General Game Playing

Metagaming

Michael Genesereth
Computer Science Department
Stanford University
Metagaming is match-independent game processing, i.e. game processing that is done independent of any particular opponent or any particular state.

Objective of metagaming - to optimize performance in playing specific matches of the game.

Usually done offline, i.e. during the startclock or between moves or in parallel with regular game play.
Headstart = game tree search during the start clock.

```javascript
function start (r,rs,sc,pc)
{role = r;
 library = definemorerules([],rs.slice(1));
roles = findroles(library);
state = findinits(library);
startclock = parseInt(sc);
playclock = parseInt(pc);
var reward = parseInt(findreward(role,state,library));
tree = makenode(state,findcontrol(state,library),reward);
return 'ready'}

function play (move)
{if (move!==nil)
 {tree = subtree(move,tree); state = tree.state};
if (findcontrol(state,library) !== role) {return false};
var deadline = Date.now()+playclock*1000;
while (Date.now()<deadline) {process(tree)};
return selectaction(tree)}
```
function start (r, rs, sc, pc)
{role = r;
    library = definemorerules([], rs.slice(1));
    roles = findroles(library);
    state = findinits(library);
    startclock = parseInt(sc);
    playclock = parseInt(pc);
    var reward = parseInt(findreward(role, state, library));
    tree = makenode(state, findcontrol(state, library), reward);
    var deadline = Date.now() + startclock*1000;
    while (Date.now()<deadline) {process(tree)};
    return 'ready'}

function play (move)
{if (move!==nil)
    {tree = subtree(move, tree); state = tree.state};
if (findcontrol(state, library)!==role) {return false};
    var deadline = Date.now()+playclock*1000;
    while (Date.now()<deadline) {process(tree)};
    return selectaction(tree)}
Bigswitch is using the startclock to analyze a game and use the analysis to adjust runtime play.

(1) runtime players (e.g. iterative deepening vs greedy)

(2) evaluation functions (e.g. mobility, intermediate values, depth charges)

(3) selection functions or coefficients (e.g. coefficients for exploration and exploitation)
How to Decide

Analyze rules *somehow*

Run some depth charges:
  - Check for intermediate state values
  - Branching factor
  - Depth
  - Cost of expanding
#3 Rewrite Rules

**Source-to-Source Transformations**
changing game descriptions before runtime
possibly changing players

(a) Logical Optimization (week 6 - today!)

(b) Game Tree Restructuring (week 7)

(c) Game Grounding, Symbolizing, and Vectorizing (week 8)
Symbolic Reasoning (week 9)
Automatic Theorem Proving
Automatic Programming
Logical Simplification
Example

SEND
+MORE
-----
MONEY
solution(S,E,N,D,M,O,R,Y) :-
    digit(S) & digit(E) & digit(N) & digit(D) &
    digit(M) & digit(O) & digit(R) & digit(Y) &
    M!=0 & M!=S & M!=E & M!=N & M!=D &
    O!=S & O!=E & O!=N & O!=D & O!=M &
evaluate(S*1000+E*100+N*10+D,U) &
evaluate(M*1000+O*100+R*10+E,V) &
evaluate(M*10000+O*1000+N*100+E*10+Y,W) &
evaluate(plus(U,V),W)
Analysis

Data

digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)

Rule

goal(S,E,N,D,M,O,R,Y) :-
digit(S) & digit(E) & digit(N) & digit(D) &
digit(M) & digit(O) & digit(R) & digit(Y) & ...

