AAAI'08 Tutorial

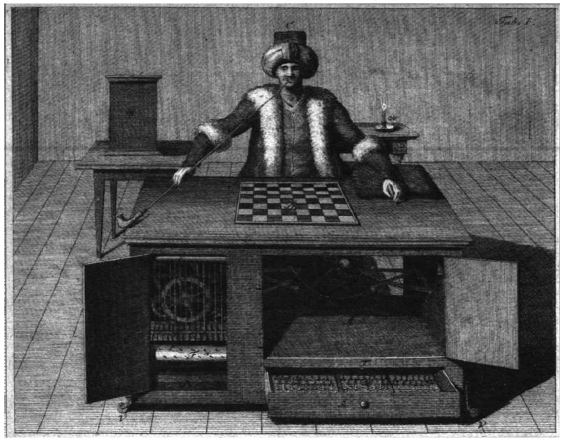
General Game Playing

Michael Thielscher, Dresden

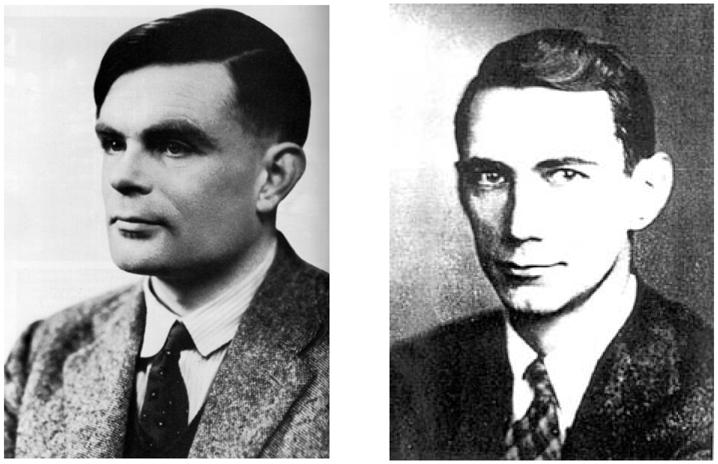
Some of the material presented in this tutorial originates in work by Michael Genesereth and the Stanford Logic Group. We greatly appreciate their contribution.

Chess Players

The Turk (18th Century)



Alan Turing & Claude Shannon (~1950)



Deep-Blue Beats World Champion (1997)

In the early days, game playing machines were considered a key to Artificial Intelligence (AI).

But chess computers are highly specialized systems. Deep-Blue's intelligence was limited. It couldn't even play a decent game of Tic-Tac-Toe or Rock-Paper-Scissors.

With General Game Playing many of the original expectations with game playing machines get revived.

A General Game Player is a system that

- understands formal descriptions of arbitrary strategy games
- learns to play these games well without human intervention

Traditional research on game playing focuses on

- constructing specific evaluation functions
- building libraries for specific games

The intelligence lies with the programmer, not with the program!

A General Game Player needs to exhibit much broader intelligence:

- abstract thinking
- strategic planning
- learning

Rather than being concerned with a specialized solution to a narrow problem, General Game Playing encompasses a variety of AI areas:

- Game Playing
- Knowledge Representation
- Planning and Search
- Learning

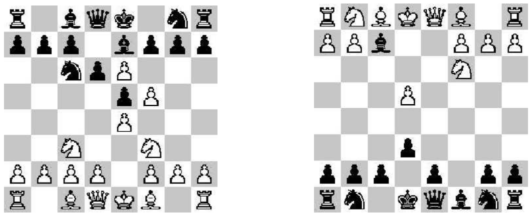
General Game Playing is considered a grand AI Challenge

General Game Playing and AI

Games	Agents
Deterministic, complete information	Competitive environments
Nondeterministic, partially observable	Uncertain environments
Rules partially unknown	Unknown environment model
Robotic player	Real-world environments

Application (1)

Commercially available chess computers can't be used for a game of Bughouse Chess.



An adaptable game computer would allow the user to modify the rules for arbitrary variants of a game.

Application (2): Economics

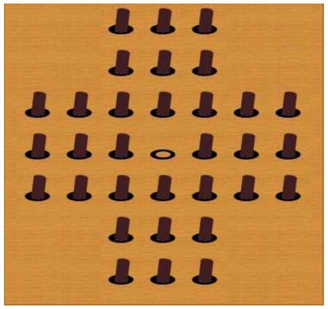
A General Game Playing system could be used for negotiations, marketing strategies, pricing, etc.

It can be easily adapted to changes in the business processes and rules, new competitors, etc.

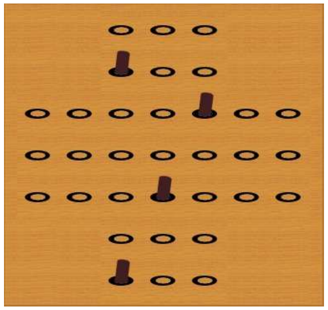
The rules of an *e*-marketplace can be formalized as a game, so that agents can automatically learn how to participate.

Example Games

Single-Player, Deterministic

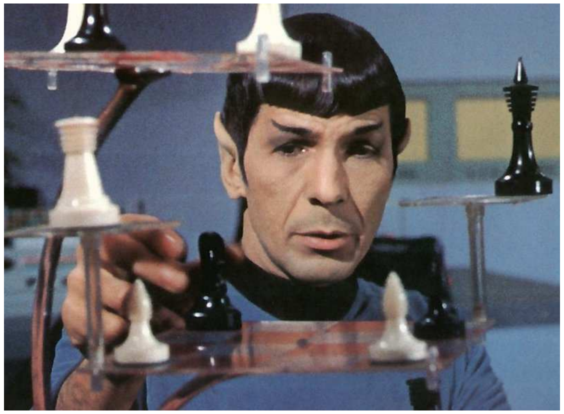


Single-Player, Deterministic

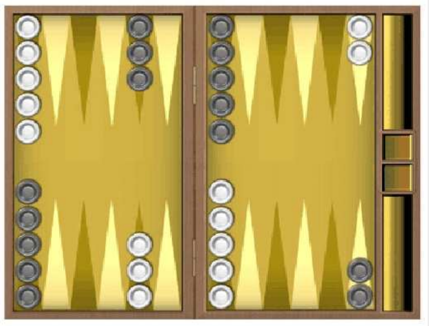


Two-Player, Zero-Sum, Deterministic

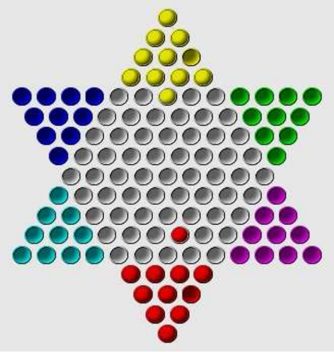
Two-Player, Zero-Sum, Deterministic



Two-Player, Zero-Sum, Nondeterministic



n-Player, Deterministic



n-Player, Incomplete Information, Nondeterministic

General Game Playing Initiative

`games.stanford.edu`

- Game description language
- Variety of games/actual matches
- Basic player available for download
- Annual world cup @AAAI (since 2005)
  Price money: US$ 10,000

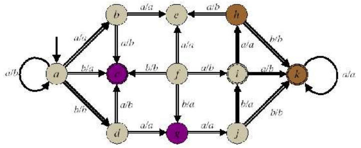
(deterministic games w/ complete information only)

Roadmap

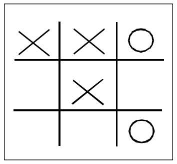
- The Game Description Language GDL:
  Knowledge Representation
- How to make legal moves:
  Automated Reasoning
- How to solve simple games:
  Planning & Search
- How to play well:
  Learning

Game Description Language

Every finite game can be modeled as a state transition system



But direct encoding impossible in practice



19,683 states

~ 10^{43} legal positions

Modular State Representation: Fluents

`cell(X,Y,C)`

`X ∈ {a,…,h}`

`Y ∈ {1,…,8}`

`C ∈ {whiteKing,…,blank}`

`control(P)`

`P ∈ {white,black}`

Fluent Representation for Chess (2)

`canCastle(P,S)`

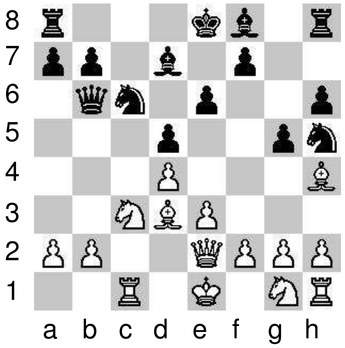
`P ∈ {white,black}`

`S ∈ {kingsSide,queensSide}`

`enPassant(C)`

`C ∈ {a,…,h}`

Actions



`move(U,V,X,Y)`

`U,X ∈ {a,…,h}`

`V,Y ∈ {1,…,8}`

`promote(X,Y,P)`

`X,Y ∈ {a,…,h}`

`P ∈ {whiteQueen,…}`

Game Rules (I)

