Handlungsplanung und Allgemeines Spiel

„Allgemeine Verbesserungen“

Peter Kissmann

Themen Allgemeines Spiel

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

Aufbau der Vorlesung

- Allgemeine Verbesserungen
  - Verbesserungen an Spielbeschreibung
  - Dekomposition von Spielen
  - Finden von Symmetrien

Allgemeine Verbesserungen

- bisherige Verfahren sehr auf jeweilige Algorithmen abgestimmt
  - Transpositionstabellen können aber auch bei UCT verwendet werden, um Duplikate zu finden
- es gibt aber auch allgemeinere Verbesserungen, die etwa
  - auf Basis der Spielbeschreibung operieren (um Prologaufrufe zu beschleunigen)
  - Struktur der Spiele analysieren und Schlüsse ziehen

Verbesserungen an Spielbeschreibung

- Oft, Spielbeschreibungen schlecht für Prolog
- findet zu viele Duplikate
- braucht zu lange, um Formeln zu beweisen
- recht einfache Mittel können schon helfen

Ordnung von Konjunktionen

- Beispiel:
  - `(<= (goal ?x ?z))`
    `(p ?x ?y) (q ?y ?z) (distinct ?y b))`
- besser:
  - `(<= (goal ?x ?z))`
    `(p ?x ?y) (distinct ?y b) (q ?y ?z))`
- schlechter:
  - `(<= (goal ?x ?z))`
    `(distinct ?y b) (p ?x ?y) (q ?y ?z))`

Rest nicht ausgewertet, wenn ?y = b

Problem: Prolog bindet keine Variablen in Negationen

Ordnung von Konjunktionen

- Beispiel:
  - `(<= (wins ?p)`
    `(true (cell ?x ?y ?p)) (corner ?x ?y) (queen ?p))`
- Definitionsbereiche:
  - |`(true (cell ?x ?y ?p))`| = 768
  - |`(corner ?x ?y)`| = 4
  - |`(queen ?p)`| = 2
- daher besser:
  - `(<= (wins ?p)`
    `(queen ?p) (corner ?x ?y) (true (cell ?x ?y ?p)))`

nur 8 der 768 Möglichkeiten untersucht, da ?x, ?y und ?p schon gebunden

Datenextraktion

- Original:
  - `(<= (p 10) (q a))`
    `(<= (p 20) (q b))`
    `(<= (p 30) (q c))`
    `(<= (q ?x) …)`
- Annahme:
  - Auswertung von q teuer
- Bessere Version (q wird nur noch einmal ausgewertet):
  - `(<= (p ?y) (q ?x) (r ?x ?y))`
    `(r a 10)`
    `(r b 20)`
    `(r c 30)`
    `(<= (q ?x) …)`
  - mit `(r ?x ?y)` neue Relation

Dekomposition von Spielen

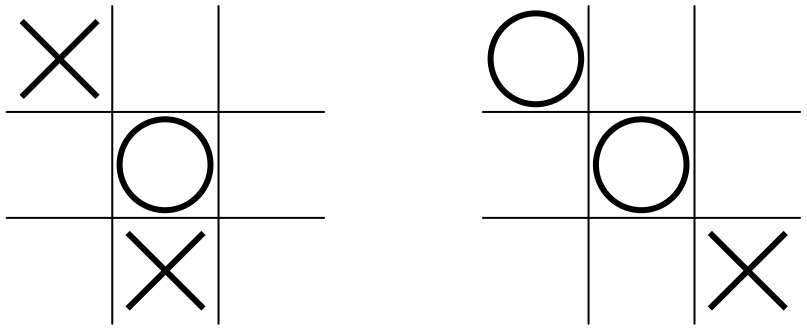
- Manchmal, größere Spiele aus kleineren Spielen zusammengesetzt
  - größere Spiele haben großen Branchingfaktor
- Dekomposition: Zerlegen großen Spiels in Menge von kleinen Spielen
  - kleinerer Branchingfaktor
  - mehr Zustände passen in RAM
- [Günther *et al.*, 2009], [Cox *et al*., 2009] und [Zhao *et* al., 2009]
  - hier anhand von [Günther *et al.*, 2009]

