General Game Playing
Grounding

Michael Genesereth
Logic Group
Stanford University

Grounding

Grounding is the process of *instantiating* game rules, i.e. replacing all variables by ground terms.

Example

```
row(M,R) :- cell(M,1,R) & cell(M,2,R) & cell(M,3,R)
col(N,R) :- cell(1,N,R) & cell(2,N,R) & cell(3,N,R)
```

```
row(1,x) :- cell(1,1,x) & cell(1,2,x) & cell(1,3,x)
row(2,x) :- cell(2,1,x) & cell(2,2,x) & cell(2,3,x)
row(3,x) :- cell(3,1,x) & cell(3,2,x) & cell(3,3,x)
row(1,o) :- cell(1,1,o) & cell(1,2,o) & cell(1,3,o)
row(2,o) :- cell(2,1,o) & cell(2,2,o) & cell(2,3,o)
row(3,o) :- cell(3,1,o) & cell(3,2,o) & cell(3,3,o)

col(1,x) :- cell(1,1,x) & cell(2,1,x) & cell(3,1,x)
col(2,x) :- cell(1,2,x) & cell(2,2,x) & cell(3,2,x)
col(3,x) :- cell(1,3,x) & cell(2,3,x) & cell(3,3,x)
col(1,o) :- cell(1,1,o) & cell(2,1,o) & cell(3,1,o)
col(2,o) :- cell(1,2,o) & cell(2,2,o) & cell(3,2,o)
col(3,o) :- cell(1,3,o) & cell(2,3,o) & cell(3,3,o)
```

Symbolizing

```
row(1,x) :- cell(1,1,x) & cell(1,2,x) & cell(1,3,x)
row(2,x) :- cell(2,1,x) & cell(2,2,x) & cell(2,3,x)
row(3,x) :- cell(3,1,x) & cell(3,2,x) & cell(3,3,x)
row(1,o) :- cell(1,1,o) & cell(1,2,o) & cell(1,3,o)
row(2,o) :- cell(2,1,o) & cell(2,2,o) & cell(2,3,o)
row(3,o) :- cell(3,1,o) & cell(3,2,o) & cell(3,3,o)
```

```
row1x :- cell11x & cell12x & cell13x
row2x :- cell21x & cell22x & cell23x
row3x :- cell31x & cell32x & cell33x
row1o :- cell11o & cell12o & cell13o
row2o :- cell21o & cell22o & cell23o
row3o :- cell31o & cell32o & cell33o
```

Why / Why Not?

Benefits

- Ground descriptions are simpler
- Interpreting ground descriptions can be more efficient

Disadvantages:

- Ground descriptions are more verbose / larger
- Need for different interpreter to realize benefits

- Grounding takes time
- Not all descriptions can be grounded in available time

Experimental Results

Alquerque to depth 4

```
Nodes: 9547
General Time:  10098 msec
Grounded Time:  1540 msec
```

CBNK to depth 4

```
Nodes: 5364
General Time:   3313 msec
Grounded Time:   294 msec
```

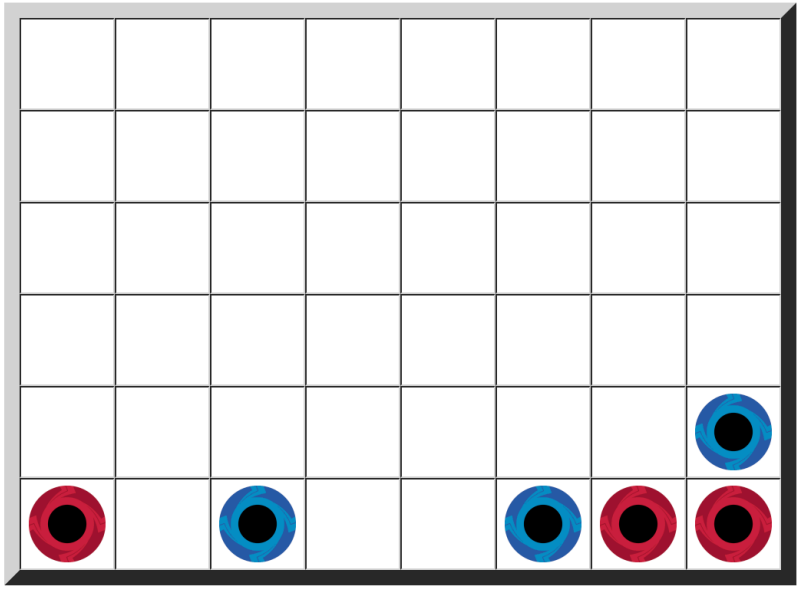
Connectfour to depth 4

```
Nodes: 4681
General Time:   6849 msec
Grounded Time:  1167 msec
```

