung:
  - Verwendung von Threshold t (0,5 < t < 1)
  - Intention: Werte > t entspr. wahr, Werte < 1 - t entspr. falsch
  - verwendete T-Norm T':

$$T'(a,b) = \begin{cases} \max(T(a,b), t) & \text{, falls } \min(a,b) > 0{,}5 \\ T(a,b) & \text{, sonst} \end{cases}$$

    mit T beliebige Standard T-Norm
  - entsprechende T-Conorm: S'(a, b) = 1 - T'(1 - a, 1 - b)

Evaluationsfunktionen

- damit sichergestellt:
  - erfüllte Formeln haben Wert ≥ t
  - nicht-erfüllte Formeln haben Wert ≤ 1 - t
  - also Werte unterschiedlicher Formeln vergleichbar
- Aber:
  - T' nicht assoziativ
  - damit keine echte T-Norm im ursprünglichen Sinne
  - also für semantisch identische aber syntaktisch unterschiedliche Formeln evtl. unterschiedliche Werte
  - Effekt minimal für geeignete T-Norm
  - gewählte T-Norm T:
    - $T(a, b) = 1 - S(1 - a, 1 - b)$
    - $S(a, b) = (a^q + b^q)^{1/q}$
  - sinnvoll: q klein bei vielen Disjunktionen, q groß bei vielen Konjunktionen (damit nicht gegen 1 / gegen 0 für viele Zustände)

Evaluationsfunktionen

- eval(f, z): Evaluation von Zustand z bzgl. Formel f
  - (a Fluent, f und g beliebige Formeln)
- eval(a, z) = p, falls a wahr in aktuellem Zustand; 1 - p sonst
- $eval(f \wedge g, z) = T'(eval(f, z), eval(g, z))$
- $eval(f \vee g, z) = S'(eval(f, z), eval(g, z))$
- $eval(\neg f, z) = 1 - eval(f, z)$
- eval hat folgende Eigenschaften:
  - $\forall f, z.\ eval(f, z) \geq t > 0{,}5$ gdw. f erfüllt in z
  - $\forall f, z.\ eval(f, z) \leq 1 - t < 0{,}5$ gdw. f nicht erfüllt in z

Evaluationsfunktionen

- vollständige Evaluationsfunktion für Zustand z:

$$h(z) = \frac{1}{\sum_{gv \in GV} gv} \cdot \bigoplus_{gv \in GV} h(gv, z) \cdot \frac{gv}{100}$$

$$h(gv, z) = \begin{cases} \text{eval}(\text{goal}(gv) \vee \text{terminal}, z) & \text{, falls goal}(gv) \\ \text{eval}(\text{goal}(gv) \wedge \neg\text{terminal}, z) & \text{, falls} \neg\text{goal}(gv) \end{cases}$$

versucht, Terminalzustand zu erreichen, wenn goal erfüllt

versucht, Terminalzustand zu vermeiden, wenn goal nicht erfüllt

- dabei:
  - GV: Menge aller möglichen goal values (Werte der Zielbedingungen)
  - $\oplus$: Summe von (Produkt-)T-Conormen
  - h(gv,z): Evaluation für einen goal value
  - gv: goal value
  - terminal: (ausgerollte) Terminierungsbedingung
  - goal(gv): (ausgerollte) Zielbedingung mit Wert gv

Strukturen - Statische Strukturen

- Statische Strukturen:
  - Strukturen, unabhängig von aktuellem Zustand
    - in Regeln taucht kein `(true …)` auf
  - etwa Nachfolgerrelation, Ordnungsrelation

Strukturen - Statische Strukturen

- Nachfolgerrelation:
  - binäre Relation
  - antisymmetrisch
  - funktional
  - injektiv
  - azyklische Graphrepräsentation
- Beispiele:
  - `(succ 1 2) (succ 2 3) (succ 3 4) …`
  - `(nextrow a b) (nextrow b c) (nextrow c d) …`