Zusammengesetzte Spiele

- gleichzeitiges Spielen zweier Spiele
  - Branching-Faktoren: a und b
  - Gesamtbranching-Faktor: a * b
  - Knoten in Suchtiefe n: $(a * b)^n$
  - Branching-Faktor (faktorisierte Spiele): a + b
  - Knoten in Suchtiefe n (faktorisierte Spiele): $(a + b)^n$

Zusammengesetzte Spiele

- Beispiel: Double Tic-Tac-Toe
- zwei TTT-Spiele parallel (x-Spieler hat ersten Zug auf board1, o-Spieler ersten Zug auf board2)



- #gültige Züge: 81, 64, 49, 36, 25, 16, 9, 4, 1
- #gültige Züge (faktorisierte Spiele): 18, 16, 14, 12, 10, 8, 6, 4, 2

Zusammengesetzte Spiele

- auch möglich: mehrere Spiele hintereinander spielen, Bewertung ganz am Ende
- Branching-Faktor unverändert
- aber: schwierig, zu erkennen, wann einzelne Spiele gewonnen

Teilspielerkennung

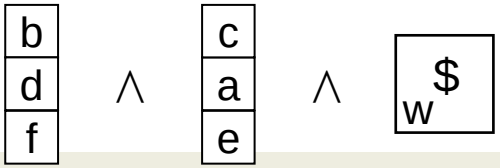
- unabhängige Teilspiele durch Abhängigkeitsgraphen identifizieren
  - Hier: Abhängigkeitsgraph für Züge:
    - Vorbedingungen
    - positive Effekte
    - negative Effekte
- Jede Zusammenhangskomponente entspricht Teilspiel

Teilspielerkennung

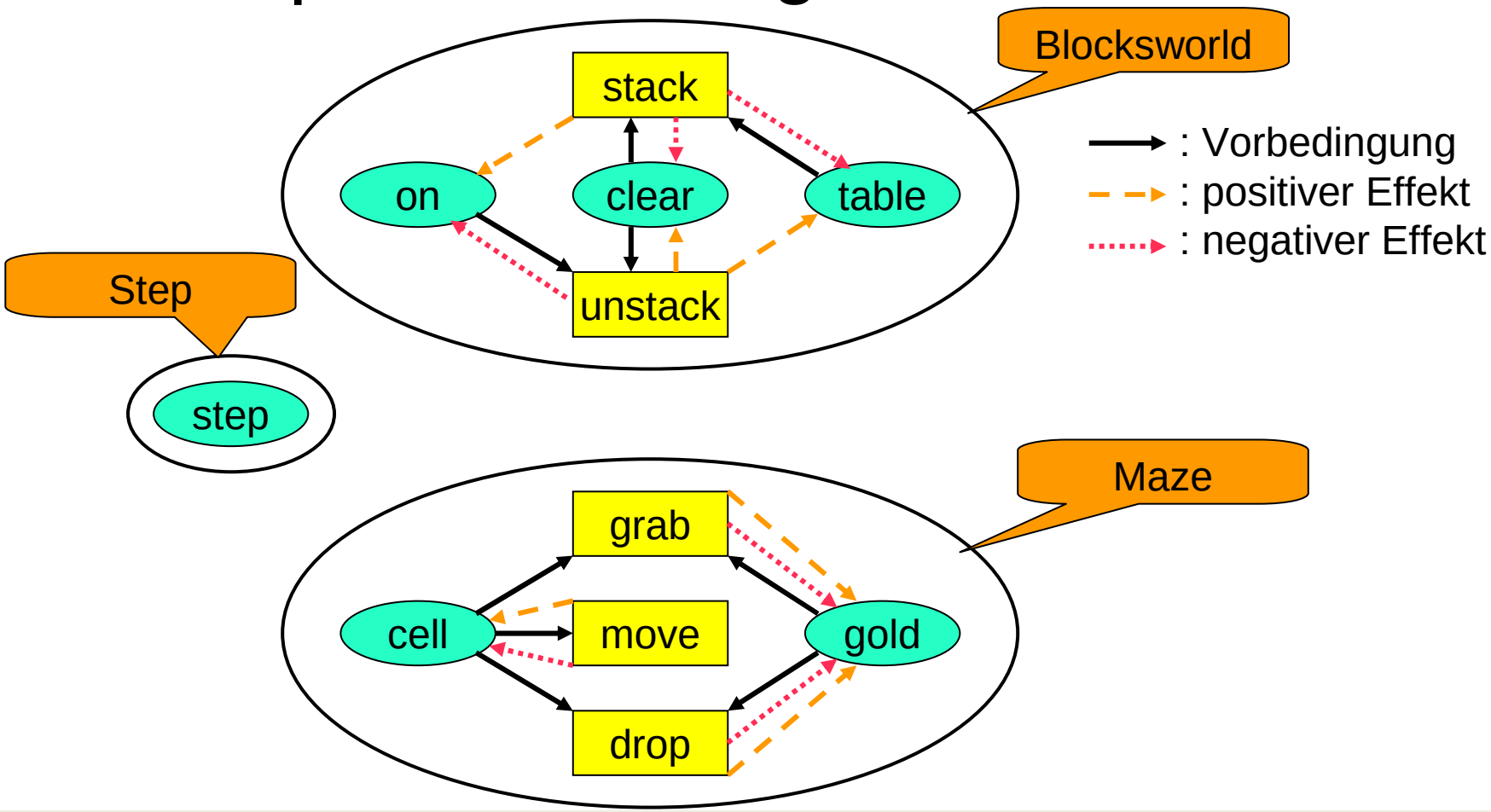
- Beispiel: incredible (blocks + maze)
  - Initialzustand:

  - Terminierung:

  - Ziel:



Teilspielerkennung



Teilspielerkennung

- Problem in GDL: positive und negative Effekte nicht explizit
- daher: verwendung potenzieller (positiver / negativer) Effekte und potenzieller Vorbedingungen
  - Fluent `f` potenzieller positiver Effekt von Zug `m`, falls es Regel `(<= (next (f x1 … xn)) B)` gibt, so dass `B` nicht `(true (f x1 … xn))` impliziert und mit ∃`p`, `y1`, …, `ym`: `(does p (m y1 … ym))` kompatibel ist
    - (`f` ist potenzieller positiver Effekt von Zug `m`, falls es aus einer Nicht-Frame Regel resultiert, die mit Zug `m` kompatibel ist)

Teilspielerkennung

- Fluent `f` potenzieller negativer Effekt von Zug `m`, falls es keine Regel `(<= (next (f x1 … xn)) B)` gibt, so dass ∀p, y1, …, ym: `((and (true (f x1 … xn)) (does p (m y1 … ym))` → `B)`
  - (`f` ist potenzieller negativer Effekt von Zug `m`, falls es keine Frame-Regel für `f` gibt, die mit `m` kompatibel ist)
- Fluent `f` potenzielle Vorbedingung für Zug `m`, falls `f` im Aufrufgraph des Body einer Regel mit
  - Head `(legal p (m y1 … ym))` oder
  - Head `(next (f' x1 … xn))`, mit `f'` potenzieller positiver oder negativer Effekt von `m`

Teilspielerkennung

- Jeder tatsächliche (postive / negative) Effekt ist auch potenzieller Effekt
- Umkehrung gilt nicht
- daher: Abhängigkeitsgraph kann überflüssige Kanten enthalten
  - aber: Finden der exakten positiven und negativen Effekte würde vollständige Suche bedeuten

Teilspielerkennung

- Problem bei Zug-unabhängigen Fluents (etwa einem Step-Counter)
  - gemäß Definition sind sie positive sowie negative Effekte aller Züge
  - damit alle Züge in gleicher Zusammenhangskomponente
  - damit keine Zerteilung in Teilspiele möglich
- Behebung: Zug-unabhängige Fluents identifizieren und in eigenes Teilspiel stecken
  - Fluent `f` zug-unabhängig, falls
    - Aufrufgraph jeder Regel mit Head `(next (f x1 … xn))` keine Züge oder Fluents außer `f` enthält und
    - `f` in keinem Aufrufgraphen des Body irgendeiner next- oder legal-Regel auftaucht