Analysis

10x10x10x10x10x10x10x10x10 = 10^8 = 100,000,000 instances
8 Queens
digit(1)  digit(5)
digit(2)  digit(6)
digit(3)  digit(7)
digit(4)  digit(8)

base(cell(X,Y)) :- digit(X) & digit(Y)

solution(C1,C2,C3,C4,C5,C6,C7,C8) :-
    cell(C1) & cell(C2) & cell(C3) & cell(C4) &
    cell(C5) & cell(C6) & cell(C7) & cell(C8) &
    ~attacks(C1,C2) & ... & ~attacks(C7,C8)

Analysis

64 x 64 x 64 x 64 x 64 x 64 x 64 x 64 =
281,474,976,710,656 instances
Subgoal Ordering

Original Rule

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
```

Reformulation

```
solution(X,Y) :- p(X) & q(X) & r(X,Y)
```
Analysis

Original Rule

\[ \text{solution}(X,Y) :- p(X) \land r(X,Y) \land q(X) \]

\[ (n^2 + 2n) + n*((n^2 + 2n) + n*(n^2 + 2n)) = n^4 + 3n^3 + 3n^2 + 2n \]

Reformulation

\[ \text{solution}(X,Y) :- p(X) \land q(X) \land r(X,Y) \]

\[ (n^2 + 2n) + n*((n^2 + 2n) + 1*(n^2 + 2n)) = 2n^3 + 5n^2 + 2n \]
Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove first subgoal, add to new query, and repeat.

Output the new query.

Example:

\[
\text{solution}(X,Y) :- \ p(X) \ & \ r(X,Y) \ & \ q(X) \\
\text{solution}(X,Y) :- \\
\]
Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove first subgoal, add to new query, and repeat.

Output the new query.

Example:

\[
\text{solution}(X,Y) :- p(X) \& r(X,Y) \& q(X) \\
\text{solution}(X,Y) :- p(X)
\]
Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove first subgoal, add to new query, and repeat.

Output the new query.

Example:

\[
\text{solution}(X,Y) :- p(X) \& r(X,Y) \& q(X) \\
\text{solution}(X,Y) :- p(X) \& q(X)
\]
Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove first subgoal, add to new query, and repeat.

Output the new query.

Example:

\[
\text{solution}(X,Y) \leftarrow p(X) \land r(X,Y) \land q(X)
\]

\[
\text{solution}(X,Y) \leftarrow p(X) \land q(X) \land r(X,Y)
\]
Example

SEND
+MORE
-----
MONEY
solution(S,E,N,D,M,O,R,Y) :-
digit(S) & digit(E) & digit(N) & digit(D) &
digit(M) & digit(O) & digit(R) & digit(Y) &
M!=0 & M!=S & M!=E & M!=N & M!=D &
O!=S & O!=E & O!=N & O!=D & O!=M &
evaluate(S*1000+E*100+N*10+D,X) &
evaluate(M*1000+O*100+R*10+E,Y) &
evaluate(M*10000+O*1000+N*100+E*10+Y,Z) &
evaluate(plus(X,Y),Z)
Data

digit(1)  digit(6)
digit(2)  digit(7)
digit(3)  digit(8)
digit(4)  digit(9)
digit(5)  digit(0)

Rule

solution(S,E,N,D,M,O,R,Y) :-
    digit(S) & digit(E) & digit(N) & digit(D) &
    digit(M) & digit(O) & digit(R) & digit(Y) & ...

Analysis

10x10x10x10x10x10x10x10 = 10^8 = 100,000,000 cases
111,111,110 unifications
Running time ~ minutes
solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & S!=0 &
  digit(E) & E!=S &
  digit(N) & N!=S & N!=E &
  digit(D) & D!=S & D!=E & D!=N &
  digit(M) & M!=0 & M!=S & M!=E & M!=N & M!=D &
  digit(O) & O!=S & O!=E & O!=N & O!=D & O!=M &
  digit(R) & R!=S & R!=E & R!=N & R!=D &
      R!=M & R!=O &
  digit(Y) & Y!=S & Y!=E & Y!=N & Y!=D &
      Y!=M & Y!=O & Y!=R &
  evaluate(S*1000+E*100+N*10+D,X) &
  evaluate(M*1000+O*100+R*10+E,Y) &
  evaluate(M*10000+O*1000+N*100+E*10+Y,Z) &
  evaluate(plus(X,Y),Z)
Computational Analysis

Rule

\[
solution(S,E,N,D,M,O,R,Y) :-
\]
\[
digit(S) \land S\neq 0 \land \\
digit(E) \land E\neq S \land \\
digit(N) \land N\neq S \land N\neq E \land \\
digit(D) \land D\neq S \land D\neq E \land D\neq N \land \\
digit(M) \land M\neq 0 \land M\neq S \land M\neq E \land M\neq N \land M\neq D \land \\
digit(O) \land O\neq S \land O\neq E \land O\neq N \land O\neq D \land O\neq M \land \\
digit(R) \land R\neq S \land R\neq E \land R\neq N \land R\neq D \land
\]
\[
R\neq M \land R\neq O \land \\
digit(Y) \land Y\neq S \land Y\neq E \land Y\neq N \land Y\neq D \land
\]
\[
Y\neq M \land Y\neq O \land Y\neq R \land \ldots \]

Analysis

\[
10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 = 1,814,400 \text{ cases}
\]
\[
5,989,558 / 7,921,010 \text{ unifications}
\]
Interpreted $\sim$ 40 seconds  Compiled $\sim$ 4 seconds
Rule