Strukturen - Statische Strukturen

- Ordnungsrelation:
  - binäre Relation
  - antisymmetrisch
  - transitiv
- Beispiel:
  - ```
    (<= (lessthan ?a ?b)
        (succ ?a ?b))
    (<= (lessthan ?a ?c)
        (succ ?a ?b)
        (lessthan ?b ?c))
    ```
    ```

Strukturen - Statische Strukturen

- Eigenschaften lassen sich recht einfach prüfen
  - alle Definitionsbereiche endlich
- hier: keine Annahme über Syntax, nur Semantik
- Vorteile:
  - syntaktische Beschreibung kann unterschiedlich sein
  - komplexere Relationen (z.B. Ordnungsrelation) auffindbar

Strukturen - Dynamische Strukturen

- Dynamische Strukturen
  - abhängig von aktuellem Zustand
  - können sich im Spielverlauf ändern
- Beispiel:
  - Spiebrettpositionen
- hier: finden von Spielbrettern wie vorher, aber:
  - Spielbretter nicht zwingend 3-dimensional (2 Input-, 1 Output-Parameter)
  - jedes Fluent mit ≥ 2 Parametern potenzielles Spielbrett

Strukturen - Dynamische Strukturen

- “Mengenfluent”
  - Einwertig
  - Geordnet
  - Singleton (keine Input-Parameter) → nur einmal in jedem Zustand
- Beispiel:
  - Step-Counter
- Auch Spielbrettbelegungen können Mengen sein
  - wenn Output-Parameter geordnet, also durch Nachfolge- oder Ordnungsrelation verknüpft

Strukturen - Dynamische Strukturen

- finden durch Simulation
- n-wertiges Fluent
- Hypothesen:
  - 1. Parameter ist Input
  - 2. Parameter ist Input
  - …
  - n. Parameter ist Input
  - 1. + 2. Parameter sind Input
  - …
  - alle n Parameter sind Input

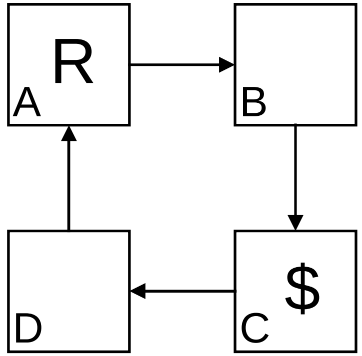
Strukturen - Dynamische Strukturen

- Überprüfung der Hypothesen durch Simulation
  - gleiche Input-Belegung aber unterschiedliche Output-Belegung:
    - verwerfe Hypothese
  - Problem:
    - exponentiell (in Parameteranzahl) viele Hypothesen
  - Aber:
    - oft geringe Parameteranzahl:
      - 2 Parameter → 3 Hypothesen
      - 3 Parameter → 7 Hypothesen
      - 4 Parameter → 15 Hypothesen
    - häufig: nur wenige Hypothesen übrig nach Initialzustand

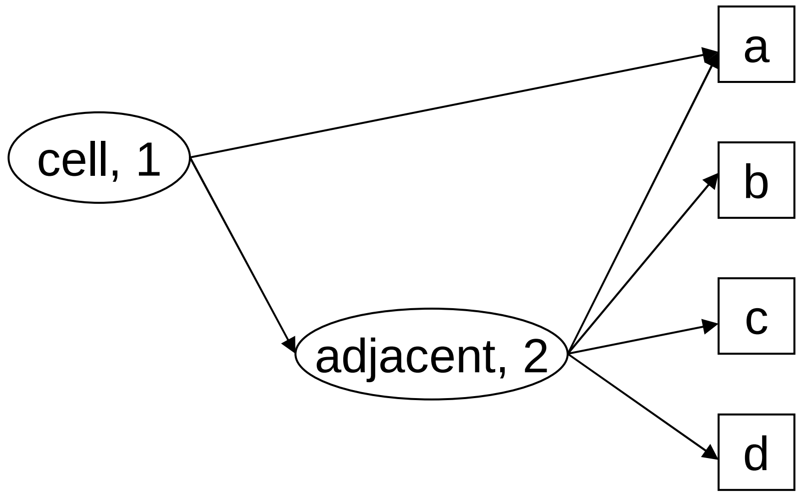
Strukturen - Abhängigkeitsgraph

- für Mengen wichtig:
  - (Output-)Parameter geordnet
  - Definitionsbereich der Argumente bekannt
- Obermenge der Definitionsbereiche durch Abhängigkeitsgraph
  - Parameter in Head abhängig von Parameter in Body gdw. identische Variable
  - dann kann Parameter in Head potenziell identische Werte wie in Body annehmen

Strukturen - Abhängigkeitsgraph



- `(init (cell a))`
- ```
  (<= (next (cell ?y))
      (does robot move)
      (true (cell ?x))
      (adjacent ?x ?y))
  ```
- `(adjacent a b)`
- `(adjacent b c)`
- `(adjacent c d)`
- `(adjacent d a)`



- Definitionsbereich von `cell:`
- `(cell a)`
- `(cell b)`
- `(cell c)`
- `(cell d)`
  ```

