IJCAI'09 Tutorial

New Trends in
General Game Playing

Michael Thielscher, Dresden
Chess Players
The 1st Chess Computer ("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.

But Deep Blue is a highly specialized system—it can't even play a decent game of Tic-Tac-Toe or Rock-Paper-Scissors!

A General Game Player is a system that

- understands formal descriptions of arbitrary games
- learns to play these games well without human intervention
Rather than being concerned with a specialized solution to a narrow problem, General Game Playing encompasses a variety of AI areas:

- AI Game Playing
- Knowledge Representation and Reasoning
- Search, Planning
- Learning
- ... and more!
General Game Playing and AI

<table>
<thead>
<tr>
<th>Agents</th>
<th>Games</th>
</tr>
</thead>
<tbody>
<tr>
<td>Competitive environments</td>
<td>Deterministic, complete information</td>
</tr>
<tr>
<td>Uncertain environments</td>
<td>Nondeterministic, partially observable</td>
</tr>
<tr>
<td>Unknown environment model</td>
<td>Rules partially unknown</td>
</tr>
<tr>
<td>Real-world environments</td>
<td>Robotic player</td>
</tr>
</tbody>
</table>

|
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.
A **General Agent** is a system that

- understands formal descriptions of arbitrary multiagent environments
- learns to function in this environment without human intervention

**Examples**

- Rules of e-marketplaces made accessible to agents
- Internet platforms that are formally described
- Providing details in agent competitions (eg, TAC) at runtime
Example Games
Single-Player, Deterministic
Demo: Single-Player, Deterministic
Two-Player, Zero-Sum, Deterministic
Two-Player, Zero-Sum, Deterministic
Two-Player, Zero-Sum, Nondeterministic
Two-Player, Simultaneous Moves
$n$-Player, Incomplete Information, Nondeterministic
The History of General Game Playing

- 2005  1st AAAI General Game Playing Competition
- 2006  First publications on General Game Playing
- 2009  1st General Game Playing Workshop (GIGA'09)

- Research groups world-wide: Austin, Bremen, Dresden, Edmonton, Liverpool, Paris, Potsdam, Reykjavik, Sydney
Roadmap: New Trends in GGP

- Description Languages
- Reasoning about Game Descriptions
- Generating Evaluation Functions
- Learning by Simulation
Description Languages
Description Languages: Overview

- The technology of General Agents requires languages to describe the rules that govern an environment

- Descriptions
  - should be easy to understand and maintain
  - can be fully automatically processed by a computer
  - must have a precise semantics

- Examples
  - Game Description Language GDL
  - Market Specification Language MSL
Every finite game can be modeled as a state transition system

But direct encoding impossible in practice

19,683 states

$\sim 10^{46}$ legal positions
Modular State Representation: Features

\[ \text{cell}(X,Y,C) \]
\[ X \in \{a, \ldots, h\} \]
\[ Y \in \{1, \ldots, 8\} \]
\[ C \in \{\text{whiteKing}, \ldots, \text{blank}\} \]

\[ \text{control}(P) \]
\[ P \in \{\text{white}, \text{black}\} \]
canCastle(P, S)
P ∈ \{white, black\}
S ∈ \{kingsSide, queensSide\}

enPassant(C)
C ∈ \{a, ... , h\}
Moves

move(U, V, X, Y)
U, X ∈ {a, ..., h}
V, Y ∈ {1, ..., 8}

promote(X, Y, P)
X, Y ∈ {a, ..., h}
P ∈ {whiteQueen, ...}
Based on the features and moves of a game, the rules can be described in **formal logic** using a few standard predicate symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role(P)</td>
<td>P is a player</td>
</tr>
<tr>
<td>init(F)</td>
<td>F holds in the initial position</td>
</tr>
<tr>
<td>true(F)</td>
<td>F holds in the current position</td>
</tr>
<tr>
<td>legal(P,M)</td>
<td>player P has legal move M</td>
</tr>
<tr>
<td>does(P,M)</td>
<td>player P does move M</td>
</tr>
<tr>
<td>next(F)</td>
<td>F holds in the next position</td>
</tr>
<tr>
<td>terminal</td>
<td>the current position is terminal</td>
</tr>
<tr>
<td>goal(P,N)</td>
<td>player P gets reward N in current position</td>
</tr>
</tbody>
</table>
Elements of a Game Description (1)