Gamemaster

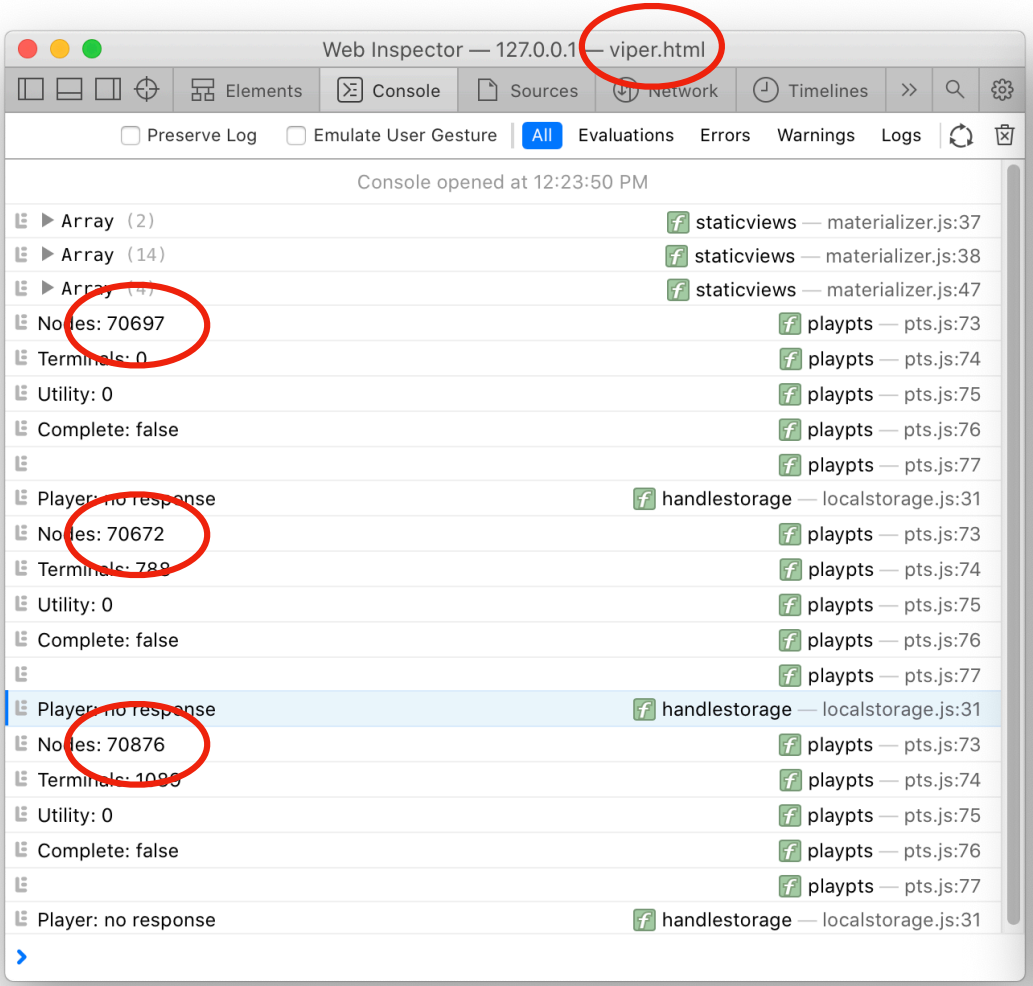
Sign In

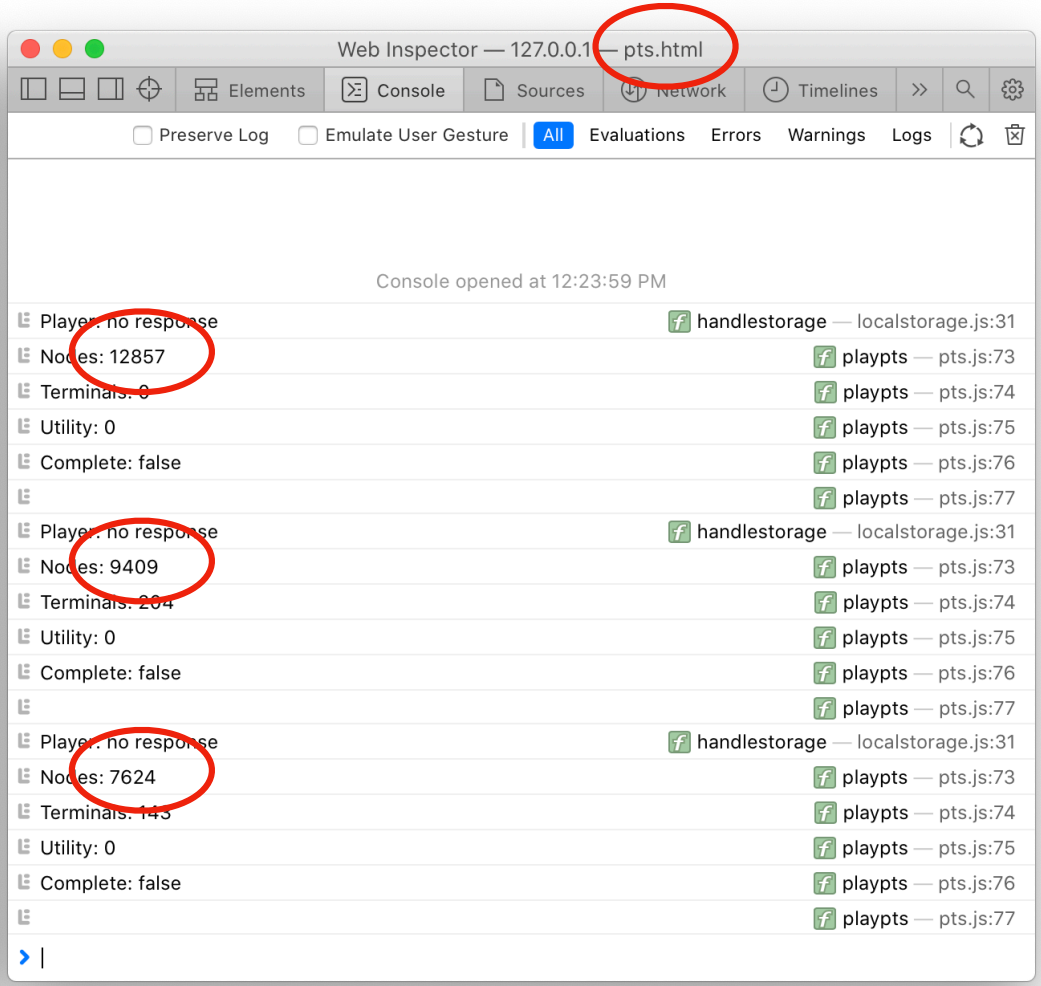
Protocol: localstorage	Game: connectfour
Strategy: stepper	Startclock: 10
Identifier: manager	Playclock: 10