Von Strukturen zur Evaluationsfunktion

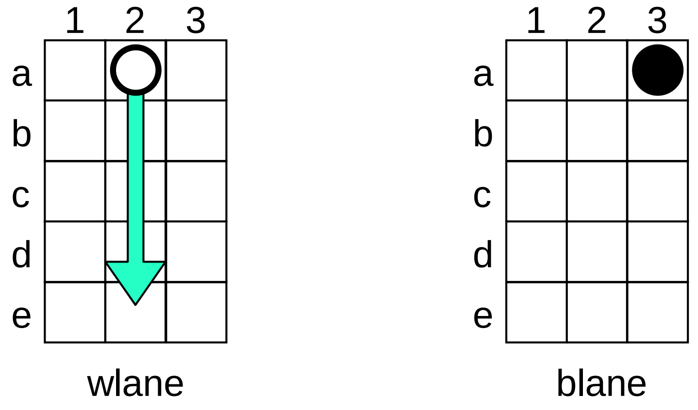
- Verbesserung der Evaluationsfunktion durch Nutzung nicht-binärer Auswertungen gemäß Strukturen
- Für Ordnungsrelation r:

$$\mathrm{eval}(\mathrm{r}(a,b),z)=\begin{cases} t+(1-t)\cdot\dfrac{\Delta(a,b)}{|\mathrm{dom}(r)|} & \text{, falls } \mathrm{r}(a,b) \\ (1-t)-(1-t)\cdot\dfrac{\Delta(b,a)}{|\mathrm{dom}(\mathrm{r})|} & \text{, falls } \neg\mathrm{r}(a,b) \end{cases}$$

  - Δ(a,b): #Schritte nötig um b von a zu erreichen, gemäß Nachfolgerfunktion, die Basis von r
  - |dom(r)|: Größe des Definitionsbereichs der Argumente von r
  - Erinnerung: t: Threshold für T-Norm Berechnung

Von Strukturen zur Evaluationsfunktion

- für geordnete Spielbretter:
  - Manhattan-Distanz zum Ziel
  - bei mehreren identischen Figuren: mittlere Distanz
- Beispiel: Racetrack-Corridor
  - goal: `(true (cell wlane e ?x white))`
  - Distanz 3. Koordinate = 0, da Variable



Von Strukturen zur Evaluationsfunktion

- 2-dimensionales geordnetes Spielbrett:

$$\mathrm{eval}(\mathrm{f}(x,y,c),z)=\frac{1}{N}\cdot\sum_{\mathrm{f}(\mathrm{x}',\mathrm{y}',\mathrm{c})\in z}\frac{1}{2}\cdot\left(\frac{\Delta(x,x')}{|\mathrm{dom}(\mathrm{f},1)|}+\frac{\Delta(y,y')}{|\mathrm{dom}(\mathrm{f},2)|}\right)$$

  - N: Anzahl Vorkommen von f(x', y', c) in z für beliebiges x', y' gegeben c
  - Δ(x,x'): Anzahl Schritte zwischen x und x' gemäß Nachfolgerelation, die Basis für Ordnung von f
  - |dom(f,i)|: Größe Definitionsbereich des i-ten Arguments von f
- kann einfach auf höherdimensionale Spielbretter erweitert werden

Von Strukturen zur Evaluationsfunktion

- für Mengenfluent (oder Spielbretter, deren Zellzustände Mengen sind):
  - Evaluation basierend auf Unterschied zwischen Mengen in aktuellem Zustand und zu evaluierendem Zustand (etwa Ziel)
- Wichtig bei Step-Counter
  - wenn Abbruch nach Anzahl Schritten
  - Evaluation identischer Zustände (abgesehen von Step-Counter):
    - Präferenz von Zustand mit geringerem Step-Counter, solange goal nicht erfüllt
    - (Handlungsplanung: kürzere Pläne besser als längere)
- Evaluation unären Mengenfluents:

$$\mathrm{eval}(\mathrm{f}(x), z) = \frac{\Delta(x, x')}{|\mathrm{dom}(\mathrm{f}, 1)|} \quad , \text{falls } \mathrm{f}(x') \in z$$

Weitere Eigenschaften durch Simulationen

- Teams
  - gleiche Bewertung in allen Terminalzuständen → spielen zusammen
  - Annahme: alle spielen zusammen
  - wenn Terminalzustand in Simulation erreicht, prüfe Annahme
    - alle, die unterschiedliche Bewertung haben in unterschiedliche Teams
    - wenn Reduzierung auf “unser Team” und Gegner:
      - alle mit anderer Bewertung als unserer in Gegner-Team

Weitere Eigenschaften durch Simulationen

- Nicht-simultane Züge
  - Annahme: Spiel ist nicht-simultan
  - dann: in jedem Zustand höchstens ein Spieler mit > 1 gültigen Zug
  - Züge der anderen entspr. Noop-Züge
    - möglich, Namen der Noop-Züge zu finden
  - wenn Zustand mit mehreren Spielern mit mehreren gültigen Zügen
    - Spiel simultan

Quellen (Evaluationsfunktionen)

- G. Kuhlmann, K. Dresner & P. Stone: *Automatic Heuristic Construction in a Complete General Game Player*, AAAI, pp. 1457-1462, 2006
- S. Schiffel & M. Thielscher: *Automatic Construction of a Heuristic Search Function for General Game Playing*, IJCAI-Workshop on Nonmontonic Reasoning, Action and Change, 2007