Logic Programming

General Game Playing

Michael Genesereth
Computer Science Department
Stanford University
Game Playing

Human Game Playing
• Intellectual Activity
• Skill Comparison

Computer Game Playing
• Testbed for AI
• Limitations
Limitations of Game Playing for AI

Narrowness
  Good at one game, not so good at others
  Cannot do anything else

Not really testing intelligence of machine
  Programmer does all the interesting analysis / design
  Machine simply follows the recipe
General Game Players are systems able to play arbitrary games effectively based solely on formal descriptions supplied at “runtime”.

Translation: They don’t know the rules until the game starts.

Must figure out for themselves:
- legal moves, winning strategy
- in the face of incomplete info and resource bounds
Variety of Games
Novelty
International GGP Competition
Annual GGP Competition

Held at AAAI or IJCAI conference
Administered by Stanford University
(Stanford folks not eligible to participate)
GGP-05 Winner Jim Clune
International GGP Competition
ClunePlayer - Jim Clune (USA)
FluxPlayer - Schiffel, Thielscher (Germany)
CadiaPlayer - Bjornsson, Finsson (Iceland)
CadiaPlayer - Bjornsson, Finsson (Iceland)
Ary - Mehat (France)
TurboTurtle - Schreiber (USA)
CadiaPlayer - Bjornsson, Finsson (Iceland)
TurboTurtle - Schreiber (USA)
Sancho - Draper (USA), Rose (UK)
Galvanise - Emslie
WoodStock - Piette (France)
Carbon versus Silicon
Human Race Being Defeated
Game Description
Finite Synchronous Games

Environment
- Environment with finitely many states
- One initial state and one or more terminal states
- Each state has a unique goal value for each player

Players
- Fixed, finite number of players
- Each with finitely many moves

Dynamics
- Finitely many steps
- All players move on all steps (some no ops)
- Environment changes only in response to moves
Multiple Player Game
Good News: Since all of the games that we are considering are finite, it is possible in principle to communicate game information in the form of state graphs.
Problem: Size of description. Even though everything is finite, these sets can be large.

Solution:
   Exploit regularities / structure in state graphs to produce compact encoding
Game Description Language (or GDL) is a formal language for encoding the rules of games.

Game rules written as sentences in Symbolic Logic.

GDL is widely used in the research literature and is used in virtually all General Game Playing competitions.

GDL extensions are applicable in real-world applications such as Enterprise Management and Computational Law.
Symbols:

- \( x, o \) - roles
- \( 1, 2, 3 \) - indices of rows and columns
- \( b \) - blank

Constructors:

- \text{mark}(index, index)

Predicates:

- \text{cell}(index, index, mark)
- \text{control}(role)
- \text{row}(index, mark)
- \text{column}(index, mark)
- \text{diagonal}(mark)
- \text{line}(mark)
- \text{open}
State Representation

<table>
<thead>
<tr>
<th>X</th>
<th></th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

= cell(1,1,x)
cell(1,2,b)
cell(1,3,b)
cell(2,1,b)
cell(2,2,o)
cell(2,3,b)
cell(3,1,b)
cell(3,2,b)
cell(3,3,x)
cell(control(o))
**Rules of Tic-Tac-Toe**

- **role** (white)
- **role** (black)

- **base** (cell(M,N,Z)) :-
  - index(M) &
  - index(N) &
  - filler(Z)

- **base** (control(W)) :- role(W)

- **input** (W, mark(X,Y)) :-
  - role(W) &
  - index(X) &
  - index(Y)

- **input** (W, noop) :- role(W)

- **init** (cell(X,Y,b)) :-
  - index(X) &
  - index(Y)

- **init** (control(white))

- **legal** (P, mark(X,Y)) :-
  - true (cell(X,Y,b)) &
  - true (control(P))

- **legal** (x, noop) :-
  - true (control(black))

- **legal** (o, noop) :-
  - true (control(white))

- **next** (cell(M,N,x)) :-
  - does (white, mark(M,N))

- **next** (cell(M,N,b)) :-
  - does (P, mark(J,K)) &
  - true (cell(M,N,b)) &
  - distinct (M,J)

- **next** (cell(M,N,0)) :-
  - does (black, mark(M,N))

- **next** (cell(M,N,Z)) :-
  - does (P, mark(M,N)) &
  - true (cell(M,N,Z)) & Z!=b

- **terminal** :- line(P)

- **row** (M,P) :-
  - true (cell(M,1,P)) &
  - true (cell(M,2,P)) &
  - true (cell(M,3,P))

- **column** (N,P) :-
  - true (cell(1,N,P)) &
  - true (cell(2,N,P)) &
  - true (cell(3,N,P))

- **diagonal** (P) :-
  - true (cell(1,1,P)) &
  - true (cell(2,2,P)) &
  - true (cell(3,3,P))