Teilspielerkennung

- weiteres Problem: nur für Spiele ohne gleichzeitige Züge in Teilspielen
  - jeder Spieler nur ein Zug pro Zustand
  - gleichzeitige Züge in mehreren Teilspielen = ein Zug, der mehrere Teilspiele betrifft
  - Teilspiele in Abhängigkeitsgraph damit nicht getrennt
- Erweiterung (auch zu Mehrpersonenspielen): [Zhao *et al.*, 2009] (später)

Suche mit Teilspielen

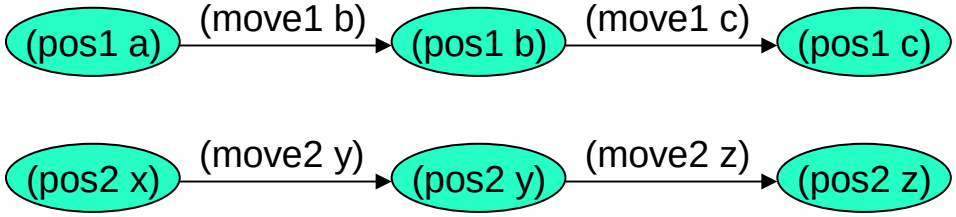
- in [Günther *et al.*, 2009] nur für Einpersonenspiele
- Was tun mit gefundenen Teilspielen?
- Naïve Idee:
  - Jedes Teilspiel wie eigenständiges Spiel behandeln
  - Resultat: Lokaler Plan (lokale Lösung)
  - Zusammenführen lokaler Pläne

Globale Suche

- Problem: Resultat (Zusammenführen lokaler Pläne) nicht unbedingt optimal
  - goal nur auf vollständigen Zuständen definiert
    - lokale Suchen können Bewertung partiellen Zustands nicht berechnen
  - Verknüpfung der Pläne muss nicht klappen
    - terminal kann Spiel beenden, bevor alle Pläne abgearbeitet
- Abhilfe:
  - alle Permutationen von Zügen untersuchen, die Ordnung der Züge im jeweiligen Teilspiel berücksichtigt
  - beste von diesen auswählen

Globale Suche

- Beispiel:
  - Spiel mit zwei Teilspielen gemäß Graphen



  - Initialzustand: [(pos1 a), (pos2 x)]
  - Terminalzustände:
    - [(pos1 a), (pos2 z)]: 0 Punkte
    - [(pos1 c), (pos2 x)]: 0 Punkte
    - [(pos1 c), (pos2 z)]: 100 Punkte
  - Optimale Lösungen:
    - [(move1 b), (move2 y), (move1 c), (move2 z)]
    - [(move1 b), (move2 y), (move2 z), (move1 c)]
    - [(move2 y), (move1 b), (move1 c), (move2 z)]
    - [(move2 y), (move1 b), (move2 z), (move1 c)]

Teilspielerkennung (erweitert)

- Problem bisher: nur Fluents mit Zügen verknüpft - unabhängig von evtl. Instanziierungen
  - Beispiel: Nim
    - mehrere Stapel von Streichhölzern
    - Zug betrifft immer nur einen Stapel
    - damit eigentlich unabhängige Spiele
    - aber: Stapel über fluent (heap id nr) identifiziert (id: welcher Stapel; nr: #Streichhölzer)

Teilspielerkennung (erweitert)

- Behebung: Verwendung von teilinstanziierten Fluents / Zügen
  - i-tes Argument von Fluent `f` wird instanziiert gdw. für jede Regel `(<= (next (f … xi …)) B)` der Aufrufgraph von `B` `(true (f … xi …))` aber nicht `(true (f … xi' …))` mit `xi` ≠ `xi'` enthält
    - (Wert von Argument i ändert sich nicht und Instanzen von `f` mit unterschiedlichen Werten für Argument i interagieren nicht)
  - j-tes Argument von Zug `m` wird instanziiert gdw. i-tes Argument von Fluent `f` ist instanziiert und es gibt eine Regel `(<= (next (f … xi …)) B)` für die der Aufrufgraph von `B` `(does r, (m … yj …))` mit `yj` = `xi` enthält
    - (instanziiertes Argument von `f` hängt von Argument von `m` ab)

Teilspielerkennung (erweitert)