- Players `roles([white,black])`
- Initial position `init(cell(a,1,whiteRook)) ∧ …`
- Legal Moves

```
legal(white,promote(X,Y,P))  <=
    true(cell(X,7,whitePawn)) ∧ …
```

Game Rules (II)

- Position updates

```
next(cell(X,Y,C)) <=
    does(P,move(U,V,X,Y))
   ∧ true(cell(U,V,C))
```

- End of game

```
terminal <=
    checkmate ∨ stalemate
```

- Result

```
goal(white,100) <=
        true(control(black))
          ∧ checkmate
goal(white, 50) <= stalemate
```

Clausal Logic

Variables: `X, Y, Z`
Constants: `a, b, c`
Functions: `f, g, h`
Predicates: `p, q, r, =`
Logical Operators: `¬, ∧, ∨, <=`

Terms: `X, Y, Z, a, b, c, f(a), g(a,X), h(a,b,f(Y))`
Atoms: `p(a,b)`
Literals: `p(a,b), ¬q(X,f(a))`

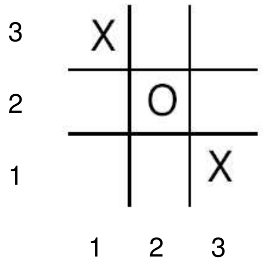
Clauses: `Head <= Body`
`Head`: relational sentence
`Body`: logical sentence built from ∧, ∨, literal

Game-Independent Vocabulary

Relations

```
roles(list-of(player))
init(fluent)
true(fluent)
does(player,move)
next(fluent)
legal(player,move)
goal(player,value)
terminal
```

Axiomatizing Tic-Tac-Toe: Fluents



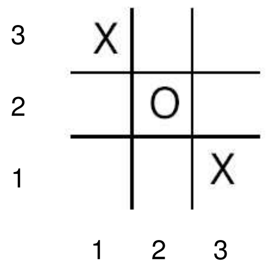
`cell(X,Y,M)`

`X,Y ∈ {1,2,3}`
`M ∈ {x,o,b}`

`control(P)`

`P ∈ {xplayer,oplayer}`

Axiomatizing Tic-Tac-Toe: Actions



`mark(X,Y)`

X,Y ∈ {1,2,3}

`noop`

Tic-Tac-Toe: Vocabulary

- Constants

`xplayer, oplayer`	Players
`x, o, b`	Marks

- Functions

`cell(number,number,mark)`	Fluent
`control(player)`	Fluent
`mark(number,number)`	Action

- Predicates
  - `row(number,mark)`
  - `column(number,mark)`
  - `diagonal(mark)`
  - `line(mark)`
  - `open`

Players and Initial Position

```
roles([xplayer,oplayer])
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))
```

Preconditions

```
legal(P,mark(X,Y)) <=
    true(cell(X,Y,b)) ∧
    true(control(P))

legal(xplayer,noop) <=
    true(cell(X,Y,b)) ∧
    true(control(oplayer))

legal(oplayer,noop) <=
    true(cell(X,Y,b)) ∧
    true(control(xplayer))
```

Update

```
next(cell(M,N,x))<= does(xplayer,mark(M,N))

next(cell(M,N,o))<= does(oplayer,mark(M,N))

next(cell(M,N,W))<= true(cell(M,N,W)) ∧ ¬W=b

next(cell(M,N,b))<= true(cell(M,N,b)) ∧
                    does(P,mark(J,K)) ∧ (¬M=J ∨ ¬N=K)

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

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

Termination

```
terminal <= line(x) ∨ line(o)
terminal <= ¬open

line(W) <= row(M,W)
line(W) <= column(N,W)
line(W) <= diagonal(W)

open <= true(cell(M,N,b))
```

Supporting Concepts

```
   row(M,W)  <= true(cell(M,1,W)) ∧
                true(cell(M,2,W)) ∧
                true(cell(M,3,W))

column(N,W)  <= true(cell(1,N,W)) ∧
                true(cell(2,N,W)) ∧
                true(cell(3,N,W))

diagonal(W)  <= true(cell(1,1,W)) ∧
                true(cell(2,2,W)) ∧
                true(cell(3,3,W))

diagonal(W)  <= true(cell(1,3,W)) ∧
                true(cell(2,2,W)) ∧
                true(cell(3,1,W))
```

Goals

```
goal(xplayer,100) <= line(x)
goal(xplayer,50)  <= ¬line(x) ∧ ¬line(o) ∧ ¬open
goal(xplayer,0)   <= line(o)

goal(oplayer,100) <= line(o)
goal(oplayer,50)  <= ¬line(x) ∧ ¬line(o) ∧ ¬open
goal(oplayer,0)   <= line(x)
```

Finite Games

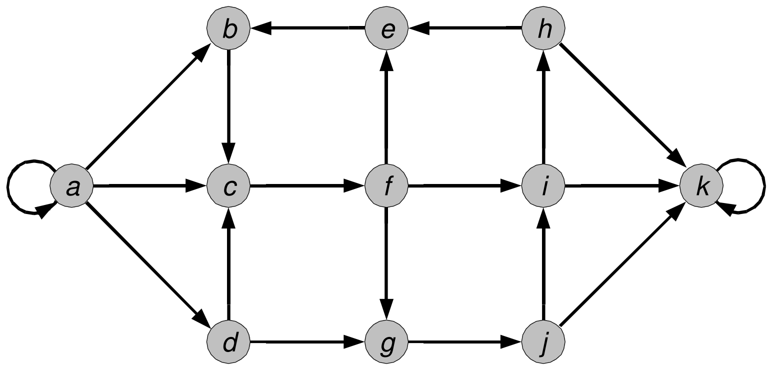
Finite Environment

- Game “world” with finitely many states
- One initial state and one or more terminal states
- Fixed finite number of players
- Each with finitely many “percepts” and “actions”
- Each with one or more goal states

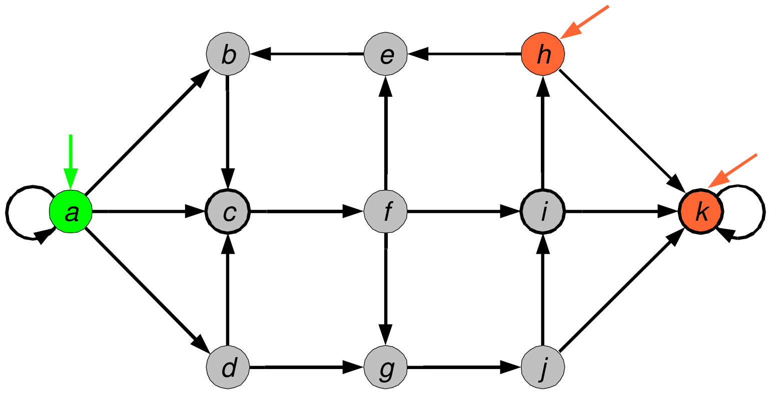
Causal Model

- Environment changes only in response to moves
- Synchronous actions

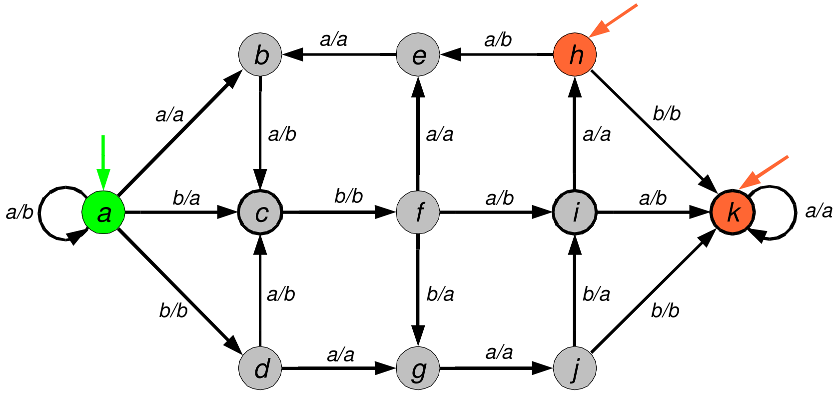
Games as State Machines



Initial State and Terminal States



Simultaneous Actions



Game Model

An *n*-player game is a structure with components:

S – set of states
$A_1, \ldots, A_n$ – *n* sets of actions, one for each player

$I_1, \ldots, I_n$ – where $I_i \subseteq A_i \times S$, the legality relations
u: $S \times A_1 \times \ldots \times A_n \rightarrow S$ – update function

$s_1 \in S$ – initial game state
$t \subseteq S$ – the terminal states
$g_1, \ldots g_n$ – where $g_i \subseteq S \times \mathbb{N}$, the goal relations

---

GDL for Trading Games: Example (English Auction)