Control: red

Roles	~~red~~	~~black~~
Players	viper	pts
Score	0	0
Errors	0	0


Web Inspector — 127.0.0.1 — viper.html
Elements
Console
Sources
Network
Timelines
Preserve Log
Emulate User Gesture
All
Evaluations
Errors
Warnings
Logs
Console opened at 12:23:50 PM
Array (2) staticviews — materializer.js:37
Array (14) staticviews — materializer.js:38
Array (4) staticviews — materializer.js:47
Nodes: 70697 playpts — pts.js:73
Terminals: 0 playpts — pts.js:74
Utility: 0 playpts — pts.js:75
Complete: false playpts — pts.js:76
playpts — pts.js:77
Player: no response handlestorage — localstorage.js:31
Nodes: 70672 playpts — pts.js:73
Terminals: 788 playpts — pts.js:74
Utility: 0 playpts — pts.js:75
Complete: false playpts — pts.js:76
playpts — pts.js:77
Player: no response handlestorage — localstorage.js:31
Nodes: 70876 playpts — pts.js:73
Terminals: 1089 playpts — pts.js:74
Utility: 0 playpts — pts.js:75
Complete: false playpts — pts.js:76
playpts — pts.js:77
Player: no response handlestorage — localstorage.js:31


Web Inspector — 127.0.0.1 — pts.html
Elements
Console
Sources
Network
Timelines
Preserve Log
Emulate User Gesture
All
Evaluations
Errors
Warnings
Logs
Console opened at 12:23:59 PM
Player: no response handlestorage — localstorage.js:31
Nodes: 12857 playpts — pts.js:73
Terminals: 0 playpts — pts.js:74
Utility: 0 playpts — pts.js:75
Complete: false playpts — pts.js:76
playpts — pts.js:77
Player: no response handlestorage — localstorage.js:31
Nodes: 9409 playpts — pts.js:73
Terminals: 264 playpts — pts.js:74
Utility: 0 playpts — pts.js:75
Complete: false playpts — pts.js:76
playpts — pts.js:77
Player: no response handlestorage — localstorage.js:31
Nodes: 7624 playpts — pts.js:73
Terminals: 143 playpts — pts.js:74
Utility: 0 playpts — pts.js:75
Complete: false playpts — pts.js:76
playpts — pts.js:77

Grounding

Procedure

(1) Compute **base propositions**.

(2) Compute **actions**.

(3) Compute **view propositions**. Iterate over view rules, matching subgoals to propositions, adding corresponding rule heads to propositions. Repeat to fixpoint.

(4) Compute **view rules**. Iterate over view rules matching subgoals to propositions, adding corresponding rules to output.

(5) Compute **operation rules**. Iterate over operation rules matching heads to actions and conditions to propositions, adding corresponding rules to output.

Computing Base Propositions

```
role(x)                    index(1)
role(o)                    index(2)
                           index(3)

base(cell(M,N,x)) :- index(M) & index(N)
base(cell(M,N,o)) :- index(M) & index(N)
base(cell(M,N,b)) :- index(M) & index(N)
base(control(R))  :- role(R)
```

```
cell(1,1,x)
cell(1,2,x)
    ...
control(x)
control(o)
```

Computing Actions

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

action(mark(M,N)) :- index(M) & index(N)
```

```
mark(1,1)
mark(1,2)
   ...
mark(3,1)
mark(3,3)
```

Adding View Propositions

```
cell(1,1,x)
cell(1,2,x)
    ...
control(x)
control(o)
```

```
row(M,R) :- cell(M,1,R) & cell(M,2,R) & cell(M,3,R)
```

```
row(1,x)
row(2,x)
row(3,x)
row(1,o)
row(2,o)
row(3,o)
```

Adding View Propositions

```
cell(1,1,x)     row(1,x)
cell(1,2,x)     row(2,x)
    ...         row(3,x)
control(x)      row(1,o)
control(o)      row(2,o)
                row(3,o)
```

```
line(R) :- row(M,R)
```

```
line(x)
line(o)
```

Instantiating View Rules

```
cell(1,1,x)     row(1,x)        ...
cell(1,2,x)     row(2,x)        ...
    ...         row(3,x)        ...
control(x)      row(1,o)        ...
control(o)      row(2,o)        ...
                row(3,o)        ...
```

