General Game Playing

Game Tree Restructuring

Michael Genesereth
Logic Group
Stanford University
Types of Optimization

Techniques so far focus on
- Grounding game descriptions
- Optimizing ground descriptions, e.g. dropping subgoals
- Executing ground and/or symbolized descriptions
  *typically provide polynomial improvement*

Techniques today focus on changing shape of game trees
- Simplifying game trees
  *can provide exponential improvement*
Hodgepodge = Chess + Othello

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: $a^n + b^n$
Double Tic Tac Toe

Analysis of joint game:
Branching factor: 81, 64, 49, 36, 25, 16, 9, 4, 1
Branching factor: 9, 8, 7, 6, 5, 4, 3, 2, 1
"Multiple" Games
Propnet for "Multiple" Game
Ground the Game.

Compute actions to goals, termination, legality.

Adjust legalities.
Grounding

\[
\text{legal}(a(X)) \leftarrow \text{index}(X) \& \neg \text{step}(X,7)
\]

\[
\begin{align*}
\text{legal}(a(1)) & \leftarrow \text{index}(1) \& \neg \text{step}(1,7) \\
\text{legal}(a(2)) & \leftarrow \text{index}(2) \& \neg \text{step}(2,7) \\
\text{legal}(a(3)) & \leftarrow \text{index}(3) \& \neg \text{step}(3,7) \\
\text{legal}(a(4)) & \leftarrow \text{index}(4) \& \neg \text{step}(4,7) \\
\text{legal}(a(5)) & \leftarrow \text{index}(5) \& \neg \text{step}(5,7) \\
\text{legal}(a(6)) & \leftarrow \text{index}(6) \& \neg \text{step}(6,7) \\
\text{legal}(a(7)) & \leftarrow \text{index}(7) \& \neg \text{step}(7,7) \\
\text{legal}(a(8)) & \leftarrow \text{index}(8) \& \neg \text{step}(8,7) \\
\text{legal}(a(9)) & \leftarrow \text{index}(9) \& \neg \text{step}(9,7)
\end{align*}
\]
Actions That Affect Goals

\[
\begin{align*}
\text{goal(robot,100)} & : p(5) \land q(5) \land r(5) \\
\text{goal(robot,50)} & : p(5) \land q(5) \land \neg r(5) \\
\text{goal(robot,50)} & : p(5) \land \neg q(5) \land r(5) \\
\text{goal(robot,50)} & : \neg p(5) \land q(5) \land r(5) \\
\text{goal(robot,25)} & : p(5) \land \neg q(5) \land \neg r(5) \\
\text{goal(robot,25)} & : \neg p(5) \land q(5) \land \neg r(5) \\
\text{goal(robot,25)} & : \neg p(5) \land \neg q(5) \land \neg r(5) \\
\text{goal(robot,0)} & : \neg p(5) \land \neg q(5) \land \neg r(5)
\end{align*}
\]

\{a(5), b(5), c(5)\}
Termination

\[
\text{terminal} :: \ p(5) \ & \ q(5) \ & \ r(5) \\
\text{terminal} :: \ \text{step}(5,7) \\
\{ \text{a}(5), \ \text{b}(5), \ \text{c}(5) \} 
\]
Legality of Relevant Actions

\[
\begin{align*}
\text{legal}(a(5)) & \leftarrow \text{index}(5) \land \lnot \text{step}(5,7) \\
\text{legal}(b(5)) & \leftarrow \text{index}(5) \land \lnot \text{step}(5,7) \\
\text{legal}(c(5)) & \leftarrow \text{index}(5) \land \lnot \text{step}(5,7)
\end{align*}
\]

\{a(5), b(5), c(5)\}
Adjusting Legality

\[
\text{legal}(a(X)) \leftarrow \text{index}(X) \& \neg \text{step}(X, 7) \\
\text{legal}(b(X)) \leftarrow \text{index}(X) \& \neg \text{step}(X, 7) \\
\text{legal}(c(X)) \leftarrow \text{index}(X) \& \neg \text{step}(X, 7)
\]

\[
\text{legal}(a(5)) \leftarrow \text{index}(5) \& \neg \text{step}(5, 7) \\
\text{legal}(b(5)) \leftarrow \text{index}(5) \& \neg \text{step}(5, 7) \\
\text{legal}(c(5)) \leftarrow \text{index}(5) \& \neg \text{step}(5, 7)
\]
Resources

Gamemaster

Sign In

Resources

Game Processing Utilities

Sierra - interactive development environment for Epilog rulesets and datasets. Click here for documentation.

Check Rules - allows users to test (Sierra configuration)

Check Style - allows users to test Javascript stylesheets on sample states.

Game Checker - allows users to test games and sstylesheets.

Ground - allows users to convert game descriptions with variables to fully grounded versions. Useful for game analysis, such as finding factors, latches, dead states, and so forth.

Simplify - allows users to simplify games to include only potentially relevant actions based on dependency analysis.