- **Players**
  
  \[
  \text{role(white)} \leq= \\
  \text{role(black)} \leq= \\
  \]

- **Initial position**
  
  \[
  \text{init(cell(a,1,whiteRook))} \leq= \\
  \ldots \\
  \]

- **Moves**
  
  \[
  \text{legal(white,promote(X,Y,P))} \leq= \\
  \text{true(cell(X,7,whitePawn))} \land \ldots \\
  \]
Elements of a Game Description (2)

- **Moves: Update**
  \[
  \text{next}(\text{cell}(X,Y,C)) \Leftarrow \text{does}(P, \text{move}(U,V,X,Y)) \wedge \text{true}(\text{cell}(U,V,C))
  \]

- **End of game**
  \[
  \text{terminal} \Leftarrow \text{checkmate} \vee \text{stalemate}
  \]

- **Result**
  \[
  \text{goal}(\text{white}, 100) \Leftarrow \text{checkmate} \wedge \text{true}(\text{control}(\text{black}))
  \]
  \[
  \text{goal}(\text{white}, 50) \Leftarrow \text{stalemate}
  \]
A Complete Formalization of Tic-Tac-Toe (1/3)

role(xplayer) <= role(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)) <=

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))
Rules of Tic-Tac-Toe (2/3)

\[
\begin{align*}
\text{next(cell}(M,N,x)) & \iff \text{does(xplayer,mark}(M,N)) \\
\text{next(cell}(M,N,o)) & \iff \text{does(oplayer,mark}(M,N)) \\
\text{next(cell}(M,N,W)) & \iff \text{true(cell}(M,N,W)) \land \text{does(P,mark}(J,K)) \land (\neg M=J \lor \neg N=K) \\
\text{next(control}(xplayer)) & \iff \text{true(control}(oplayer)) \\
\text{next(control}(oplayer)) & \iff \text{true(control}(xplayer)) \\
\text{terminal} & \iff \text{line}(x) \lor \text{line}(o) \lor \neg \text{open} \\
\text{line}(W) & \iff \text{row}(M,W) \lor \text{column}(M,W) \lor \text{diagonal}(M,W) \\
\text{open} & \iff \text{true(cell}(M,N,b))
\end{align*}
\]
Rules of Tic-Tac-Toe (3/3)

\[
\begin{align*}
goal(xplayer,100) & \leq line(x) \\
goal(xplayer,50) & \leq \neg line(x) \land \neg line(o) \land \neg open \\
goal(xplayer,0) & \leq line(o) \\
goal(oplayer,100) & \leq line(o) \\
goal(oplayer,50) & \leq \neg line(x) \land \neg line(o) \land \neg open \\
goal(oplayer,0) & \leq line(x)
\end{align*}
\]

\[
\begin{align*}
row(M,W) & \leq \\
& \quad true(cell(M,1,W)) \land true(cell(M,2,W)) \land true(cell(M,3,W))
\end{align*}
\]

\[
\begin{align*}
column(N,W) & \leq \\
& \quad true(cell(1,N,W)) \land true(cell(2,N,W)) \land true(cell(3,N,W))
\end{align*}
\]

\[
\begin{align*}
diagonal(W) & \leq \\
& \quad true(cell(1,1,W)) \land true(cell(2,2,W)) \land true(cell(3,3,W)) \\
& \quad \lor true(cell(1,3,W)) \land true(cell(2,2,W)) \land true(cell(3,1,W))
\end{align*}
\]
Properties of GDL

- GDL rules are logic programs, including the use of negation-as-failure

- Additional, syntactic restrictions ensure that all relevant derivations are finite

- The language is completely knowledge-free: symbols like cell and control acquire meaning only through the rules