\[
\text{goal}(S,E,N,D,M,O,R,Y) :- \\
\text{digit}(D) \ & \\
\text{digit}(E) \ & \text{mutex}(E,D) \ & \\
\text{digit}(Y) \ & \text{mutex}(Y,D,E) \ & \\
\text{evaluate}(\text{remainder}(\text{plus}(D,E),10),Y) \ & \\
\text{digit}(S) \ & \text{distinct}(S,0) \ & \text{mutex}(S,D,E,Y) \ & \\
\text{digit}(N) \ & \text{mutex}(N,D,E,Y,S) \ & \\
\text{digit}(M) \ & \text{distinct}(M,0) \ & \text{mutex}(M,D,E,Y,S,N) \ & \\
\text{digit}(O) \ & \text{mutex}(O,D,E,Y,S,N,M) \ & \\
\text{digit}(R) \ & \text{mutex}(R,D,E,Y,S,N,M,O) \ & \ldots
\]

Analysis

438,728 / 7,921,010 unifications
Interpreted ~ 4 seconds
Original Rule:

\[ \text{solution}(X,Y) \ :- \ p(X,Y) \ & \ q(Y) \ & \ q(Z) \]

Equivalent Reformulation:

\[ \text{solution}(X,Y) \ :- \ p(X,Y) \ & \ q(Y) \]
Redundant Rules:

\[
\text{solution}(X) :- \ p(X,b) & q(b) & r(Z) \\
\text{solution}(X) :- \ p(X,Y) & q(Y) & r(Z)
\]

Non-Redundant Rules:

\[
\text{solution}(X) :- \ p(X,b) & q(b) & r(Z) \\
\text{solution}(X) :- \ p(X,Y) & q(Y) & r(c)
\]
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.
Game Description:

\[ s(X,Y,Z) :- q(X,Y) \land r(Y,Z) \land p(X) \]
\[ h(X,Y,Z) :- g(X,Y) \land q(Y,Z) \land g(X,Z) \]
\[ j(Y) :- g(X,Y) \land p(Y) \]
\[ j(Y) :- g(X,Y) \]
\[ p(a) \]
\[ p(b) \]
\[ p(c) \]
\[ g(a,a) \]
\[ g(a,b) \]
\[ g(b,c) \]
\[ g(b,d) \]
\[ g(c,a) \]
\[ g(c,b) \]
\[ g(d,a) \]

Analysis:
Loading Metagaming Code

<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/optimizer.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 = numberize(sc);
    playclock = numberize(pc);

    rules = prunerulesubgoals(rules);
    rules = prunerules(rules);
    rules = fixrules(rules);

    library = definemorerules([], rs);
    roles = findroles(library);
    state = findinits(library);
    var active = findcontrol(state, library);
    var reward = parseInt(findreward(role, state, library));
    tree = makenode(state, active, reward, false);
    return 'ready'}
Materialization
Materialization is the process of precomputing a view relation and storing the result so that subsequent queries can be answered by lookup rather than computation.

**Advantage:** Look up is often much cheaper than computation. Efficiency advantage if the relation is queried often.

**Disadvantages:** Relation may be large (e.g. third generation relatives of US population) or even infinite (e.g. the set of all primes) and/or expensive to compute (e.g. factors of a large number) and may be accessed only infrequently.
goal(robot,0) :- cell(1,1) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0) :- cell(2,2) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,100) :- cell(3,3) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0) :- cell(4,4) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0) :- cell(5,5) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0) :- cell(M,N) & distinct(M,N)
solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5) :-
  index(X1) & index(Y1) &
  index(X2) & index(Y2) &
  index(X3) & index(Y3) &
  index(X4) & index(Y4) &
  index(X5) & index(Y5) &
  ~queenmove(X1,Y1,X2,Y2) &
  ~queenmove(X1,Y1,X3,Y3) &
  ~queenmove(X1,Y1,X4,Y4) &
  ~queenmove(X1,Y1,X5,Y5) &
  ~queenmove(X2,Y2,X3,Y3) &
  ~queenmove(X2,Y2,X4,Y4) &
  ~queenmove(X2,Y2,X5,Y5) &
  ~queenmove(X3,Y3,X4,Y4) &
  ~queenmove(X3,Y3,X5,Y5) &
  ~queenmove(X4,Y4,X5,Y5)
Example

goal(robot,0) :- cell(1,1) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0) :- cell(2,2) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,100) :- cell(3,3) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0) :- cell(4,4) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0) :- cell(5,5) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0) :- cell(M,N) & distinct(M,N)

solution(1,1,2,3,3,5,4,2,5,4)
...

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
- **Minimaxed** - 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

Materializer

Game Description:

```prolog
role(robot)

base(f(X,Y)) :- g(X,Y)
base(e(X,Y)) :- g(X,Y)

action(doit(a))
init(f(a,b))
init(e(a,b))

s(X,Y,Z) :- g(X,Y) & r(Y,Z) & p(X)
h(X,Y,Z) :- g(X,Y) & g(Y,Z) & j(Z)
j(Y) :- g(X,Y) & k(Y)
k(Y) :- f(X,Y)
l(Y) :- g(Y,X)
g(a,a)
g(a,b)
```

Analysis:
<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/materializer.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 = numberize(sc);
   playclock = numberize(pc);

   rules = materializestaticrelations(rules);

   library = definemorerules([],rs);
   roles = findroles(library);
   state = findinits(library);
   var active = findcontrol(state,library);
   var reward = parseInt(findreward(role,state,library));
   tree = makenode(state,active,reward,false);
   return 'ready'}
Simplification
Simplification is the process of substituting static facts into rules and eliminating true subgoals and eliminating rules with false subgoals.
Example

\[ h(a,b,c) :- g(a,b) \land g(b,c) \]

\[ j(a) :- g(a,a) \land k(a) \]
\[ j(b) :- g(a,b) \land k(b) \]
\[ j(c) :- g(b,c) \land k(c) \]
\[ j(a) :- g(c,c) \land k(a) \]
\[ j(b) :- g(c,b) \land k(b) \]

\[ k(a) :- f(a,a) \]
\[ k(b) :- f(a,b) \]
\[ k(c) :- f(b,c) \]

\[ g(a,b) \]
\[ g(b,c) \]
\[ g(b,c) \]
\[ g(c,a) \]
\[ g(c,b) \]
Example

\[ h(a,b,c) :- g(a,b) \land g(b,c) \]

\[ j(a) :- g(a,a) \land k(a) \]
\[ j(b) :- g(a,b) \land k(b) \]
\[ j(c) :- g(b,c) \land k(c) \]
\[ j(a) :- g(c,c) \land k(a) \]
\[ j(b) :- g(c,b) \land k(b) \]

\[ k(a) :- f(a,a) \]
\[ k(b) :- f(a,b) \]
\[ k(c) :- f(b,c) \]

\[ g(a,b) \]
\[ g(b,c) \]
\[ g(c,a) \]
\[ g(c,b) \]
Example of Materialization + Simplification

goal(robot,0)  :-  cell(1,1) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)  :-  cell(2,2) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,100)  :-  cell(3,3) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)  :-  cell(4,4) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)  :-  cell(5,5) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)  :-  cell(M,N) & distinct(M,N)

solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5) :-
  index(X1) & index(Y1) &
  index(X2) & index(Y2) &
  index(X3) & index(Y3) &
  index(X4) & index(Y4) &
  index(X5) & index(Y5) &
  ~queenmove(X1,Y1,X2,Y2) &
  ~queenmove(X1,Y1,X3,Y3) &
  ~queenmove(X1,Y1,X4,Y4) &
  ~queenmove(X1,Y1,X5,Y5) &
  ~queenmove(X2,Y2,X3,Y3) &
  ~queenmove(X2,Y2,X4,Y4) &
  ~queenmove(X2,Y2,X5,Y5) &
  ~queenmove(X3,Y3,X4,Y4) &
  ~queenmove(X3,Y3,X5,Y5) &
  ~queenmove(X4,Y4,X5,Y5)
Example of Materialization + Simplification

\[
\begin{align*}
goal(robot,0) & \quad :- \ cell(1,1) \\
goal(robot,0) & \quad :- \ cell(2,2) \\
goal(robot,100) & \quad :- \ cell(3,3) \\
goal(robot,0) & \quad :- \ cell(4,4) \\
goal(robot,0) & \quad :- \ cell(5,5) \\
goal(robot,0) & \quad :- \ cell(M,N) & \text{distinct}(M,N)
\end{align*}
\]
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.

GGP.org (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 Description:

```prolog
role(robot)
init(f(a,b))
init(e(a,b))

h(a,a,a) :- g(a,a) & g(a,a) & j(a)

h(a,a,b) :- g(a,a) & g(a,b) & j(b)

h(a,b,c) :- g(a,b) & g(b,c) & j(c)

h(a,b,d) :- g(a,b) & g(b,d) & j(d)

h(b,c,a) :- g(b,c) & g(c,a) & j(a)

h(b,c,b) :- g(b,c) & g(c,b) & j(b)

h(b,d,c) :- g(b,d) & g(d,c) & j(c)

h(b,d,d) :- g(b,d) & g(d,d) & j(d)

h(c,a,a) :- g(c,a) & g(a,a) & j(a)

h(c,a,b) :- g(c,a) & g(a,b) & j(b)

h(c,b,c) :- g(c,b) & g(b,c) & j(c)

h(c,b,d) :- g(c,b) & g(b,d) & j(d)

h(d,c,a) :- g(d,c) & g(c,a) & j(a)

h(d,c,b) :- g(d,c) & g(c,b) & j(b)

h(d,d,c) :- g(d,d) & g(d,c) & j(c)
```

Result:

```prolog
role(robot)
init(f(a,b))
init(e(a,b))

h(a,a,a) :- j(a)

h(a,a,b) :- j(b)

h(a,b,c) :- j(c)

h(a,b,d) :- j(d)

h(b,c,a) :- j(a)

h(b,c,b) :- j(b)

h(b,d,c) :- j(c)

h(b,d,d) :- j(d)

h(c,a,a) :- j(a)

h(c,a,b) :- j(b)

h(c,b,c) :- j(c)

h(c,b,d) :- j(d)

h(d,c,a) :- j(a)

h(d,c,b) :- j(b)

h(d,d,c) :- j(c)
```
<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/simplifier.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 = numberize(sc);
     playclock = numberize(pc);

     rules = simplifyrules(rules);

     library = definemorerules([], rs);
     roles = findroles(library);
     state = findinits(library);
     var active = findcontrol(state, library);
     var reward = parseInt(findreward(role, state, library));
     tree = makenode(state, active, reward, false);
     return 'ready'}
General Game Playing