```
role(bidder_1) ∧ ... ∧ role(bidder_n)

init(highestBid(0))
init(round(0))

legal(P,bid(X)) <=
   true(highestBid(Y)) ∧ greaterthan(X,Y)
legal(P,noop)

terminal <= true(round(10))
```

```
next(winner(P)) <= does(P,bid(X)) ∧ bestbid(X)
next(highestBid(X)) <= does(P,bid(X)) ∧ bestbid(X)
next(winner(P)) <= true(winner(P)) ∧ not bid
next(highestBid(X)) <= true(highestBid(X) ∧ not bid
next(round(N)) <= true(round(M)), successor(M,N)

bid <= does(P,bid(X))
bestbid(X) <= does(P,bid(X)) ∧ not overbid(X)
overbid(X) <= does(P,bid(Y)) ∧ greaterthan(Y,X)
```

Try it Yourself: Play this Game!

```
role(you)

init(step(1))
init(cell(1,onecoin))
init(cell(Y,onecoin)) <= succ(X,Y)
succ(1,2) ∧ succ(2,3) ∧ ... ∧ succ(7,8)

next(step(Y)) <= true(step(X)) ∧ succ(X,Y)
next(cell(X,zerocoins)) <= does(you,jump(X,Y))
next(cell(Y,twocoins)) <= does(you(jump(X,Y))
next(cell(X,C)) <= does(you,jump(Y,Z)) ∧
                   true(cell(X,C)) ∧
                   distinct(X,Y)∧distinct(X,Z)

terminal <= ~continuable
continuable <= legal(you,M)
goal(you,100) <= true(step(5))
goal(you,0) <= true(cell(X,onecoin))
```

```
legal(you,jump(X,Y)) <=
   true(cell(X,onecoin)) ∧ true(cell(Y,onecoin)) ∧
   ( twobetween(X,Y) | twobetween(Y,X) )

zerobetween(X,Y) <= succ(X,Y)
zerobetween(X,Y) <= succ(X,Z) ∧ true(cell(Z,zerocoins))
                    ∧ zerobetween(Z,Y)
onebetween(X,Y) <= succ(X,Z) ∧ true(cell(Z,zerocoins))
                   ∧ onebetween(Z,Y)
onebetween(X,Y) <= succ(X,Z) ∧ true(cell(Z,onecoin))
                   ∧ zerobetween(Z,Y)
twobetween(X,Y) <= succ(X,Z) ∧ true(cell(Z,zerocoins))
                   ∧ twobetween(Z,Y)
twobetween(X,Y) <= succ(X,Z) ∧ true(cell(Z,onecoin))
                   ∧ onebetween(Z,Y)
twobetween(X,Y) <= succ(X,Z) ∧ true(cell(Z,twocoins))
                   ∧ zerobetween(Z,Y)
```

Automated Reasoning

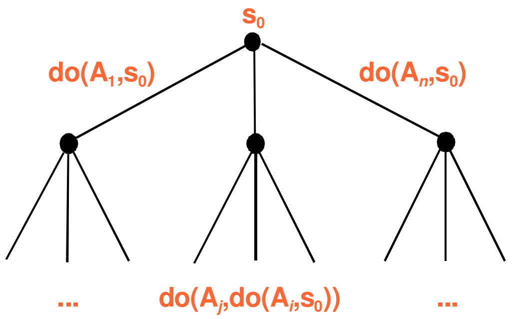
Game descriptions are a good example of knowledge representation with formal logic.

Automated reasoning about actions necessary to

- determine legal moves
- update positions
- recognize end of game

Background: Reasoning about Actions

McCarthy's Situation Calculus (1963)



Reasoning about Actions using Situations

Effect Axioms:

$(\forall S)(\forall M,N)\ cell(M,N,x,do(xplayer,mark(M,N),S))$

The Frame Problem (McCarthy & Hayes, 1969) arises because mere effect axioms do not suffice to infer non-effects!

How does cell(2,2,o,s) imply cell(2,2,o,do(xplayer,mark(3,3),s))?

The Frame Problem

A frame axiom for Tic-Tac-Toe:

$$(\forall S)(\forall ...)\ cell(M,N,W,do(P,mark(J,K),S)) \Leftarrow cell(M,N,W,S) \wedge (M \neq J \vee N \neq K)$$

Compare this to the GDL axiom