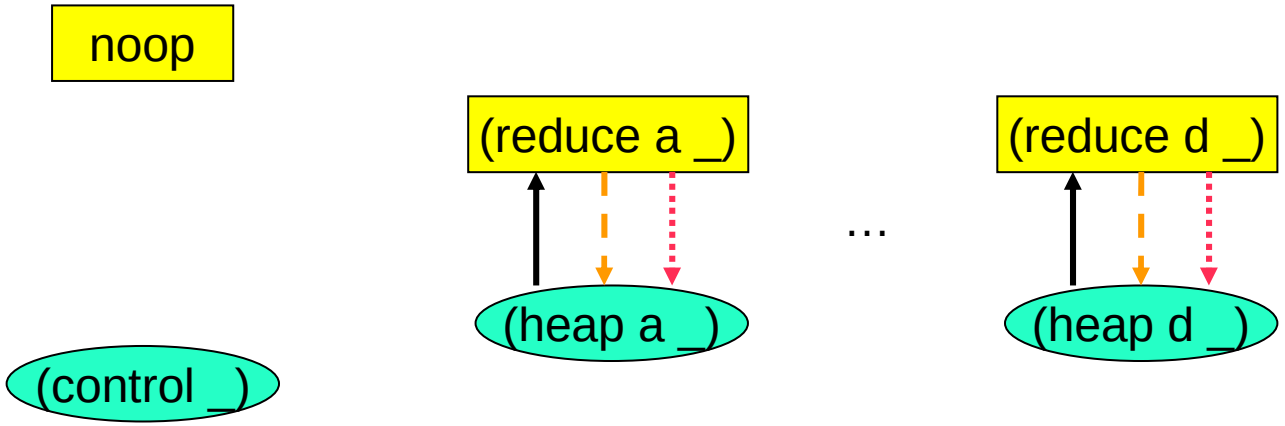
- weiteres Problem bisher:
  - typischerweise kombinierte Züge (Kombination von Zügen aller Spieler)
  - welcher ursprüngliche Zug verantwortlich für positive / negative Effekte von kombiniertem Zug?
- Behebung:
  - Erweiterung von Definition potenzieller Effekte nach Rollen, die Fluent betreffen

Teilspielerkennung (erweitert)

- Rolle `r` betrifft Fluent `f` gdw. Regel `(<= (next f) B)` existiert und Aufrufgraph von `B` enthält `(does r' m)` und `r` und `r'` unifizierbar
- Zug `m` noop-Zug, falls einzig gültiger Zug für nicht-aktiven Spieler und `m` nicht in Aufrufgraph einer next-Regel
- Fluent `f` potenzieller positiver Effekt von nicht-Noop-Zug `m`, falls es Regel `(<= (next (f x1 … xn)) B)` gibt, so dass `B` nicht `(true (f x1 … xn))` impliziert und mit $\exists$`p`, `y1`, …, `ym`: `(does p (m y1 … ym))` kompatibel ist und `p` betrifft `f`
- Fluent `f` potenzieller negativer Effekt von nicht-Noop-Zug `m`, falls es keine Regel `(<= (next (f x1 … xn)) B)` gibt, so dass für alle `p`, die `f` betreffen, $\forall$`y1`, …, `ym`: `((and (true (f x1 … xn)) (does p (m y1 … ym)))` $\rightarrow$ `B`)
- Fluent `f` potenzielle Vorbedingung für Zug `m`, falls `f` im Aufrufgraph des Body einer Regel mit
  - Head `(legal p (m y1 … ym))` oder
  - Head `(next (f' x1 … xn))`, mit `f'` potenzieller positiver oder negativer Effekt von `m`

Teilspielerkennung (erweitert)

- Abhängigkeitsgraph von Nim mit 4 Stapeln



Symmetrien [Schiffel, 2010]

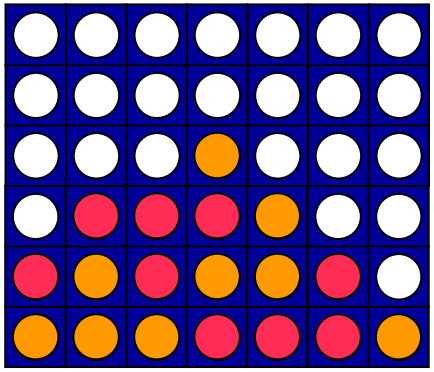
- Symmetrien können logisch aus GDL-Regeln abgeleitet werden
- Symmetrierelation ist Äquivalenzrelation, so dass
  - zwei symmetrische Zustände sind entweder beide terminal oder nicht-terminal
  - wenn beide terminal sind haben sie identische Bewertungen
  - wenn sie nicht-terminal sind sind gültige Züge symmetrisch zueinander und generieren symmetrische Zustände

