## General vs Symbolized Skirmish

**General:**
- minimax depth 1  21 nodes in 77 msec
- minimax depth 2  421 nodes in 1151 msec
- minimax depth 3  9383 nodes in 23532 msec
- depthcharge      35 nodes in 1560 msec

**Symbolized:**
- minimax depth 1  21 nodes in 39 msec
- minimax depth 2  421 nodes in 335 msec
- minimax depth 3  9383 nodes in 5354 msec
- depthcharge      35 nodes in 446 msec
Before:
  2174 base propositions
  2048 actions
  564194 rules computed in 258,565 msec

After:
  2287 base propositions
  2177 actions
  53884 rules in 48,343 msec
General Game Playing

Game Reformulation

Michael Genesereth
Logic Group
Stanford University
Types of Optimization

Game Optimizations with standard interpreter:
Logical Optimization, e.g. dropping subgoals
*Objective: compute game tree faster*

Game Optimizations with different interpreters:
Grounding
Symbolizing
*Objective: compute game tree faster*

Game Reformulation:
Pruning game trees
Decomposing games into independent subgames
*Objective: decrease the size of the game tree*
Egghead versus Maverick

Multipletictactoe
Gamemaster

Protocol: standalone
Game: multiplebuttonsandlights

Move: ☐
Gamemaster

Protocol: standalone
Game: multipleswitches

Move:
"Multiple" Games
Propnet for "Multiple" Game
Inertia
(1) Ground the Game.

(2) Compute actions affecting goals, termination, legality.

(3) Adjust legalities to eliminate useless actions.
Grounding

\[
\text{legal(a(X)) :- index(X) } \& \sim \text{step(X,7)}
\]

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

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

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

terminal :- p(5) & q(5) & r(5)
terminal :- step(5,7)

a(5) :: step(5,1) ==> ~step(5,1) & step(5,2)
a(5) :: step(5,2) ==> ~step(5,2) & step(5,3)

... 
b(5) :: step(5,1) ==> ~step(5,1) & step(5,2)
b(5) :: step(5,2) ==> ~step(5,2) & step(5,3)

... 
c(5) :: step(5,5) ==> ~step(5,5) & step(5,6)
c(5) :: step(5,6) ==> ~step(5,6) & step(5,7)

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

a(5) :: ~p(5) ==> p(5)
a(5) :: p(5) ==> ~p(5)
b(5) :: q(5) ==> p(5)
b(5) :: ~q(5) ==> ~p(5)
b(5) :: p(5) ==> q(5)
b(5) :: ~p(5) ==> ~q(5)
c(5) :: q(5) ==> r(5)
c(5) :: ~q(5) ==> ~r(5)
c(5) :: r(5) ==> q(5)
c(5) :: ~r(5) ==> ~q(5)
Legality of Relevant Actions

legal(\(a(5)\)) :- index(5) & \(\neg\)step(5,7)
legal(\(b(5)\)) :- index(5) & \(\neg\)step(5,7)
legal(\(c(5)\)) :- index(5) & \(\neg\)step(5,7)

\(a(5)::step(5,1)===>\neg step(5,1)&step(5,2)\)
\(a(5)::step(5,2)===>\neg step(5,2)&step(5,3)\)
\(\vdots\)
\(b(5)::step(5,1)===>\neg step(5,1)&step(5,2)\)
\(b(5)::step(5,2)===>\neg step(5,2)&step(5,3)\)
\(\vdots\)
\(c(5)::step(5,5)===>\neg step(5,5)&step(5,6)\)
\(c(5)::step(5,6)===>\neg step(5,6)&step(5,7)\)

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

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

\{a(5), b(5), c(5)\}
Protocol: standalone
Game: multipletictactoe

Game over

Move: ___
### Experimental Results

The table below presents the results of various games using the *Gamemaster* software.

<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>
</table>
Egghead versus Maverick

Parallelknightthrough
Egghead versus Maverick

Parallelknightthrough
**Standard Players**

- **Legal** - Legal player
- **Random** - Random player
- **Onestep** - One Step player using intermediate values
- **Twostep** - Two Step player using intermediate values
- **Minimax** - Full Minimax player using intermediate values
- **Minimaxdepth** - Minimax player to fixed depth using intermediate values
- **Minimaxid** - Minimax player with iterative deepening using intermediate values
- **MCS** - Monte Carlo Search - one step player using depth charges
- **PTS** - Minimax player with persistent breadth-first search using intermediate values
- **Greedy** - PTS player with search based on exploration and exploitation

**Standard Metagamers**

- **Optimizer** - performs optimizations on game descriptions.
- **Materializer** - materializes relations used in game descriptions.
- **Simplifier** - simplifies games by eliminating subgoals or rules based on ground facts in the game description.
- **Grounder** - converts game descriptions with variables to fully grounded versions.
- **Symbolizer** - converts game descriptions with variables to fully grounded and symbolized versions (i.e. with all ground atoms converted to propositions).
- **Pruner** - prunes games to include only potentially relevant actions based on dependency analysis of fully grounded game descriptions.