+

```
line(R) :- row(M,R)
```

```
line(x) :- row(1,x)
line(x) :- row(2,x)
line(x) :- row(3,x)
line(o) :- row(1,o)
line(o) :- row(2,o)
line(o) :- row(3,o)
```

Instantiating Operation Rules

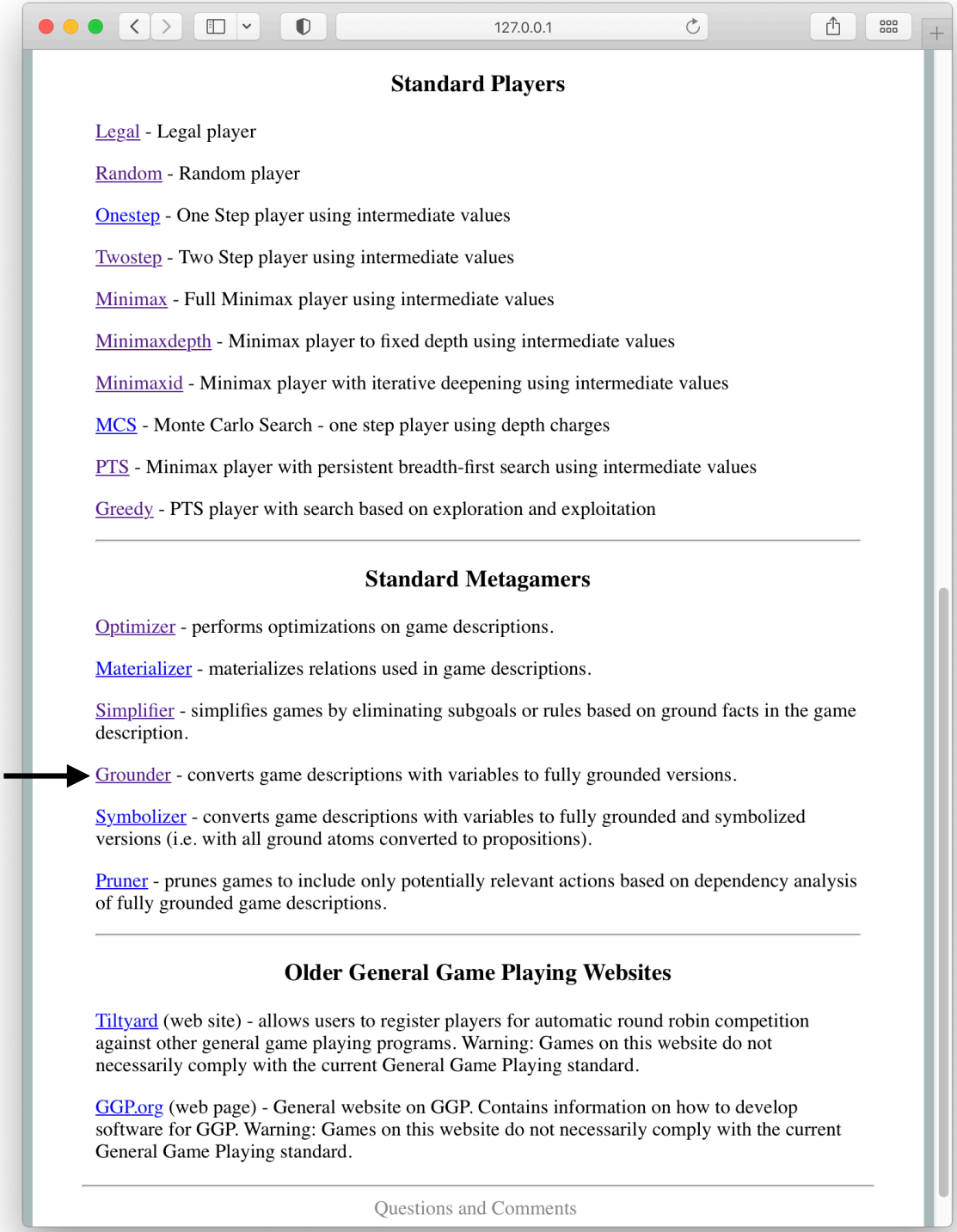
```
mark(1,1)
mark(1,2)
mark(1,3)
   ...
mark(3,1)
mark(3,2)
mark(3,3)
```

```
cell(1,1,x)     row(1,x)     ...
cell(1,2,x)     row(2,x)     ...
    ...         row(3,x)     ...
control(x)      row(1,o)     ...
control(o)      row(2,o)     ...
                row(3,o)     ...
```