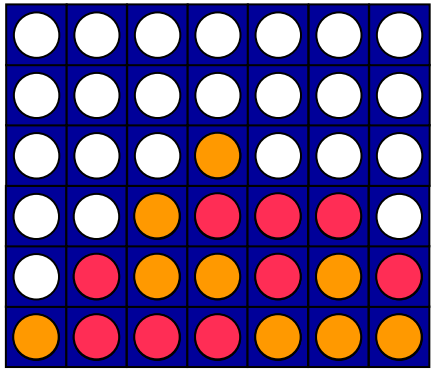
Symmetrien

- zueinander symmetrische Zustände können als identisch angenommen werden
- kann Suchraum stark beschneiden
  - von allen zueinander symmetrischen Zuständen braucht bloß einer voll analysiert zu werden

Symmetrien

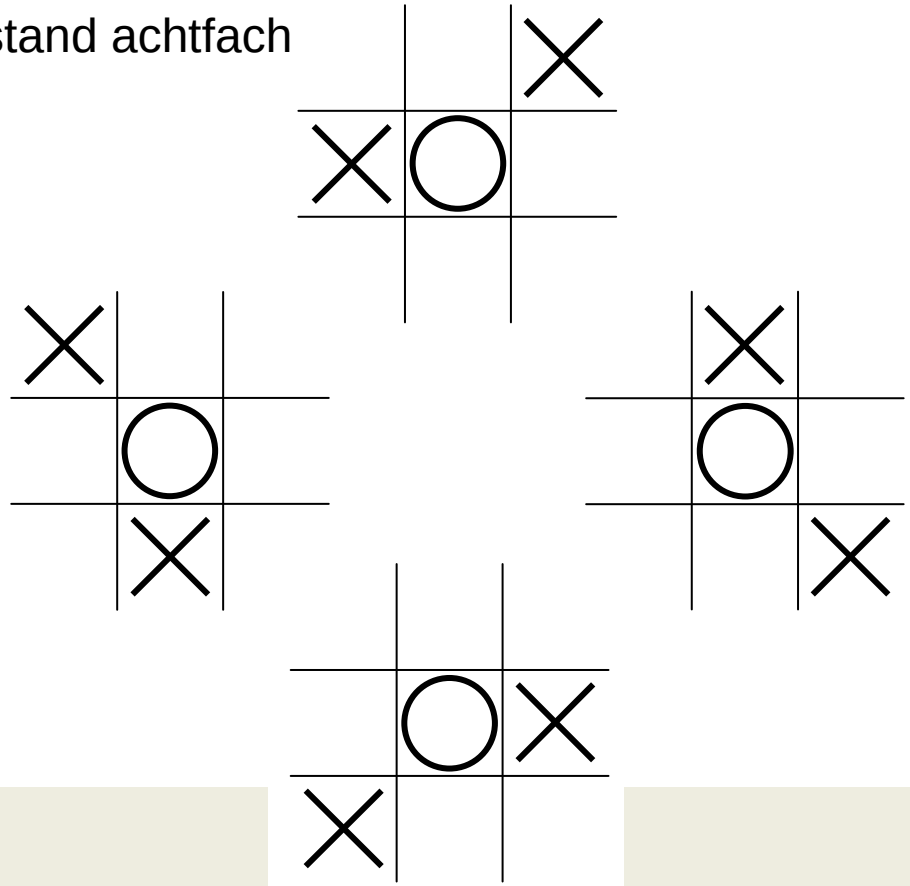
- 4-Gewinnt: spiegelsymmetrisch
  - jeder Zustand doppelt