**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.
Gamemaster

General
Game
Playing

Pruner

Game Description:

```prolog
%% legal

legal(a(X)) :- index(X) & ~step(X,7)
legal(b(X)) :- index(X) & ~step(X,7)
legal(c(X)) :- index(X) & ~step(X,7)
```

```prolog
%% operations

a(X) :: ~p(X) ==> p(X)
a(X) :: p(X) ==> ~p(X)
a(X) :: step(X,N) & successor(M,N) ==> ~step(X,M) & step(X,N)
```

```prolog
b(X) :: q(X) ==> p(X)
b(X) :: ~q(X) ==> ~p(X)
```

Result:

```prolog
goal(robot,50) :- p(5) & ~q(5) & r(5)
goal(robot,50) :- ~p(5) & q(5) & r(5)
goal(robot,25) :- p(5) & ~q(5) & ~r(5)
goal(robot,25) :- ~p(5) & q(5) & ~r(5)
goal(robot,0) :- ~p(5) & ~q(5) & ~r(5)
terminal :- p(5) & q(5) & r(5)
terminal :- step(5,7)
successor(1,2)
successor(2,3)
successor(3,4)
successor(4,5)
successor(5,6)
successor(6,7)
legal(a(5)) :- index(5) & ~step(5,7)
legal(b(5)) :- index(5) & ~step(5,7)
legal(c(5)) :- index(5) & ~step(5,7)
```
Software

Standard Players

- legal.js - Legal player
- random.js - Random player
- onestep.js - One Step player
- twostep.js - Two Step player
- minimax.js - Full Minimax player
- minimaxdepth.js - Minimax player with fixed depth
- minimaxid.js - Minimax player with iterative deepening
- greedy.js - Greedy player
- mcs.js - Monte Carlo Search player

Metagamers

- grounder.js - Grounding subroutines
- symbolizer.js - Symbolizing subroutines
- pruner.js - Simplification subroutines

Reasoners

- general.js - Subroutines for computing properties of games in general representation
- ground.js - Subroutines for computing properties of grounded games
- symbol.js - Basic subroutines for computing properties of symbolized games
Including Metagaming Code in Players

<script src='http://epilog.stanford.edu/javascript/epilog.js'></script>
<script src='http://gamemaster.stanford.edu/javascript/localstorage.js'></script>

<script src='http://gamemaster.stanford.edu/metagaming/grounder.js'></script>
<script src='http://gamemaster.stanford.edu/metagaming/symbolizer.js'></script>
<script src='http://gamemaster.stanford.edu/metagaming/simplifier.js'></script>

<script src='http://gamemaster.stanford.edu/metagaming/pruner.js'></script>

<script src='http://gamemaster.stanford.edu/gameplaying/pts.js'></script>
<script src='http://gamemaster.stanford.edu/reasoning/general.js'></script>
function start (r,rs,sc,pc)
    {role = r;
     rules = rs.slice(1)
     startclock = parseInt(sc);
     playclock = parseInt(pc);

     library = definemorerules([],rules);
     roles = findroles(library);
     state = findinits(library);
     return 'ready'}
function start (r, rs, sc, pc)
    {role = r;
        rules = rs.slice(1)
        startclock = parseInt(sc);
        playclock = parseInt(pc);

        rules = definemorerules([], rules);
        rules = groundrules(rules);
        rules = symbolizerules(rules);
        rules = simplifyrules(rules);

        rules = definemorerules([], rules);
        rules = pruneprogram(rules);

        library = definemorerules([], rules);
        roles = findroles(library);
        state = findinits(library);
        return 'ready'}
Game Factoring
Propnet for a "Multiple" Game

```
  p1  q1  r1  
  |    |    |    
  a1  b1  c1  

  p2  q2  r2  
  |    |    |    
  a2  b2  c2  

  p3  q3  r3  
  |    |    |    
  a3  b3  c3  
```

spaghetti

spaghetti

spaghetti
Propnet for a "Best" Game

Diagram with labeled nodes and edges.
Propnet for a "Joint" Game
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: $a^n + b^n$
Best Tic Tac Toe

Joint branching factor: 81, 64, 49, 36, 25, 16, 9, 4, 1
Separate branching factor: 9, 8, 7, 6, 5, 4, 3, 2, 1
Propnet for a "Best" Game
Modified Propnet

```
spaghetti
\[ t1 \]
\[ p1 \quad q1 \quad r1 \]
\[ a1 \quad b1 \quad c1 \]

spaghetti
\[ t2 \]
\[ p2 \quad q2 \quad r2 \]
\[ a2 \quad b2 \quad c2 \]

spaghetti
\[ t3 \]
\[ p3 \quad q3 \quad r3 \]
\[ a3 \quad b3 \quad c3 \]
```
Conditional Factoring
Conditional Factoring
Other Techniques
Examples

Bottlenecks
  Series of games
  each of which must terminate before next begins

Invariant Detection (aka latch detection)
  Find states that lead only to max terminal value
  Find states that lead only to min terminal value
  e.g. step off roof, 0 value from there on out

Goal Monotonicity
  Detect monotonicity in states - use as values
  e.g. goal values in non-terminal states never decrease