- To make this clear, GDL descriptions are often obfuscated

For details see [Genesereth, Love & Pell, 2006]
Obfuscated Rules: How the Computer Sees a Game Description

\[
\text{next}(\text{thuis}(M,N,\text{een})) \leq \text{does}(\text{jij}, \text{huur}(M,N))
\]
\[
\text{next}(\text{thuis}(M,N,\text{het})) \leq \text{does}(\text{wij}, \text{huur}(M,N))
\]
\[
\text{next}(\text{fiets}(\text{jij})) \leq \text{true}(\text{fiets}(\text{wij}))
\]
\[
\text{next}(\text{fiets}(\text{wij})) \leq \text{true}(\text{fiets}(\text{jij}))
\]
\[
\text{terminal} \leq \text{brommer}(\text{een}) \lor \text{brommer}(\text{het}) \lor \neg \text{keer}
\]
\[
\text{brommer}(W) \leq \text{gaag}(M,W) \lor \text{daag}(M,W) \lor \text{naar}(M,W)
\]
\[
\ldots
\]
Semantics: Games as State Machines

![Graph]

- States: a, b, c, d, e, f, g, h, i, j, k
- Edges: a/a, a/b, b/b
- Arrows indicate transitions from one state to another based on input symbols.

- Initial state: a
- Final state: k

The diagram illustrates the state transitions of a game as a state machine, where each state represents a point in the game, and the edges represent possible transitions based on input 'a' or 'b'.
A game is a structure with the following components:

\( R \) – set of players
\( S \) – set of states
\( A \) – set of moves

\( \mathcal{L} \subseteq R \times A \times S \) – the legality relation
\( u: M \times S \rightarrow S \) – the update function, for joint moves \( m: R \rightarrow A \)

\( s_i \in S \) – initial game state
\( t \subseteq S \) – terminal states
\( g \subseteq R \times S \times \mathbb{N} \) – the goal relation
A GDL description $P$ encodes\

$$s_1 = \{ f : P \models \text{init}(f) \}$$

\begin{align*}
\text{init}(\text{cell}(1,1,b)) & \leq \\
\text{init}(\text{cell}(1,2,b)) & \leq \\
\ldots & \\
\text{init}(\text{cell}(3,3,b)) & \leq \\
\text{init}(\text{control}(\text{xplayer})) & \leq \\
\end{align*}
Let \( S^{\text{true}} := \{ \text{true}(f) : f \in S \} \).

Then \( P \) encodes

\[
I = \{ (r \in R, a, S) : P \cup S^{\text{true}} \models \text{legal}(r, a) \}
\]

\[
\text{legal}(P, \text{mark}(X, Y)) \Leftarrow \text{true}(\text{cell}(X, Y, b)) \land \text{true}(\text{control}(P))
\]

...
From the Rules to the Game Model: Update Function

Let \( m^{\text{does}} := \{ \text{does}(r,m(r)) : r \in R \} \).

Then \( P \) encodes

\[
u(m,S) = \{ f : P \cup S^{\text{true}} \cup m^{\text{does}} \models \text{next}(f) \}\]

\[
\text{next}(\text{cell}(M,N,x)) \leq \text{does}(\text{xplayer},\text{mark}(M,N))
\]
\[
\text{next}(\text{cell}(M,N,o)) \leq \text{does}(\text{oplayer},\text{mark}(M,N))
\]
\[
\text{next}(\text{cell}(M,N,W)) \leq \text{true}(\text{cell}(M,N,W)) \land \\
\text{does}(P,\text{mark}(J,K)) \land (\neg M=J \lor \neg N=K)
\]

\ ...

For details see [Schiffel & Thielscher, 2009a]
A Basic Player

- Game Description
- Compiled Theory
- Reasoner
  - Move List
  - State Update
  - Termination & Goal
- Search
Actual Game Play

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

Player_1  Player_2  ...  Player_n
Actual Game Play

Game Master

Player_1 → Player_2 → ... → Player_n → Start
Actual Game Play

Game Master

Player$_1$  Player$_2$  ...  Player$_n$

Your move, please
Actual Game Play

Game Master

Player$_1$

Player$_2$

... Player$_n$

Individual moves
Actual Game Play

Game Master

Joint move

Player_1 → Player_2 → ... → Player_n
Actual Game Play

Game Master

Player_1  Player_2  ...  Player_n

End of game
Demo: Bidding Tic-Tac-Toe
Towards Other Description Languages

- The GGP principle can be transferred to other areas

- A General Trading Agent is a system that
  - understands the rules of unknown market places
  - learns how to participate without human intervention

- A specification language for markets must account for
  - information asymmetry
  - asynchronous actions

  → introduce market maker + private message passing
Market Specification Language MDL