+

```
mark(M,N) :: control(R) ==> cell(M,N,R) & ~cell(M,N,b)
```

```
mark(1,1) :: control(x) ==> cell(1,1,x) & ~cell(1,1,b)
mark(1,2) :: control(x) ==> cell(1,2,x) & ~cell(1,2,b)
                    ...
mark(3,2) :: control(o) ==> cell(3,2,o) & ~cell(3,2,b)
mark(3,3) :: control(o) ==> cell(3,3,o) & ~cell(3,3,b)
```


127.0.0.1
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.
Questions and Comments

Gamemaster

General Game Playing

Grounder

Game Description:

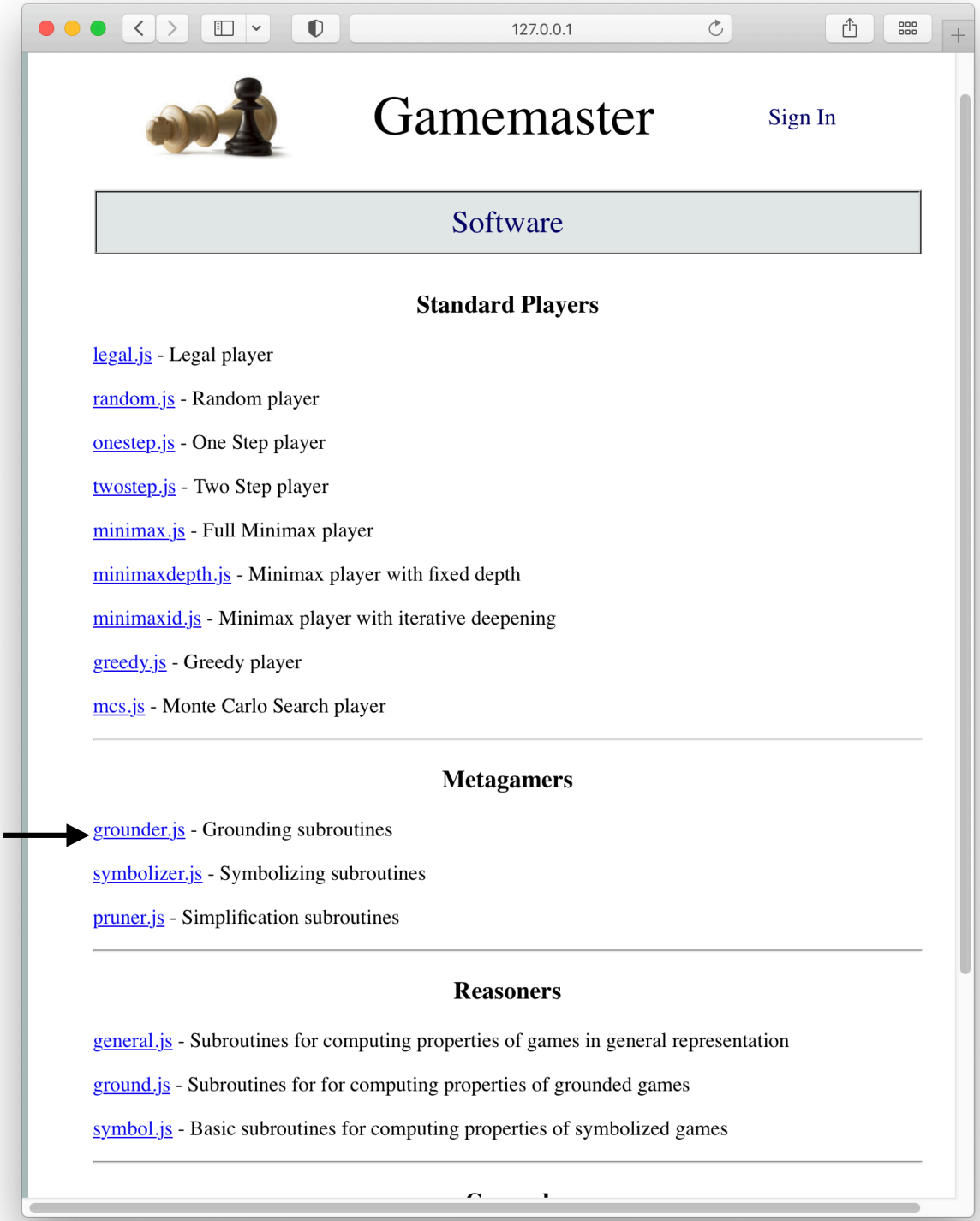
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% operations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

drop(X) :: control(R) & ~cell(X,1,red) & ~cell(X,1,black) ==> cell(X,1,R)

drop(X) ::
  control(R) & cell(X,Y1,P) & succ(Y1,Y2) & cellopen(X,Y2)
  ==> cell(X,Y2,R)

drop(X) :: control(red) ==> ~control(red) & control(black)
drop(X) :: control(black) ==> ~control(black) & control(red)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% goal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal(red,100) :- line(red)
```

Count Bases | Count Actions | Count Facts | Count Rules

Ground Bases | Ground Actions | Ground Facts | Ground Rules

Output: 259 milliseconds

```
role(red)
role(black)
base(cell(1,1,red)) :- column(1) & height(1) & role(red)
base(cell(1,1,black)) :- column(1) & height(1) & role(black)
base(cell(1,2,red)) :- column(1) & height(2) & role(red)
base(cell(1,2,black)) :- column(1) & height(2) & role(black)
base(cell(1,3,red)) :- column(1) & height(3) & role(red)
base(cell(1,3,black)) :- column(1) & height(3) & role(black)
base(cell(1,4,red)) :- column(1) & height(4) & role(red)
base(cell(1,4,black)) :- column(1) & height(4) & role(black)
base(cell(1,5,red)) :- column(1) & height(5) & role(red)
base(cell(1,5,black)) :- column(1) & height(5) & role(black)
base(cell(1,6,red)) :- column(1) & height(6) & role(red)
base(cell(1,6,black)) :- column(1) & height(6) & role(black)
base(cell(2,1,red)) :- column(2) & height(1) & role(red)
base(cell(2,1,black)) :- column(2) & height(1) & role(black)
base(cell(2,2,red)) :- column(2) & height(2) & role(red)
base(cell(2,2,black)) :- column(2) & height(2) & role(black)
```