Conjoin - allows users to factor games into conjunctive subgames based on dependency analysis.

Optimize - allows users to perform various optimizations on game descriptions.

Materialize - allows users to materialize relations used in game descriptions.

Game Management Applications (Server-Based Games)

Standalone - human player interface and game manager in single web page

Competitor - human player interface for competitions

Autoplayer - automated player interface for competitions

Manager - game manager for competitions
Gamemaster

Simplify

Game Description:

```
role(robot)
base(p(X)) :- index(X)
base(q(X)) :- index(X)
base(r(X)) :- index(X)

base(step(X,1)) :- index(X)
base(step(X,2)) :- index(X)
base(step(X,3)) :- index(X)
base(step(X,4)) :- index(X)
base(step(X,5)) :- index(X)
```

Result:

```
role(robot)
base(p(X)) :- index(X)
base(q(X)) :- index(X)
base(r(X)) :- index(X)
base(step(X,1)) :- index(X)
base(step(X,2)) :- index(X)
base(step(X,3)) :- index(X)
base(step(X,4)) :- index(X)
base(step(X,5)) :- index(X)
base(step(X,6)) :- index(X)
base(step(X,7)) :- index(X)
action(a(X)) :- index(X)
action(b(X)) :- index(X)
action(c(X)) :- index(X)
index(1)
index(2)
index(3)
index(4)
```
Gamemaster

Protocol: standalone
Game: multiplebuttonsandlights 🕳️

Move: 🐸
Gamemaster

Protocol: standalone
Game: multipleswitches

Move:
Gamemaster

Protocol: standalone
Game: multipletictactoe

Game over

Move: 

O O O

O X X X

O
### Gamemaster

**General Game Playing**

<table>
<thead>
<tr>
<th>Game</th>
<th>Depth</th>
<th>Result</th>
<th>Normal Terms</th>
<th>Normal Nodes</th>
<th>Normal Runtime</th>
<th>Simple Terms</th>
<th>Simple Nodes</th>
<th>Simple Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplebuttonsandlights</td>
<td>4</td>
<td>50</td>
<td>531441</td>
<td>551881</td>
<td>16700</td>
<td>81</td>
<td>121</td>
<td>6</td>
</tr>
<tr>
<td>multipleswitches</td>
<td>5</td>
<td>100</td>
<td>3074591520</td>
<td>~4B</td>
<td>~22M</td>
<td>15120</td>
<td>18730</td>
<td>210</td>
</tr>
</tbody>
</table>
Gamemaster

Resources

Game Processing Utilities

- **Sierra** - interactive development environment for Epilog rulesets and datasets. Click [here](#) for documentation.
- **Check Rules** - allows users to test (Sierra configuration)
- **Check Style** - allows users to test Javascript stylesheets on sample states.
- **Game Checker** - allows users to test games and sstylesheets.
- **Ground** - allows users to convert game descriptions with variables to fully grounded versions. Useful for game analysis, such as finding factors, latches, dead states, and so forth.
- **Simplify** - allows users to simplify games to include only potentially relevant actions based on dependency analysis.
- **Conjoin** - allows users to factor games into conjunctive subgames based on dependency analysis.
- **Optimize** - allows users to perform various optimizations on game descriptions.
- **Materialize** - allows users to materialize relations used in game descriptions.

Game Management Applications (Server-Based Games)

- **Standalone** - human player interface and game manager in single web page
- **Competitor** - human player interface for competitions
- **Autoplayer** - automated player interface for competitions
- **Manager** - game manager for competitions
Simplification Subroutines

Browser-Based Players

- Legal - Legal player
- Random - Random player
- One-step - One Step player
- Minimax - Full Minimax player
- Minimax-depth - Minimax player with fixed depth
- Minimax-id - Minimax player with iterative deepening
- Greedy - Greedy player
- MCS - Monte Carlo Search player

JavascriptCode

- EpilogJS - Logic Programming Interpreter
- Basics - Basic subroutines for general games
- Ground - Basic subroutines for ground games
- Symbol - Basic subroutines for symbolized games
- Simple - Simplification subroutines
- ParametricJS - Code for standard players
- PlayerJS - Player set-up for NodeJS

Older General Game Playing Websites

- Tiltyard (web site) - allows users to register players for automatic round robin competition against other general game playing programs. Warning: Games on this website do not necessarily comply with the current General Game Playing standard.

- GGPore (web page) - General website on GGP. Contains information on how to develop software for GGP. Warning: Games on this website do not necessarily comply with the current General Game Playing standard.
Game Factoring
Propnet for Best Buttons and Lights
Modified Propnet
Conditional Factoring
Conditional Factoring
Other Techniques
Bottlenecks
  Series of games
  each of which must terminate before next begins

Dead State Elimination
  Find states that cannot lead to acceptable outcomes
  Prune whole subtrees

Goal Monotonicity
  Detect monotonicity in states
  e.g. higher goal value in non-terminal states
  correlated with progress toward goal