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>trader(A)</td>
<td>A is a trader</td>
</tr>
<tr>
<td>message(A,M)</td>
<td>trader A can send message M</td>
</tr>
<tr>
<td>init(F)</td>
<td>F holds in the initial state</td>
</tr>
<tr>
<td>true(F)</td>
<td>F holds in the current state</td>
</tr>
<tr>
<td>next(F)</td>
<td>F holds in the next state</td>
</tr>
<tr>
<td>legal(A)</td>
<td>market maker can do action A</td>
</tr>
<tr>
<td>does(A)</td>
<td>market maker does action A</td>
</tr>
<tr>
<td>receive(A,M)</td>
<td>receiving message M from trader A</td>
</tr>
<tr>
<td>send(A,M)</td>
<td>sending message M to trader A</td>
</tr>
<tr>
<td>time(T)</td>
<td>T is the current time</td>
</tr>
<tr>
<td>terminal</td>
<td>the market is closed</td>
</tr>
</tbody>
</table>

For details see [Thielscher & Zhang, 2009]
Example: Sealed-Bid Auction

\[ \text{trader}(a_1) \leq \]
\[
\ldots
\]
\[ \text{trader}(a_n) \leq \]
\[ \text{message}(A, \text{my\_bid}(P)) \leq \text{trader}(A) \land P \geq 0 \]

\[ \text{next}(\text{bid}(A,P)) \leq \text{accept}(\text{bid}(A,P)) \]
\[ \text{accept}(\text{bid}(A,P)) \leq \text{receive}(A, \text{my\_bid}(P)) \land \text{time}(1) \]
\[ \text{bestbid}(A,P) \leq \text{true}(\text{bid}(A,P)) \land \neg \text{outbid}(P) \]
\[ \text{outbid}(P) \leq \text{true}(\text{bid}(A,P1)) \land P1 > P \]

\[ \text{legal}(\text{clearing}(A,P)) \leq \text{bestbid}(A,P) \land \text{time}(2) \]

\[ \text{send}(A, \text{bid\_accepted}(P)) \leq \text{accept}(\text{bid}(A,P)) \]
\[ \text{send}(A, \text{winner}(A1,P)) \leq \text{trader}(A) \land \text{does}(\text{clearing}(A1,P)) \]
\[ \text{terminal} \leq \text{time}(3) \]
Reasoning about Game Descriptions
The Value of Knowledge

Knowledge-based players try to extract and prove useful knowledge about a game from the mere rules

Some examples of potentially useful game-specific knowledge

- The game is strictly turn-based
- Each board cell \((X, Y)\) has a unique contents \(M\)
- Markers \(\times\) and \(\circ\) in Tic-Tac-Toe are permanent

Players systematically search for such properties and use them, eg. to improve their search or to generate an evaluation function
How to Verify Game-Specific Properties

- One approach is to run a number of random games and see if the property never gets violated.

- More reliable—and often even more efficient—is to actually prove that the game rules entail the property.

- Proof by induction: the property holds initially, and whenever it is true it also holds after a legal joint move.
Induction Proofs: Example

Claim
Fluent control has a unique argument in every reachable position

\[
\begin{align*}
P : & \quad \text{init}(\text{control}(\text{xplayer})) \leq \text{next}(\text{control}(\text{xplayer})) \leq \text{true}(\text{control}(\text{oplayer})) \\
& \quad \text{next}(\text{control}(\text{oplayer})) \leq \text{true}(\text{control}(\text{xplayer}))
\end{align*}
\]

The claim holds if \( P \) implies that
- uniqueness holds init; and
- uniqueness holds next, provided it is true (and every player makes a legal move)
Induction Proofs by Answer Set Programming

ASP is an established method to compute models of logic programs. Efficient off-the-shelf implementations can be used. Proof by contradiction: claim follows if its negation admits no model.

\[ P \cup h_0 <= 1\{\text{init(control}(X)) : \text{control}_\text{dom}(X)\}1 <= h_0 \]

admits no answer set; same for

\[ P \cup 1\{\text{true(control}(X)) : \text{control}_\text{dom}(X)\}1 <= h <= 1\{\text{next(control}(X)) : \text{control}_\text{dom}(X)\}1 <= h \]
Another Example

Claim

Every board cell has a unique contents

Let $\mathcal{P}$ be the GDL rules for Tic-Tac-Toe.