```
next(cell(M,N,W))<= true(cell(M,N,W)) ∧ ¬W=b

next(cell(M,N,b))<= true(cell(M,N,b)) ∧
                    does(P,mark(J,K)) ∧ (¬M=J ∨ ¬N=K)
```

In a domain with *m* actions and *n* fluents, in the order of *n*·*m* frame axioms are needed.

Successor State Axioms

"If AI can be said to have a classic problem, then the Frame Problem is it. Like all good open problems it is subtle, challenging, and it has led to significant new technical and conceptual developments in the field." (Reiter, 1991)

A successor state axiom (Reiter, 1991) for every fluent Φ avoids extra frame axioms:

$$(\forall P,A,S)\ \Phi(do(P,A,S)) <=> \Gamma^{+} \vee [\Phi(S) \wedge \neg\Gamma^{-}]$$

Γ^{+}: reasons for Φ to become true

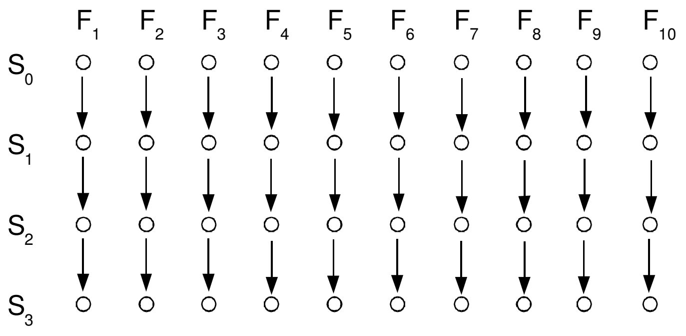
Γ^{-}: reasons for Φ to become false

Successor State Axioms for Tic-Tac-Toe

$$
\begin{array}{ll}
(\forall P,A,S)(\forall \ldots)\ cell(M,N,W,do(P,A,S)) <=> & \\
\quad W=x \wedge P=xplayer \wedge A=mark(M,N) \vee & \\
\quad W=o \wedge P=oplayer \wedge A=mark(M,N) \vee & \Gamma^{+} \\
\quad cell(M,N,W,S) \wedge \neg A=mark(M,N) & \Gamma^{-}
\end{array}
$$

$$
\begin{array}{ll}
(\forall P,A,S)(\forall R)\ control(R,do(P,A,S) <=> & \\
\quad R=xplayer \wedge control(oplayer,S) \vee & \\
\quad R=oplayer \wedge control(xplayer,S) & \Gamma^{+}
\end{array}
$$

The Computational Frame Problem



Fluent Calculus

A state update axiom (T., 1999) for every action α avoids separate update axioms for every fluent:

$$
\begin{array}{l}
(\forall S)\ \triangle_1(S) \wedge state(do(P,\alpha,S)) = state(S) - \vartheta_1^{-} + \vartheta_1^{+} \\
\qquad \vee \ldots \vee \\
\qquad \triangle_k(S) \wedge state(do(P,\alpha,S)) = state(S) - \vartheta_k^{-} + \vartheta_k^{+}
\end{array}
$$

ϑ^{+}: fluents that become true

ϑ^{-}: fluents that become false

(where subtraction $z-\vartheta^{-}$ and addition $z+\vartheta^{+}$ axiomatically defined)

State Update Axioms for Tic-Tac-Toe

(∀S)(∀...) control(oplayer,S) ∧ state(do(xplayer,mark(M,N),S)) =
state(S) – control(oplayer) ϑ_1^-
+ control(xplayer) + cell(M,N,o) ϑ_1^+
∨ control(xplayer,S) ∧ state(do(oplayer,mark(M,N),S)) =
state(S) – control(xplayer) ϑ_2^-
+ control(oplayer) + cell(M,N,x) ϑ_2^+

(∀S)(∀P) state(do(P,noop)) = state(S)

Action Programming Languages

Morgan & Claypool Publishers

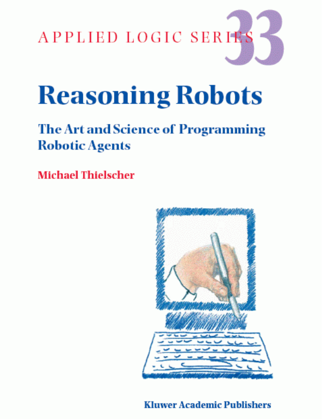
Action Programming Languages

Michael Thielscher

Synthesis Lectures on Artificial Intelligence and Machine Learning

2008

The Fluent Calculus and FLUX



APPLIED LOGIC SERIES 33

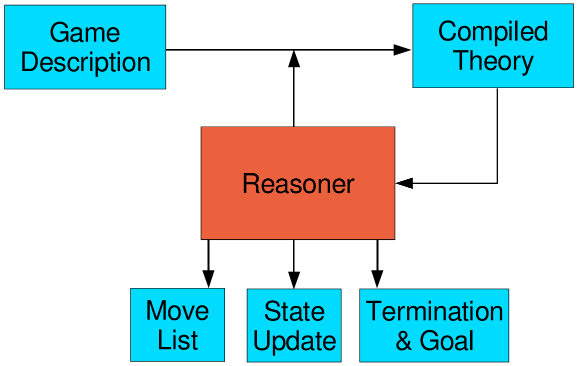
Reasoning Robots

The Art and Science of Programming Robotic Agents

Michael Thielscher

Kluwer Academic Publishers

A General Architecture



Game Description

Compiled Theory

Reasoner

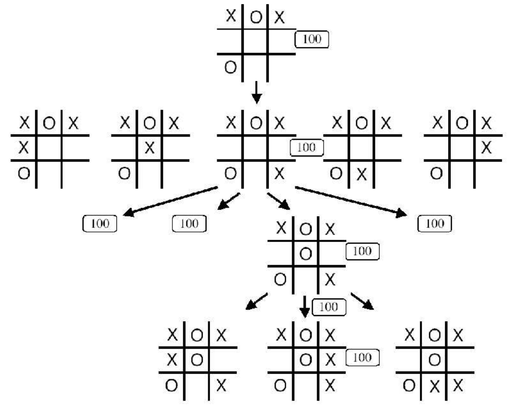
Move List

State Update

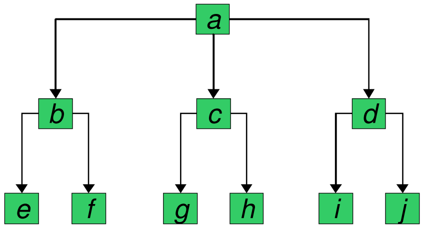
Termination & Goal

Planning and Search

Game Tree Search (General Concept)



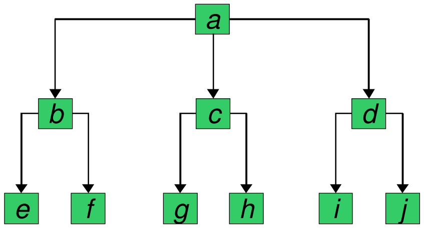
Breadth-First Search



a b c d e f g h i j

Advantage: Finds shortest solution
Disadvantage: Consumes large amount of space

Depth-First Search



a b e f c g h d i j

Advantage: Small intermediate storage
Disadvantage: Susceptible to garden paths
Disadvantage: Susceptible to infinite loops

Time and Space Comparison

Worst case for search depth *d*, solution at depth *k*

Time	*Binary*	*Branching b*
Depth-First	$2^d - 2^{d-k}$	$\frac{b^d - b^{d-k}}{b-1}$
Breadth-First	$2^k - 1$	$\frac{b^k - 1}{b-1}$

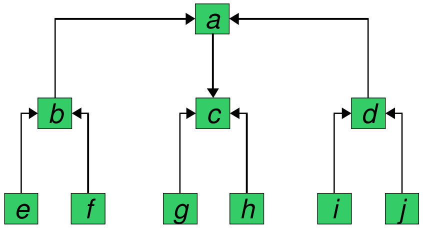
Space	*Binary*	*Branching b*
Depth-First	d	$(b - 1) * (d - 1) + 1$
Breadth-First	2^{k-1}	b^{k-1}

Iterative Deepening

Run depth-limited search repeatedly

- starting with a small initial depth *d*
- incrementing on each iteration $d := d + 1$
- until success or run out of alternatives

Example



$d = 1$: *a*
$d = 2$: *a b c d*
$d = 3$: *a b e f c g h d i j*

Advantage: Small intermediate storage
Advantage: Finds shortest solution
Advantage: Not susceptible to garden paths
Advantage: Not susceptible to infinite loops

Time Comparison

Worst case for branching factor 2

Depth	*Iterative Deepening*	*Depth-First*
1	1	1
2	4	3
3	11	7
4	26	15
5	57	31
n	$2^{n+1} - n - 2$	$2^n - 1$

Theorem: The cost of iterative deepening search is *b*/(*b*-1) times the cost of depth-first search (where *b* is the branching factor).

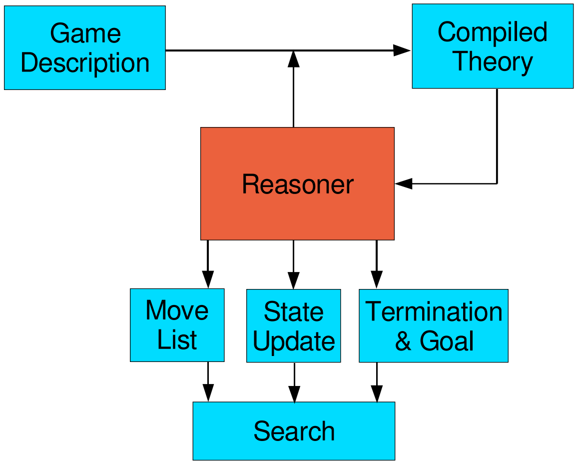
Game Rules

```
legal(P,mark(X,Y)) <= true(cell(X,Y,b)) ∧
                      true(control(P))

next(cell(M,N,x)) <= does(xplayer,mark(M,N))

next(cell(M,N,W)) <= true(cell(M,N,W)) ∧ ¬W=b

terminal <= line(x) ∨ line(o)

goal(xplayer,100) <= line(x)
```

Basic Subroutines for Search

```
function legals (role, node)
   findall(X, legal(role,X), node.position ∪ gamerules)
function simulate (node,moves)
   findall(true(P), next(P), node.position ∪ moves ∪ gamerules)
function terminal (node)
   prove(terminal, node.position ∪ gamerules)
function goal (role, node)
   findone(X, goal(role,X), node.position ∪ gamerules)
```

A General Architecture



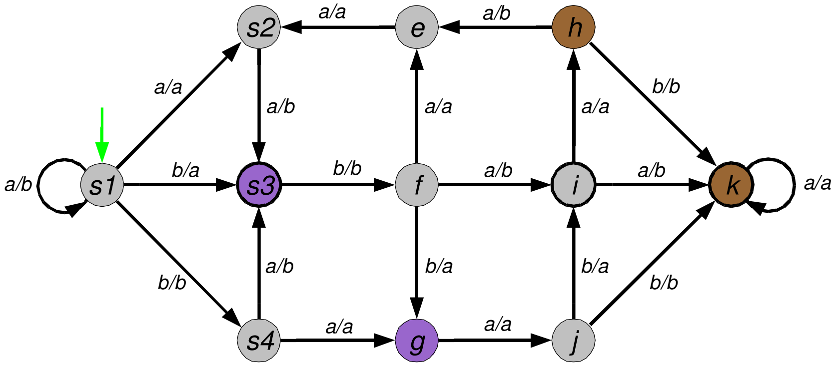
Node Expansion (Single Player Games)

```
function expand(node)
begin
   al := [ ];
   for a in legals(role,node) do
      data := simulate(node,{does(role,a)});
      new := create_node(data);
      al := {(a,new)} ∪ al
   end-for;
   return al
end
```

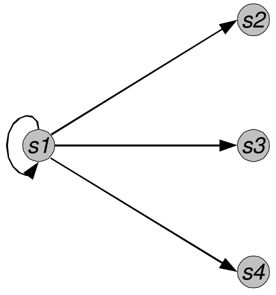
Best Move (Single Player Games)