127.0.0.1
Gamemaster
Sign In
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 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/gameplaying/
pts.js'></script>

<script src='http://gamemaster.stanford.edu/reasoning/
general.js'></script>
```

Old Code

```
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'}
```

New Code

```
function start (r,rs,sc,pc)
 {role = r;
  rules = rs.slice(1)
  startclock = parseInt(sc);
  playclock = parseInt(pc);

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

  library = definemorerules([],rules);
  roles = findroles(library);
  state = findinits(library);
  return 'ready'}
```

← *produces ground rules*

Simplifier

```
init(cell(M,N,b)) :- index(M) & index(N)
```

```
init(cell(1,1,b)) :- index(1) & index(1)
init(cell(1,2,b)) :- index(1) & index(2)
                  ...
init(cell(3,2,b)) :- index(3) & index(2)
init(cell(3,3,b)) :- index(3) & index(3)
```

```
init(cell(1,1,b))
init(cell(1,1,b))
        ...
init(cell(1,1,b))
init(cell(1,1,b))
```

Simplifier

```
init(cell(M,N,b)) :- index(M) & index(N)
```

```
init(cell(1,1,b)) :- index(1) & index(1)
init(cell(1,2,b)) :- index(1) & index(2)
                  ...
init(cell(3,2,b)) :- index(3) & index(2)
init(cell(3,3,b)) :- index(3) & index(3)
```

```
init(cell(1,1,b))
init(cell(1,1,b))
        ...
init(cell(1,1,b))
init(cell(1,1,b))
```

Executing Grounded Games

Playing Ground / Symbolized Games

Good News

General interpreter works on ground rules.

Bad News

It is often *slower*!

What?

Why?

Experimental Results

Game	Depth	Result	Terms	Nodes	Normal Normal
buttonsandlights	6	100	729	1093	20
tictactoe7	7	50	3468	7332	650
tictoctoe7	7	50	5040	13700	550
connectfour	4	0	4096	4681	2700
nineboard	4	50	6120	6914	9300

http://gamemaster.stanford.edu/players/egghead/experimental_results.html

Experimental Results

Game	Depth	Result	Terms	Nodes	Normal Normal	Normal Ground
buttonsandlights	6	100	729	1093	20	25
tictactoe7	7	50	3468	7332	650	800
tictoctoe7	7	50	5040	13700	550	800
connectfour	4	0	4096	4681	2700	3200
nineboard	4	50	6120	6914	9300	38000

Analysis

Original Rule

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

$$1 + (n\text{^}2+2n) + n*((n\text{^}2+2n) + 1*(n\text{^}2+2n)) =$$
$$2n\text{^}3 + 5n\text{^}2 + 2n + 1$$

Ground Rules

```
goal(a,a) :- p(a) & q(a) & r(a,a)
                    ...
goal(c,c) :- p(c) & q(c) & r(c,c)
```

$$n\text{^}2 * (1 + 3*(n\text{^}2 + 2n)) =$$
$$3n\text{^}4 + 6n\text{^}3 + n\text{^}2$$

Number of Comparisons

Original Rule

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

$$1 + (n\text{^}2+2n) + n*((n\text{^}2+2n) + 1*(n\text{^}2+2n)) =$$
$$2n\text{^}3 + 5n\text{^}2 + 2n + 1$$
106 unifications for $n = 3$

Ground Rules

```
goal(a,a) :- p(a) & q(a) & r(a,a)
                    ...
goal(c,c) :- p(c) & q(c) & r(c,c)
```

$$n\text{^}2 * (1 + 3*(n\text{^}2 + 2n)) =$$
$$3n\text{^}4 + 6n\text{^}3 + n\text{^}2$$
414 unifications for $n = 3$

Specialized Player for Ground Games

Our *general interpreter* must allow for *unification* of expressions with variables. Unification is expensive.

```
g(X,Y) :- p(X,c) & q(X,Y)
p(a,Y) :- m(Y) & n(Y)
```

Specialized Player for Ground Games

Our *general interpreter* must allow for *unification* of expressions with variables. Unification is expensive.

```
g(X,Y) :- p(X,c) & q(X,Y)
p(a,Y) :- m(Y) & n(Y)
```

In a *ground interpreter*, unifications are replaced by *string equality*. Equality testing is *relatively* inexpensive.

```
g(a,b) :- p(a,c) & q(a,b)
p(a,c) :- m(c) & n(c)
```

Implementation

```
function findcontrol (facts,rules)
 {return compfindx('X',seq('control','X'),facts,rules)}

function findlegalp (move,facts,rules)
 {return compfindp(seq('legal',move),facts,rules)}

function findlegalx (facts,rules)
 {return compfindx('X',seq('legal','X'),facts,rules)}

function findlegals (facts,rules)
 {return compfinds('X',seq('legal','X'),facts,rules)}

function findreward (role,facts,rules)
 {var v = compfindx('R',seq('goal',role,'R'),facts,rules);
  if (v) {return v};
  return "0"}

function findterminalp (facts,rules)
 {return compfindp('terminal',facts,rules)}
```

General Interpreter