\[
\mathcal{P} \cup h0(X,Y) <= \{\text{init(cell}(X,Y,Z)) : c\_dom(Z)\} \leq h0 \leq \neg h0(X,Y) \leq \neg h0
\]

admits no answer set
Another Example (cont'd)

For the induction step, uniqueness of control must be known!

\[
\begin{align*}
P \cup 1\{true(\text{control}(X)) : \text{control\_dom}(X)\} & \leq \\ \\ \\ \ \\ \ \\ \ \\ \ \\ \ \\ \ \\ \ \text{1\{does}(R,A) : \text{move\_dom}(A)\}\text{1} & \leq \text{1\{does}(R,A) \land \neg \text{legal}(R,A) & \\
1\{true(\text{cell}(X,Y,Z)) : \text{c\_dom}(Z)\} & \leq \text{h}(X,Y) & \leq 1\{\text{next}(\text{cell}(X,Y,Z)) : \text{c\_dom}(Z)\}\text{1} & \text{h} & \leq \neg \text{h}(X,Y) & \leq \neg \text{h} & \\
\end{align*}
\]

admits no answer set.

For details see [Schiffel & Thielscher, 2009b]
General Search Techniques for Games

- Single-player games: iterative deepening, non-uniform, ie. nodes with high estimated values searched deeper
- Transposition tables to store (position, evaluation)-pairs
- Two-player, zero-sum games with alternating moves: standard minimax with $\alpha$-$\beta$-cutoffs
- Simultaneous moves, non-zero sum, $n$-player games:
  - paranoid search (opponents choose worst move for us)
  - computing equilibria (game theory)
Using Knowledge for Search: Symmetry

Symmetries can be logically derived from the rules of a game

A *symmetry relation* over the elements of a domain is an equivalence 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
Using Knowledge for Search: Factoring

Hodgepodge = Chess + Othello

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 $n$ if factored: $a^n + b^n$
Double Tic-Tac-Toe

Branching factor: 81, 64, 49, 36, 25, 16, 9, 4, 1
Branching factor (after factoring): 18, 16, 14, 12, 10, 8, 6, 4, 1
Generating Evaluation Functions
Automatically Generated Evaluation Functions

Besides efficient inference and search algorithms, the ability to automatically generate a good evaluation function distinguishes good from bad general game players.

Approaches

- General heuristics: Mobility heuristics, Novelty heuristics, ...
- Recognizing structures: boards, pieces, piece values, ...
- Fuzzy Goal Evaluation
Mobility Heuristics

- **Idea**
  More moves means better state

- **Advantage**
  Often, being cornered or forced into making a move is quite bad
  - In Chess, having fewer moves means having fewer pieces or pieces of lower value
  - In Othello, having few moves means you have little control of the board

- **Disadvantage**
  Mobility is bad for some games
Example: Worldcup 2006 Final

Checkers (on a cylindrical board) with standard “forced capture“ rule
Novelty Heuristics

• **Idea**
  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 controlled randomness

• **Disadvantage**
  - Game state can also change if you just throw away own pieces
  - Unclear if novelty per se actually goes anywhere useful
Identifying Structures: Domains

- Domains of fluents identified by dependency graph

\[
\begin{align*}
succ(0,1) & \land succ(1,2) & \land succ(2,3) \\
init(\text{step}(0)) & \\
next(\text{step}(X)) & \leq true(\text{step}(Y)) & \land succ(Y,X)
\end{align*}
\]
Identifying Structures: Relations

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

**Example**

\[
\begin{align*}
\text{succ}(1,2) & \land \text{succ}(2,3) & \land \text{succ}(3,4) & \land \ldots \\
\text{next}(a,b) & \land \text{next}(b,c) & \land \text{next}(c,d) & \land \ldots
\end{align*}
\]

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

**Example**

\[
\begin{align*}
\text{lessthan}(A,B) & \leq \text{succ}(A,B) \\
\text{lessthan}(A,C) & \leq \text{succ}(A,B) \land \text{lessthan}(B,C)
\end{align*}
\]
Boards and Pieces

An \((m\text{-dimensional})\) board is an \(n\)-ary fluent \((n \geq m+1)\) with

- \(m\) arguments whose domains are successor relations
- 1 output argument

Example

\[
\text{cell}(a,1,\text{whiterook}) \land \text{cell}(b,1,\text{whiteknight}) \land \ldots
\]