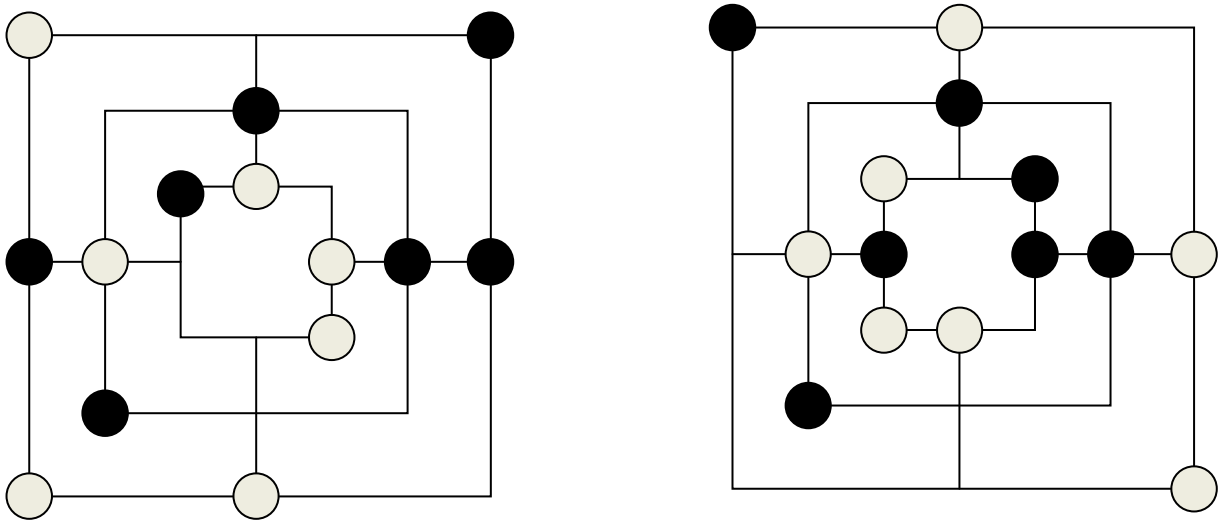
Symmetrien

- Tic-Tac-Toe: spiegelsymmetrisch und rotationssymmetrisch
  - jeder Zustand achtfach



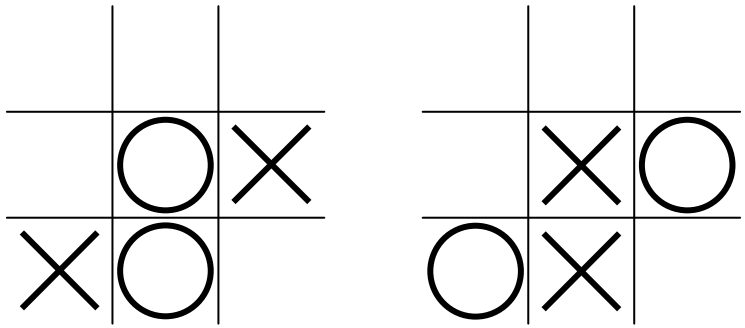
Symmetrien

- Mühle: spiegelsymmetrisch und rotationssymmetrisch und symmetrisch bzgl. Vertauschungen von äußerem und innerem Ring
  - jeder Zustand 16-fach



Symmetrien

- Tic-Tic-Toe (simultanes Tic-Tac-Toe): Symmetrien wie Tic-Tac-Toe + symmetrisch bzgl. Spieler
  - xplayer und oplayer können vertauscht werden:
    - in jedem Zug haben sie identische Möglichkeiten
    - vertauschen von ihnen resultiert in symmetrischem Zustand (alle x und o auf Brett vertauscht)



Quellen

- M. Günther, S. Schiffel & M. Thielscher: *Factoring General Games*, IJCAI-Workshop on GGP, pp. 27-34, 2009
- E. Cox, E. Schkufza, R. Madsen & M. Genesereth: *Factoring General Games using Propositional Automata*, IJCAI-Workshop on GGP, pp. 13-20, 2009
- D. Zhao, S. Schiffel & M. Thielscher: *Decomposition of Multi-Player Games*, Australasian Joint Conference on Advances in Artificial Intelligence, pp. 475-484, 2009
- S. Schiffel: *Symmetry Detection in General Game Playing*, AAAI, pp. 980-985, 2010