```
function findcontrol (facts,rules)
 {return grounditem('control',facts,rules)}

function findlegalp (move,facts,rules)
 {return groundfindp(seq('legal',move),facts,rules)}

function findlegalx (facts,rules)
 {return grounditem('legal',facts,rules)}

function findlegals (facts,rules)
 {return grounditems('legal',facts,rules)}

function findreward (role,facts,rules)
 {var v = groundvalue('goal',role,facts,rules);
  if (v) {return v};
  return 0}

function findterminalp (facts,rules)
 {return groundfindp('terminal',facts,rules)}
```

Ground Interpreter

General:

```
function compfindbackground (n,x,p,pl,al,cont,results,facts,rules)
 {for (var i=0; i<facts.length; i++)
      {var bl = {};
       var ol = seq();
       if (vnifyp(facts[i],bl,p,al,ol))
          {var ans = basesomeexit(n,x,pl,al,cont,results,facts,rules);
           backup(ol);
           if (answer) {return answer}}};
  return false}
```

Ground:

```
function groundfindbackground (p,facts,rules)
 {for (var i=0; i<facts.length; i++)
      {if (equalp(facts[i],p)) {return true}};
  return false}
```

Ground Interpreter

General:

```
function basefindbackground (n,x,p,pl,al,cont,results,facts,rules)
 {for (var i=0; i<facts.length; i++)
      {var bl = {};
       var ol = seq();
       if (vnifyp(facts[i],bl,p,al,ol))
          {var ans = basesomeexit(n,x,pl,al,cont,results,facts,rules);
           backup(ol);
           if (answer) {return answer}}};
  return false}
```

Ground:

```
function groundfindbackground (p,facts,rules)
 {for (var i=0; i<facts.length; i++)
      {if (equalp(facts[i],p)) {return true}};
  return false}
```

Gamemaster

Sign In

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 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/gameplaying/
pts.js'></script>

<script src='http://gamemaster.stanford.edu/reasoning/
ground.js'></script>
```

minimaxid

```
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'}
```

minimaxid with Grounding

```
function start (r,rs,sc,pc)
 {role = r;
  rules = rs.slice(1)
  startclock = parseInt(sc);
  playclock = parseInt(pc);

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

  library = definemorerules([],rules);
  roles = findroles(library);
  state = findinits(library);
  return 'ready'}
```

← *produces ground rules*

Experimental Results

Game	Depth	Result	Terms	Nodes	Normal Normal	Normal Ground
buttonsandlights	6	100	729	1093	20	25
tictactoe7	7	50	3468	7332	650	800
tictoctoe7	7	50	5040	13700	550	800
connectfour	4	0	4096	4681	2700	3200
nineboard	4	50	6120	6914	9300	38000

Experimental Results

Game	Depth	Result	Terms	Nodes	Normal Normal	Normal Ground	Ground Ground
buttonsandlights	6	100	729	1093	20	25	9
tictactoe7	7	50	3468	7332	650	800	250
tictoctoe7	7	50	5040	13700	550	800	240
connectfour	4	0	4096	4681	2700	3200	870
nineboard	4	50	6120	6914	9300	38000	10500

Symbolizing

Symbolizing

```
row(1,x) :- cell(1,1,x) & cell(1,2,x) & cell(1,3,x)
row(2,x) :- cell(2,1,x) & cell(2,2,x) & cell(2,3,x)
row(3,x) :- cell(3,1,x) & cell(3,2,x) & cell(3,3,x)
row(1,o) :- cell(1,1,o) & cell(1,2,o) & cell(1,3,o)
row(2,o) :- cell(2,1,o) & cell(2,2,o) & cell(2,3,o)
row(3,o) :- cell(3,1,o) & cell(3,2,o) & cell(3,3,o)
```

```
row1x :- cell11x & cell12x & cell13x
row2x :- cell21x & cell22x & cell23x
row3x :- cell31x & cell32x & cell33x
row1o :- cell11o & cell12o & cell13o
row2o :- cell21o & cell22o & cell23o
row3o :- cell31o & cell32o & cell33o
```

Specialized Player for Symbolized Games

All the benefits of ground games *and more*.

Once symbolized, propositions are represented as strings, and we can represent states using *associative arrays* (logarithmic access and update) rather than *lists* (linear access and update).

Benefit: Especially valuable for games where states characterized by many propositions, e.g. nineboard.

Benefit: In Monte Carlo depth charges, there is no need to copy states; we can just change in place.

Implementation

Ground:

```
function findinits (rules)
 {return basefinds('P',seq('init','P'),seq(),rules)}
```

Symbol:

```
function findinits (rules)
 {var state = basefinds('P',seq('init','P'),[],rules));
  return makestate(state)}

function makestate (facts)
 {var newstate = {};
  for (var i=0; i<facts.length; i++)
      {newstate[facts[i]] = true};
  return newstate}
```

Implementation

Ground:

```
function groundfindbackground (p,facts,rules)
 {for (var i=0; i<facts.length; i++)
      {if (equalp(facts[i],p)) {return true}};
  return false}
```