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

For details see [Clune, 2007]
Fuzzy Goal Evaluation: Example

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

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

line(P) <= row(P) V column(P) V diagonal(P)

row(P) <= true(cell(1,Y,P)) \land true(cell(2,Y,P)) \land true(cell(3,Y,P))

column(P) <= true(cell(X,1,P)) \land true(cell(X,2,P)) \land true(cell(X,3,P))

diagonal(P) <= true(cell(1,1,P)) \land true(cell(2,2,P)) \land true(cell(3,3,P))
\lor 
true(cell(3,1,P)) \land true(cell(2,2,P)) \land true(cell(1,3,P))
After Unfolding

goal(xplayer, 100)

<= true(cell(1, Y, x)) \land true(cell(2, Y, x)) \land true(cell(3, Y, x))

\lor

true(cell(X, 1, x)) \land true(cell(X, 2, x)) \land true(cell(X, 3, x))

\lor

true(cell(1, 1, x)) \land true(cell(2, 2, x)) \land true(cell(3, 3, x))

\lor

true(cell(3, 1, x)) \land true(cell(2, 2, x)) \land 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))
Fuzzy Goal Evaluation

- Use t-norms, eg. 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
  
  $eval(f \land g) = T(eval(f), eval(g))$
  $eval(f \lor g) = S(eval(f), eval(g))$
  $eval(\neg f) = 1 - eval(f)$
Advanced Fuzzy Goal Evaluation: Example

init(cell(green,j,13)) ∧ ...

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

Truth degree of goal literal = (Distance to current value)^-1
Identifying Metrics

• **Order relations** Binary, antisymmetric, functional, injective

  \[
  \text{succ}(1,2). \quad \text{succ}(2,3). \quad \text{succ}(3,4). \\
  \text{file}(a,b). \quad \text{file}(b,c). \quad \text{file}(c,d).
  \]

• Order relations define a **metric** on **functional** features

  \[
  \Delta(\text{cell(green,j,13)}, \text{cell(green,e,5)}) = 13
  \]
Degree to which $f(x,a)$ is true given that $f(x,b)$

$$(1-p) - (1-p) \cdot Δ(b,a) / |\text{dom}(f(x))|$$

With $p=0.9$, $\text{eval}(\text{cell(green,e,5)})$ is

0.082 if $\text{true(cell(green,f,10))}$

0.085 if $\text{true(cell(green,j,5))}$
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

For details see [Schiffel & Thielscher, 2007]
Learning by Simulation
Knowledge-Free General Game Playing: Monte Carlo Tree Search

Game Tree Search vs. MC Tree Search
Monte Carlo Tree Search

Value of move = Average score returned by simulation
Improvement: UCT Search

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

$$\text{argmax}_i \left( v_i + C \sqrt{\frac{\log n}{n_i}} \right)$$ (confidence bound)

$$n = 60$$
$$v = 70$$

$$n_1 = 4$$
$$v_1 = 20$$

$$n_2 = 24$$
$$v_2 = 65$$

$$n_3 = 32$$
$$v_3 = 80$$

UCT = Upper Confidence bounds applied to Trees
Assessment

UCT Search works particularly well for games which

- reward greedy behavior
- do not require long-term strategies
- have a large branching factor
- are difficult for humans to play

For details see [Finnsson & Björnsson, 2008]
Demo: An Unstructured Game

Knowledge-Based vs. Simulation-Based (Championship 2008)
Demo: A Structured Game

Simulation-Based vs. Knowledge-Based (Championship 2008)
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

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  games.stanford.edu
  - GDL specification
  - Basic player

- GGP in Germany  general-game-playing.de
  - Game master
  - 24/7 online game playing
  - Extensive collection of GGP literature

- Palamedes  palamedes-ide.sourceforge.net
  - GGP/GDL development tool
Papers

[Clune, 2007]

[Finnsson & Björnsson, 2008]
H. Finnsson, Y. Björnsson. Simulation-based approach to general game playing. AAAI 2008

[Genesereth, Love & Pell, 2006]

[Schiffel & Thielscher, 2007]

[Schiffel & Thielscher, 2009a]

[Schiffel & Thielscher, 2009b]

[Thielscher & Zhang, 2009]