```
function bestmove(alist)
begin
    max := 0;
    best := head(node.actionlist);
    for a in node.actionlist do
        score := maxscore(a.new.alist);
        if score = 100 then return a;
        if score > max then
            max := score; best := a
        end-if
    end-for;
    return best
end

function maxscore(alist)     % returns best score among the alist actions
```

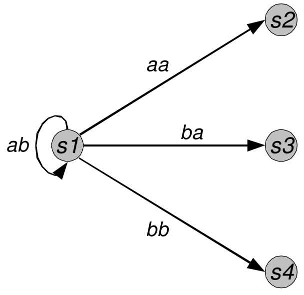
State-Space Search with Multiple Players



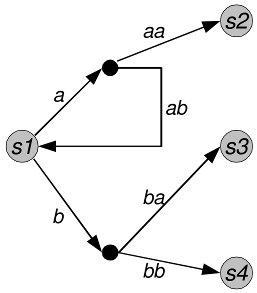
Single Player Game Graph



Multiple Player Game Graph



Bipartite Game Graph



Move Lists

Simple move list

```
[(a,s2),(b,s3)]
```

Multiple player move list

```
[([a,a],s2),([a,b],s1),
 ([b,a],s3),([b,b],s4)]
```

Bipartite move list

```
[(a,[([a,a],s2),([a,b],s1)]),
 (b,[([b,a],s3),([b,b],s4)])]
```

Multiple Player Node Expansion