- **goal** (white, 100) :- line(x) & ~line(o)
- **goal** (white, 50) :- ~line(x) & ~line(o)
- **goal** (white, 0) :- ~line(x) & line(o)
- **goal** (black, 100) :- ~line(x) & line(o)
- **goal** (black, 50) :- ~line(x) & ~line(o)
- **goal** (black, 0) :- line(x) & ~line(o)

- **terminal** :- ~open

- **line** (P) :- row(M,P)
- **line** (P) :- column(N,P)
- **line** (P) :- diagonal(P)

- **open** :- true (cell(M,N,b))

- **goal** (white, 100) :- line(x) & ~line(o)
- **goal** (white, 50) :- ~line(x) & ~line(o)
- **goal** (white, 0) :- ~line(x) & line(o)
- **goal** (black, 100) :- ~line(x) & line(o)
- **goal** (black, 50) :- ~line(x) & ~line(o)
- **goal** (black, 0) :- line(x) & ~line(o)

- **index** (1) filler (x)
- **index** (2) filler (o)
- **index** (3) filler (b)
roles(x)
roles(o)
Propositions

\begin{verbatim}
\textbf{base}(\text{cell}(X,Y,W)) :-
    index(X) &
    index(Y) &
    filler(W)

\textbf{base}(\text{control}(W)) :-
    role(W)

index(1)
index(2)
index(3)

filler(x)
filler(o)
filler(b)
\end{verbatim}
\begin{verbatim}
input(W,mark(X,Y)) :- role(W) & index(X) & index(Y)
input(W,noop) :- role(W)
\end{verbatim}


\textbf{Initial State}

\begin{verbatim}
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(x))
\end{verbatim}
\[ \text{legal}(W, \text{mark}(X,Y)) :\text{= cell}(X,Y,b) \text{ & control}(W) \]
mark(M,N) ::
        control(W) ==> ~cell(M,N,b) & cell(M,N,W)

mark(M,N) :: control(x) ==> control(o)
mark(M,N) :: control(o) ==> control(x)
row(M,W) :- cell(M,1,W) & cell(M,2,W) & cell(M,3,W)
col(N,W) :- cell(1,N,W) & cell(2,N,W) & cell(2,2,W)

diag(W) :- cell(1,1,W) & cell(2,2,W) & cell(3,3,W)
diag(W) :- cell(3,1,W) & cell(2,2,W) & cell(3,1,W)

line(W) :- row(M,W)
line(W) :- col(N,W)
line(W) :- diag(W)

open :- cell(M,N,b)
goals and termination

\[\text{goal}(x, 100) :\text{ line}(x)\]
\[\text{goal}(x, 50) :\text{ ~line}(x) \& \text{ ~line}(o)\]
\[\text{goal}(x, 0) :\text{ line}(o)\]

\[\text{goal}(o, 100) :\text{ line}(o)\]
\[\text{goal}(o, 50) :\text{ ~line}(x) \& \text{ ~line}(o)\]
\[\text{goal}(o, 0) :\text{ line}(x)\]

\text{terminal} :\text{ line}(W) \& \text{ distinct}(W, b)
\text{terminal} :\text{ ~open}
Game Management
Game Management is the process of administering a game in General Game Playing.

Match = instance of a game.

Components:
- Game Manager
- Game Playing Protocol
Game Manager
Start

Manager sends Start message to players

\texttt{start(id,role,description,startclock,playclock)}
Start
Manager sends Start message to players
\texttt{start}(id, role, description, startclock, playclock)

Play
Manager sends Play messages to players
\texttt{play}(id, [action1, \ldots, actionk])
Receives actions in response
Start
Manager sends Start message to players
\[
\text{start}(id, \text{role}, \text{description}, \text{startclock}, \text{playclock})
\]

Play
Manager sends Play messages to players
\[
\text{play}(id, [\text{action1}, \ldots, \text{actionk}])
\]
Receives actions in response

Stop
Manager sends Stop message to players
\[
\text{stop}(id, [\text{action1}, \ldots, \text{actionk}])
\]
Game Playing
Initial State

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cell(1,1,b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cell(1,2,b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cell(1,3,b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cell(2,1,b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cell(2,2,b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cell(2,3,b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cell(3,1,b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cell(3,2,b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cell(3,3,b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>control(x)</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Legal Moves

White’s moves:
mark(1,1)
mark(1,2)
mark(1,3)
mark(2,1)
mark(2,2)
mark(2,3)
mark(3,1)
mark(3,2)
mark(3,3)

Black’s moves:
noop
cell(1,1,b)  cell(1,1,b)  
cell(1,2,b)  cell(1,2,b)  
cell(1,3,b)  cell(1,3,x)  
cell(2,1,b)  cell(2,1,b)  
cell(2,2,b)  cell(2,2,b)  
cell(2,3,b)  cell(2,3,b)  
cell(3,1,b)  cell(3,1,b)  
cell(3,2,b)  cell(3,2,b)  
cell(3,3,b)  cell(3,3,b)  
control(x)  control(o)
Complete Game Graph Search
How do we evaluate non-terminal states?
General Heuristics
  Goal proximity (everyone)
  Maximize mobility (Barney Pell)
  Minimize opponent’s mobility (Jim Clune)
Second Generation GGP (2007 on)