```
function expand (node)
begin
    al := [ ]; jl := [ ];
    for a in legals(role,node) do
        for j in joints(role,a,node) do
            data := simulate(node,jointactions(j));
            new := create_node(data);
            jl := {(j,new)} ∪ jl
        end-for;
        al := {(a,jl)} ∪ al
    end-for;
    return al
end
```

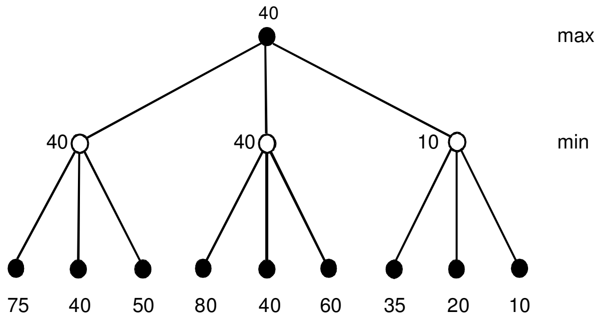
```
function joints (role,action)   % returns combinatorial list of all legal joint actions
                                % where role does action
function jointactions(j)        % returns set of does atoms for joint action j
```

Best Move

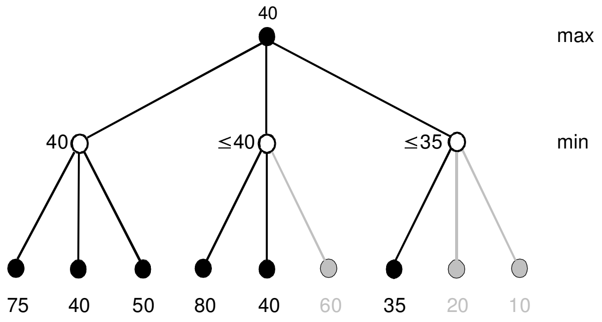
```
function bestmove (node)
begin
    max := 0;
    (best,jl) := head(node.alist);
    for (a,jl) in node.alist do
        score := minscore(jl);
        if score = 100 then return a;
        if score > max then
            max := score; best := a
        end-if
    end-for;
    return best
end
```

Note: This makes the paranoid assumption that the other players make the most harmful (for us) joint move.

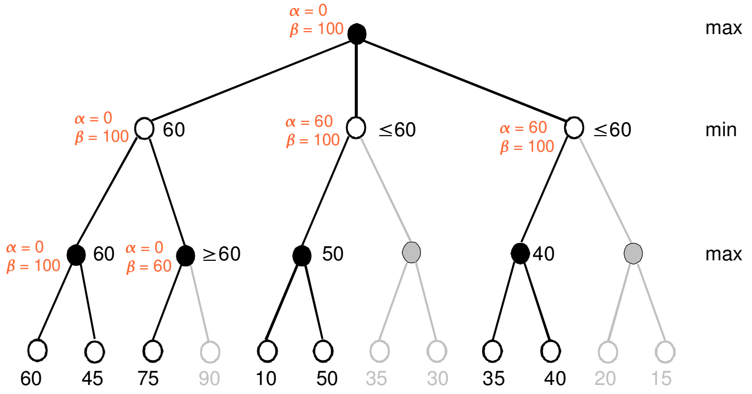
Minimax for Two-Person Zero-Sum Games



The α-β-Principle: α-Cutoffs



The α-β-Principle: α- and β-Cutoffs



State Collapse

The game tree for Tic-Tac-Toe has approximately 700,000 nodes. There are approximately 5,000 distinct states. Searching the tree requires 140 times more work than searching the graph.

Recognizing a repeat state takes time that varies with the size of the graph thus far seen. Solution: Transposition tables

Symmetry

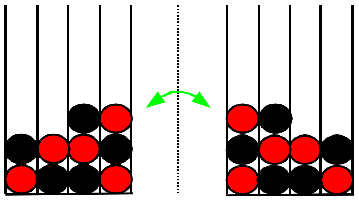
Symmetries can be logically derived from the rules of a game.

A symmetry relation over the elements of a domain is an equiva-lence relation such that

- two symmetric states are either both terminal or non-terminal
- if they are terminal, they have the same goal value
- if they are non-terminal, the legal moves in each of them are symmetric and yield symmetric states

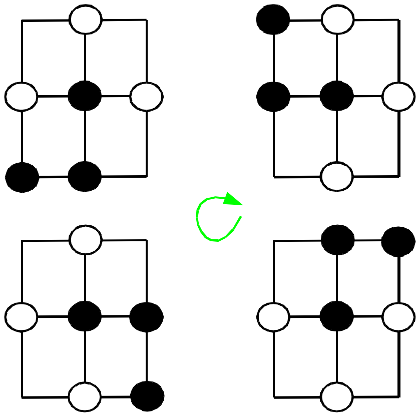
---

Reflectional Symmetry



Connect-3

Rotational Symmetry

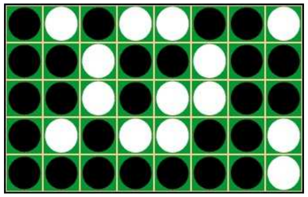


Capture Go

Factoring Example

Hodgepodge = Chess + Othello

Branching factor: a



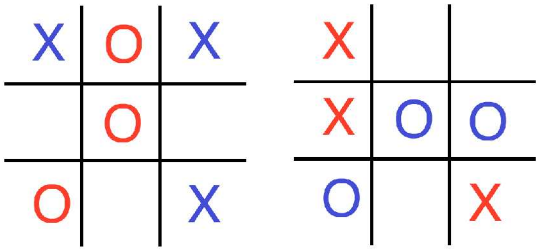
Branching factor: b

Branching factor as given to players: $a \cdot b$

Fringe of tree at depth n as given: $(a \cdot b)^n$

Fringe of tree at depth n factored: $a^n + b^n$

Double Tic-Tac-Toe



Branching factor: 81, 64, 49, 36, 25, 16, 9, 4, 1
Branching factor (factored): 9, 8, 7, 6, 5, 4, 3, 2, 1 (times 2)

Game Factoring and its Use

A set $\mathscr{F}$ of fluents and moves is a *behavioral factor* if and only if there are no connections between the fluents and moves in $\mathscr{F}$ and those outside of $\mathscr{F}$.

1. Compute factors
   - Behavioral factoring
   - Goal factoring
2. Play factors
3. Reassemble solution
   - Append plans
   - Interleave plans
   - Parallelize plans with simultaneous actions

Competition vs. Cooperation

- The “paranoid” assumption says that opponents choose the joint move that is most harmful for us.
- This is usually too pessimistic for other than zero-sum games and games with $n > 2$ players. A *rational* opponent chooses the move that's best for him rather than the one that's worst for us.
- Moreover, from a game theoretic point of view, it is incorrect to model simultaneous moves as a sequence of our move followed by the joint moves of our opponents.
  Example: Rock-Paper-Scissors

Mathematical Game Theory: Strategies

Game model:

S – set of states
$A_1, \ldots, A_n$ – n sets of actions, one for each player
$l_1, \ldots, l_n$ – where $l_i \subseteq A_i \times S$, the legality relations
$g_1, \ldots, g_n$ – where $g_i \subseteq S \times \mathbb{N}$, the goal relations

A strategy x_i for player i maps every state to a legal move for i

$x_i : S \rightarrow A_i$ (such that $(x_i(S),S) \in l_i$)

(Remark: The set of strategies is always finite in a finite game. However, there are more strategies in Chess than atoms in the universe ...)

Games in Normal Form

An *n*-player game in normal form is an n+1-tuple

$\Gamma = (X_1, \ldots, X_n, u)$

where X_i is the set of strategies for player i and

$u = (u_1, \ldots, u_n): \mathop{\times}\limits_{i=1}^{n} X_i \to \mathbb{N}^i$

are the utilities of the players for each n-tuple of strategies.

(Remark: Each n-tuple of strategies determines directly the outcome of a match, even if this consists of sequences of moves.)

Equilibria

Let $\Gamma = (X_1, \ldots, X_n, u)$ be an n-player game.

$(x_1^*, \ldots, x_n^*)$ equilibrium

if for all $i = 1, \ldots, n$ and all $x_i \in X_i$

$u_i(x_1^*, \ldots, x_{i-1}^*, x_i, x_{i+1}^*, \ldots, x_n^*) \leq u_i(x_1^*, \ldots, x_n^*)$

An equilibrium is a tuple of optimal strategies: No player has a reason to deviate from his or her strategy, given the opponent's strategies.

Dominance

A strategy $x \in X_i$ dominates a strategy $y \in X_i$ if

$u_i(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n) \geq u_i(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)$

for all $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \in X_1 \times \ldots \times X_{i-1} \times X_{i+1} \times \ldots \times X_n$.

A strategy $x \in X_i$ strongly dominates a strategy $y \in X_i$ if x dominates y and y does not dominate x.

Assume that opponents are rational:
They don't choose a strongly dominated strategy.

Dominance: Example

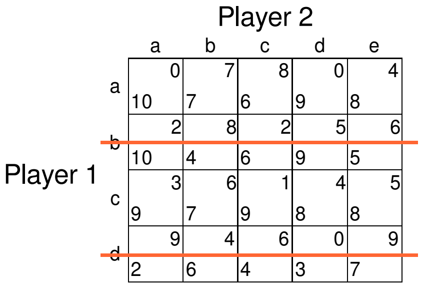
Consider a game where both players have strategies {a, b, c, d, e}. Let the goal values be given by

Player 2

Player 1

	a	b	c	d	e
a	0 10	7 7	8 6	0 9	4 8
b	2 10	8 4	2 6	5 9	6 5
c	3 9	6 7	1 9	4 8	5 8
d	9 2	4 6	6 4	0 3	9 7

Dominance: Example (ctd)

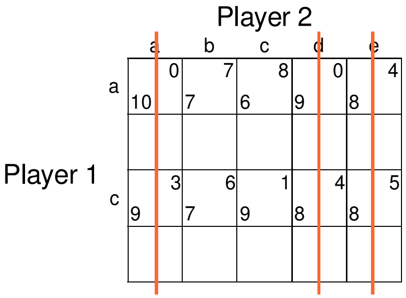


Player 2

Player 1

	a	b	c	d	e
a	0 10	7 7	8 6	0 9	4 8
b	2 10	8 4	2 6	5 9	6 5
c	3 9	6 7	1 9	4 8	5 8
d	9 2	4 6	6 4	0 3	9 7

Dominance: Example (ctd)

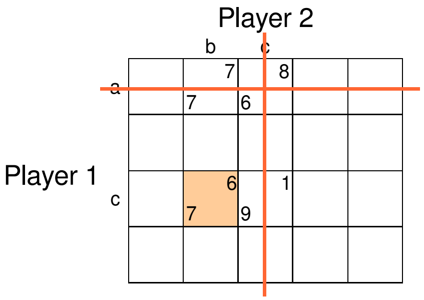


Player 2

Player 1

	a	b	c	d	e
a	0 10	7 7	8 6	0 9	4 8
c	3 9	6 7	1 9	4 8	5 8

Dominance: Example (ctd)

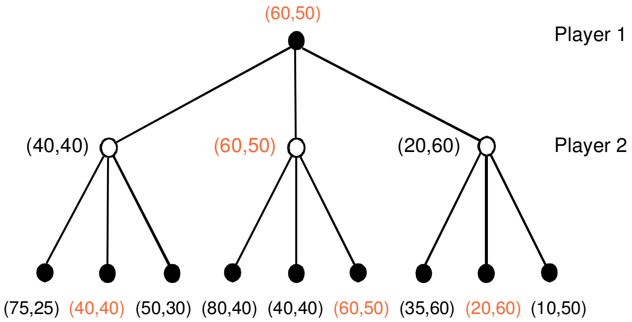


Player 2

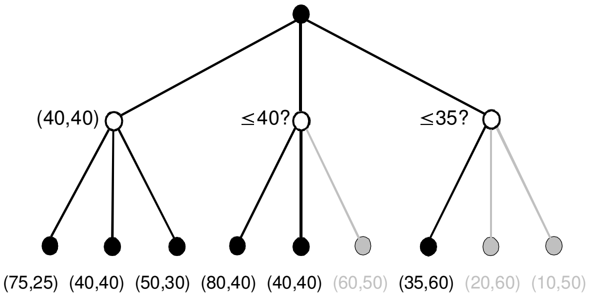
Player 1

		b	c		
a		7 7	8 6		
c		6 7	1 9		

Game Tree Search with Dominance



The α-β-Principle does not Apply



Mixed Strategies

Let $(X_1, ..., X_n, u)$ be an n-player game, then its mixed extension is

$$\Gamma = (P_1, ..., P_n, (e_1, ..., e_n))$$

where for each i=1, ..., n

$$P_i = \{p_i : p_i \text{ probability measure over } X_i\}$$

and for each $(p_1, ..., p_n) \in P_1 \times ... \times P_n$

$$e_i(p_1, ..., p_n) = \sum_{x_1 \in X_1} ... \sum_{x_n \in X_n} u_i(x_1, ..., x_n) \cdot p_1(x_1) \cdot ... \cdot p_n(x_n)$$

Nash's Theorem: Every mixed extension of an n-player game has at least one equilibrium.

Iterated Row Dominance for Mixed Strategies

Let a zero-sum game be given by

	a	b	c
a	10	0	8
b	6	4	4
c	3	8	7

Then $p_1 = \left(\frac{1}{2}, 0, \frac{1}{2}\right)$ dominates $p_1' = (0,1,0)$.

Hence, for all $(p_a', p_b', p_c') \in P_1$ with $p_b' > 0$ there exists a dominating strategy $(p_a, 0, p_c) \in P_1$.

Iterated Row Dominance for Mixed Strategies (ctd)

	a	b	c
a	10	0	8
~~b~~	~~6~~	~~4~~	~~4~~
c	3	8	7

Now $p_2 = \left(\frac{1}{2}, \frac{1}{2}, 0\right)$ dominates $p_2' = (0,0,1)$.

Iterated Row Dominance for Mixed Strategies (ctd)

	a	b	~~c~~
a	10	0	~~8~~
c	3	8	~~7~~

The unique equilibrium is $\left(\left(\frac{1}{3},0,\frac{2}{3}\right),\left(\frac{1}{2},\frac{1}{2},0\right)\right)$.

Learning

Roadmap

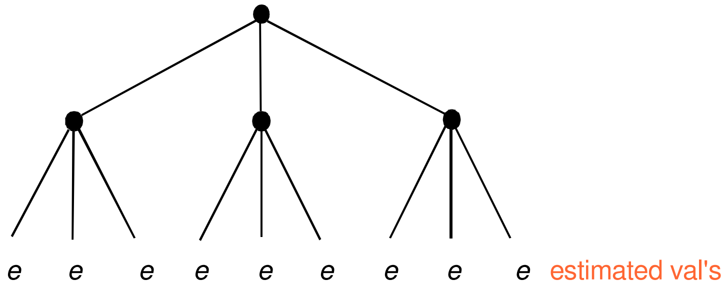
- Heuristics
- Detecting Structures
- Generating Evaluation Functions
- The Viking Method

Complete vs. Incomplete Search

Simple games like Tic-Tac-Toe and Rock-Paper-Scissors can be searched completely.

"Real" games like Peg Jumping, Chinese Checkers, Chess cannot.

Incomplete Search



Requires to automatically generate evaluation functions

Towards Good Play

Besides efficient inference and search algorithms, the ability to automatically generate a good evaluation function distinguishes good from bad General Game Playing programs.

Existing approaches:

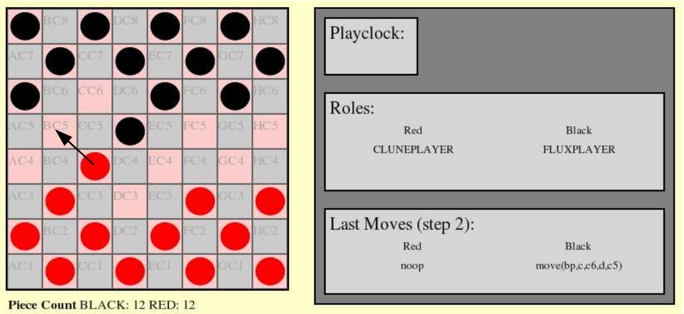
- Mobility and Novelty Heuristics
- Structure Detection
- Fuzzy Goal Evaluation
- The Viking Method: Monte-Carlo Tree Search

Constructing an Evaluation Function

Mobility

- More moves means better state
- Advantage:
  In many games, being cornered or forced into making a move is quite bad
  - In Chess, having fewer moves means having fewer pieces, pieces of lower value, or less control of the board
  - In Chess, when you are in check, you can do relatively few things compared to not being in check
  - In Othello, having few moves means you have little control of the board
- Disadvantage: Mobility is bad for some games

Worldcup 2006: Cluneplayer vs. Fluxplayer



Playclock:

Roles:

Red CLUNEPLAYER

Black FLUXPLAYER

Last Moves (step 2):

Red noop

Black move(bp,c,c6,d,c5)

Piece Count BLACK: 12 RED: 12

Inverse Mobility

- Having fewer things to do is better
- This works in some games, like Nothello and Suicide Chess, where you might in fact want to lose pieces
- How to decide between mobility and inverse mobility heuristics?

Novelty

- Changing the game state is better
- Advantage:
  - Changing things as much as possible can help avoid getting stuck
  - When it is unclear what to do, maybe the best thing is to throw in some directed randomness
- Disadvantage:
  - Changing the game state can happen if you throw away your own pieces ...
  - Unclear if novelty per se actually goes anywhere useful for anybody

---

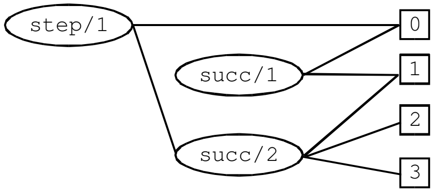
Designing Evaluation Functions

- Typically designed by programmers/humans
- A great deal of thought and empirical testing goes into choosing one or more good functions
- E.g.
  - piece count, piece values in chess
  - holding corners in Othello
- But this requires knowledge of the game's structure, semantics, play order, etc.

Identifying Domains

- Domains of fluents identified by dependency graph

```
succ(0,1) ∧ succ(1,2) ∧ succ(2,3)
init(step(0))
next(step(X)) <=  true(step(Y)) ∧ succ(Y,X)
```



Identifying Structures: Relations

A successor relation is a binary relation that is antisymmetric, functional, and injective.

Example:

```
 succ(1,2) ∧ succ(2,3) ∧ succ(3,4) ∧ ...
 next(a,b) ∧ next(b,c) ∧ next(c,d) ∧ ...
```

An order relation is a binary relation that is antisymmetric and transitive.

Example:

```
lessthan(A,B) <= succ(A,B)
lessthan(A,C) <= succ(A,B) ∧ lessthan(B,C)
```

Boards and Pieces

An (*m*-dimensional) board is an *n*-ary fluent ($n \geq m+1$) with

- *m* arguments whose domains are successor relations
- 1 output argument

Example:

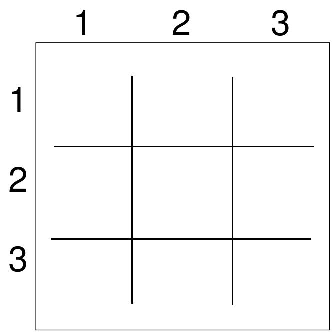
```
cell(a,1,whiterook) ∧ cell(b,1,whiteknight) ∧ ...
```

A marker is an element of the domain of a board's output argument.

A piece is a marker which is in at most one board cell at a time.

Example: Pebbles in Othello, White King in Chess

Fuzzy Goal Evaluation: Example



```
goal(xplayer,100)  <=
      line(x)
line(P)  <= row(P)
            ∨ col(P)
            ∨ diag(P)
```

Value of intermediate state = Degree to which it satisfies the goal

Full Goal Specification

```
goal(xplayer,100) <= line(x)

line(P)     <= row(P)  ∨  col(P)  ∨  diag(P)

row(P)      <= true(cell(1,Y,P))  ∧  true(cell(2,Y,P))  ∧
               true(cell(3,Y,P))
col(P)      <= true(cell(X,1,P))  ∧  true(cell(X,2,P))  ∧
               true(cell(X,3,P))
diag(P)     <= true(cell(1,1,P))  ∧  true(cell(2,2,P))  ∧
               true(cell(3,3,P))
diag(P)     <= true(cell(3,1,P))  ∧  true(cell(2,2,P))  ∧
               true(cell(1,3,P))
```

After Unfolding

```
goal(x,100)  <= true(cell(1,Y,x)) ∧ true(cell(2,Y,x)) ∧
                true(cell(3,Y,x))
                ∨
                true(cell(X,1,x)) ∧ true(cell(X,2,x)) ∧
                true(cell(X,3,x))
                ∨
                true(cell(1,1,x)) ∧ true(cell(2,2,x)) ∧
                true(cell(3,3,x))
                ∨
                true(cell(3,1,x)) ∧ true(cell(2,2,x)) ∧
                true(cell(1,3,x))
```

3 literals are true after `does(x,mark(1,1))`
2 literals are true after `does(x,mark(1,2))`
4 literals are true after `does(x,mark(2,2))`

Evaluating Goal Formula (Cont'd)

- Our t-norms: Instances of the Yager family (with parameter *q*)

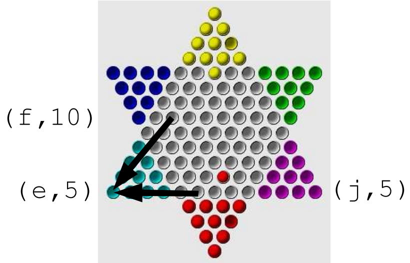
  $T(a,b) = 1 - S(1-a,1-b)$
  $S(a,b) = (a^q + b^q) ^ (1/q)$

- Evaluation function for formulas

  $eval(f \wedge g) = T'(eval(f),eval(g))$
  $eval(f \vee g) = S'(eval(f),eval(g))$
  $eval(\neg f) = 1 - eval(f)$

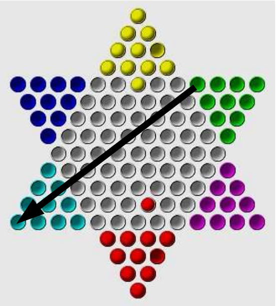
Degree to which *f(x,a)* is true given that *f(x,b)* holds:

$$(1-p) - (1-p) * \Delta(b,a) / |dom(f(x))|$$



With *p*=0.9, eval(`cell(green,e,5)`) is
0.082 if `true(cell(green,f,10))`
0.085 if `true(cell(green,j,5))`

Advanced Fuzzy Goal Evaluation: Example



```
init(cell(green,j,13)) ∧ ...