Monte Carlo Search

Monte Carlo Tree Search
  UCT - Uniform Confidence Bounds on Trees
Second Generation GGP

Monte Carlo Search
Offline Processing of Game Descriptions
Reformulate problem to decrease size of search space
Compile to do the search faster

What human programmers do in creating game players
Compilation

Conversion of logic to traditional programming language
Simple, widely published algorithms
several orders or magnitude speedup
no asymptotic change

Conversion to Field Programmable Gate Arrays (FPGAs)
several more orders of magnitude improvement
**Game Factoring**

Hodgepodge = Chess + Othello

![Chess and Othello board](image)

Branching factor: \(a\)  
Branching factor: \(b\)

Analysis of joint game:

Branching factor as given to players: \(a \times b\)

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

Fringe of tree at depth \(n\) factored: \(a^n + b^n\)
Examples
  Factoring, e.g. Hodgepodge
  Bottlenecks, e.g. Triathalon
  Symmetry detection, e.g. Tic-Tac-Toe
  Dead State Removal

Trade-off - cost of finding and using structure vs savings
  *Sometimes* cost proportional to size of description
  *Sometimes* savings proportional to size of game tree
Hex
Naive Axiomatization

Axiomatization:

\[
\begin{align*}
\text{redwin} & : \text{-} \text{left}(X) \& \text{redpath}(X,Y) \& \text{right}(Y) \\
\text{redpath}(X,X) & : \text{true(cell}(X,\text{red})) \\
\text{redpath}(X,Z) & : \text{true(cell}(X,\text{red})) \& \text{adjacent}(X,Y) \& \text{redpath}(Y,Z)
\end{align*}
\]

Results:

Very expensive if path exists
Can run forever if not
Axiomatization:

\[
\text{redwin} :- \text{left}(X) \land \text{redpath}(X,Y,\text{nil}) \land \text{right}(Y)
\]
\[
\text{redpath}(X,X,P) :- \text{true}(\text{cell}(X,\text{red}))
\]
\[
\text{redpath}(X,Z,P) :-
\begin{align*}
& \text{true}(\text{cell}(X,\text{red})) \land \\
& \text{adjacent}(X,Y) \land \\
& \lnot \text{member}(Y,P) \land \\
& \text{redpath}(Y,Z,\text{cons}(Y,P))
\end{align*}
\]

Results:

Does not run forever

\(\text{can take} \sim 1 \text{ second to compute in bad cases}\)
Result:

takes <1 millisecond to compute in worst cases
Automatic Programming

\[
\begin{align*}
  p(a,b) \quad & q(b,c) \\
  \sim p(b,d) \quad & \forall x. \forall y. (p(x,y) \Rightarrow q(x,y)) \\
  p(c,b) \lor p(c,d) \quad & \exists x. p(x,d)
\end{align*}
\]
Expertise in a Box
Opponent Modeling

```
   a   b
  4   3   3
 a  4   2
b  2   1

   a   b
  4   1   1
 a  4   3
b  3   2
```
Demoralizing the Opponent
Philosophical Remarks
General Game Playing is not a game.
Serious Business

ERP SYSTEM

- Financial Management
- Supply Chain Management
- Manufacturing Resource Planning
- Human Resource Management
- Customer Relationship Management

SAP

ORACLE

IBM
Characteristics of GGP

game descriptions contain full information
which determine optimal behavior

Useful for evaluating theories of intelligence
effects of representation
incompleteness of information
resource bounds
The main advantage we expect the **advice taker** to have is that its behavior will be improvable merely by making statements to it, telling it about its ... environment and what is wanted from it. To make these statements will require little, if any, knowledge of the program or the previous knowledge of the advice taker.
The General Problem Solver demonstrates how generality can be achieved by factoring the specific descriptions of individual tasks from the task-independent processes.
The potential use of computers by people to accomplish tasks can be “one-dimensionalized” into a spectrum representing the nature of the instruction that must be given the computer to do its job. Call it the what-to-how spectrum. At one extreme of the spectrum, the user supplies his intelligence to instruct the machine with precision exactly how to do his job step-by-step. ... At the other end of the spectrum is the user with his real problem. ... He aspires to communicate what he wants done ... without having to lay out in detail all necessary subgoals for adequate performance.

- Ed Feigenbaum 1974
A human being should be able to change a diaper, plan an invasion, butcher a hog, conn a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a new problem, pitch manure, program a computer, cook a tasty meal, fight efficiently, die gallantly. specialization is for insects.