goal(green,100)
    <= true(cell(green,e,5))
       ∧ ...
```

Truth degree of goal literal = (Distance to current value)$^{-1}$

Identifying Metrics

- Order relations Binary, antisymmetric, functional, injective

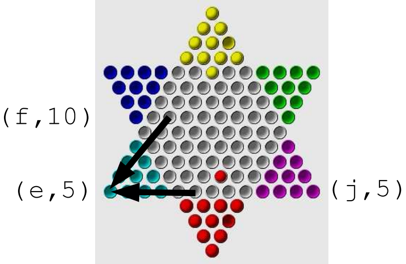
```
succ(1,2).  succ(2,3).  succ(3,4).
file(a,b).  file(b,c).  file(c,d).
```

- Order relations define a metric on functional features

`Δ(cell(green,j,13),cell(green,e,5)) = 13`

Degree to which *f(x,a)* is true given that *f(x,b)*:

$(1-p) - (1-p) * \Delta(b,a) / |dom(f(x))|$

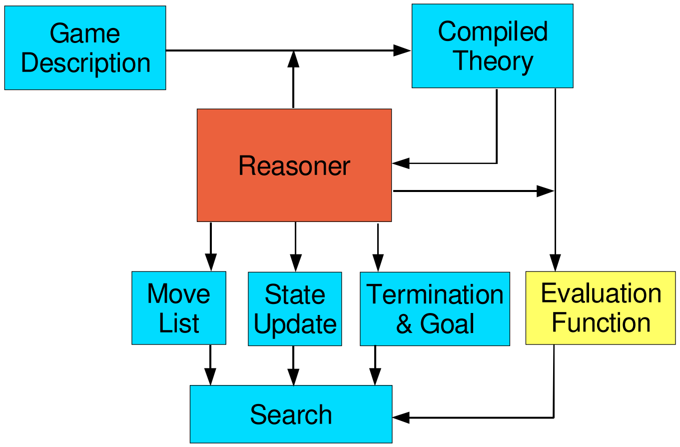


With p=0.9, eval(`cell(green,e,5)`) is

0.082 if `true(cell(green,f,10))`

0.085 if `true(cell(green,j,5))`

A General Architecture



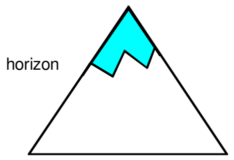
Assessment

Fuzzy goal evaluation works particularly well for games with

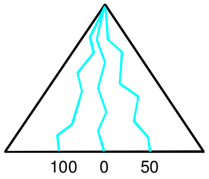
- independent sub-goals
  15-Puzzle
- converge to the goal
  Chinese Checkers
- quantitative goal
  Othello
- partial goals
  Peg Jumping, Chinese Checkers with >2 players

An Alternative Approach: The Viking Method

- aka Monte Carlo Tree Search
- used by Cadiaplayer (Reykjavik University)



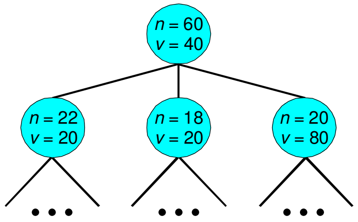
Game Tree Seach



MC Tree Search

Monte Carlo Tree Search

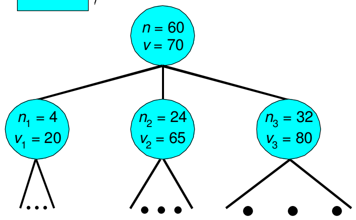
Value of move = Average score returned by simulation



Confidence Bounds

- Play one random game for each move
- For next simulation choose move

$$argmax_i \left(v_i + C * \boxed{\sqrt{\frac{\log n}{n_i}}} \right)$$ confidence bound

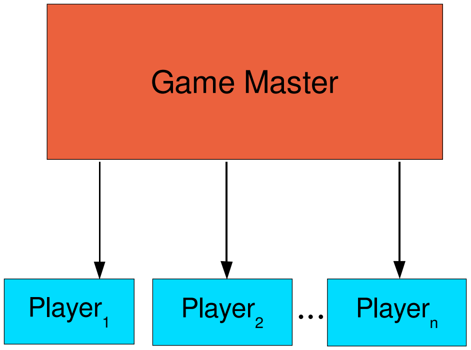


Assessment

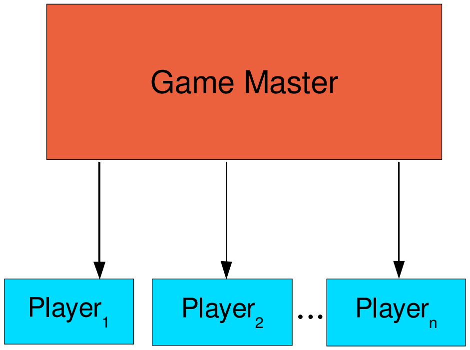
Monte Carlo Tree Search works particularly well for games which

- converge to the goal
  Checkers
- reward greedy behavior
- have a large branching factor
- do not admit a good heuristics

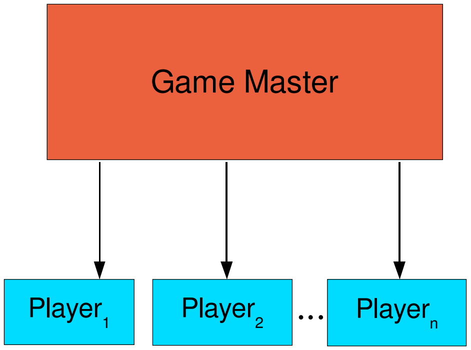
The World Cup



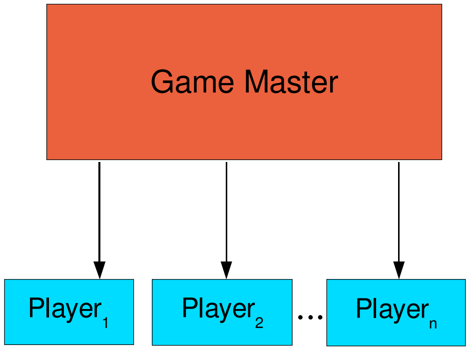
Game description
Time to think: 1,800 sec
Time per move: 45 sec
Your role



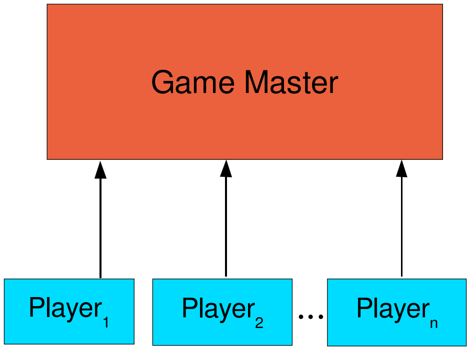
Game description
Time to think: 1,800 sec
Time per move: 45 sec
Your role



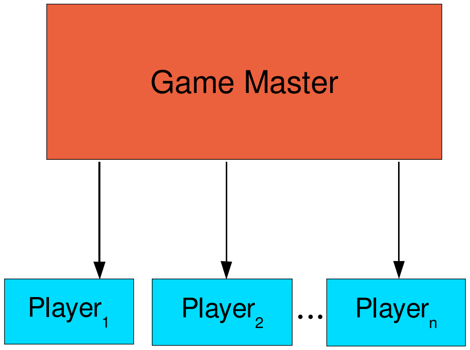
Start



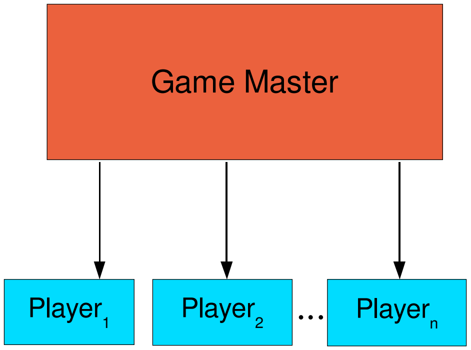
Your move, please



Individual moves



Joint move



End of game

1st World Championship 2005 in Pittsburgh

1. UCLA (Clune)
2. Florida
3. Fluxplayer
   UT Austin

2nd World Championship 2006 in Boston: Final Leaderboard

Player	Points
1. Fluxplayer	2690.75
2. UCLA	2573.75
3. UT Austin	2370.50
4. Florida	1948.25
5. TU Dresden II	1575.00
⋮	

3rd World Championship 2007 in Vancouver: Final Leaderboard

Player	Points
1. Reykjavik	2724
2. Fluxplayer	2356
3. Paris	2253
4. UCLA	2122
5. UT Austin	1798
⋮	

Summary

The GGP Challenge

Much like RoboCup, General Game Playing

- combines a variety of AI areas
- fosters developmental research
- has great public appeal
- has the potential to significantly advance AI

In contrast to RoboCup, GGP has the advantage to

- focus on high-level intelligence
- have low entry cost
- make a great hands-on course for AI students

A Vision for GGP

Uncertainty

- Nondeterministic games with incomplete information

Natural Language Understanding

- Rules of a game given in natural language

Computer Vision

- Vision system sees board, pieces, cards, rule book, ...

Robotics

- Robot playing the actual, physical game

Resources

Stanford GGP initiative - GDL specification - Basic player	`games.stanford.edu`
GGP in Germany - Game master	`general-game-playing.de`
Palamedes - GGP/GDL development tool	`palamedes-ide.sourceforge.net`

Recommended Papers

- J. Clune
  Heuristic evaluation functions for general game playing
  AAAI 2007
- H. Finnsson, Y. Björnsson
  Simulation-based approach to general game playing
  AAAI 2008
- M. Genesereth, N. Love, B. Pell
  General game playing
  AI magazine 26(2), 2006
- G. Kuhlmann, K. Dresner, P. Stone
  Automatic heuristic construction in a complete general game player
  AAAI 2006
- S. Schiffel, M. Thielscher
  Fluxplayer: a successful general game player
  AAAI 2007