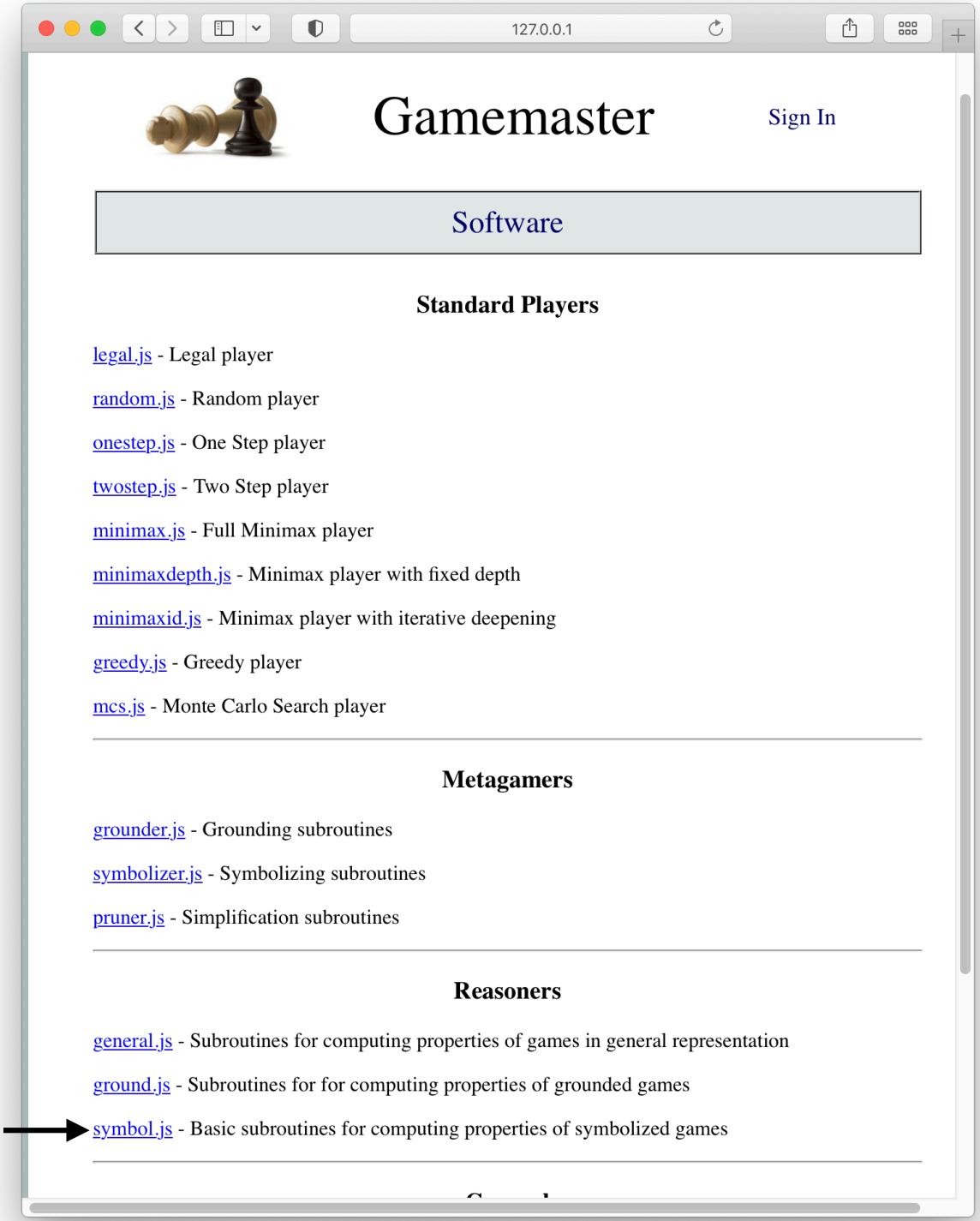
Symbol:

```
function groundfindbackground (p,facts,rules)
 {return (facts[p]===true)}
```

Implementation

```
function simulate (move,state,rules)
 {var deltas = groundexpand(move,state,rules);
  var newstate = Object.assign({},state);
  for (var i=0; i<deltas.length; i++)
      {var delta = deltas[i];
       if (!symbolp(delta) && delta[0]==='not')
          {delete(newstate[delta[1]])}};
  for (var i=0; i<deltas.length; i++)
      {var delta = deltas[i];
       if (symbolp(delta))
          {newstate[delta] = true; continue};
       if (delta[0]==='not') {continue};
       newstate[delta] = true};
  return newstate}
```

No need to copy on depth charges!!


127.0.0.1
Gamemaster
Sign In
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 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/gameplaying/
pts.js'></script>

<script src='http://gamemaster.stanford.edu/reasoning/
symbol.js'></script>
```

minimaxid with Grounding

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

function play (move)
 {if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  move = playminimaxid(role);
  return move}
```

minimaxid with Symbolization

```
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);
  library = definemorerules([],rules);
  roles = findroles(library);
  state = findinits(library);
  return 'ready'}

function play (move)
 {move = symbolizeatom(move);
  if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  move = playminimaxid(role);
  move = unsymbolizeatom(move);
  return move}
```

Experimental Results

Game	Depth	Result	Terms	Nodes	Normal Normal	Normal Ground	Ground Ground
buttonsandlights	6	100	729	1093	20	25	9
tictactoe7	7	50	3468	7332	650	800	250
tictoctoe7	7	50	5040	13700	550	800	240
connectfour	4	0	4096	4681	2700	3200	870
nineboard	4	50	6120	6914	9300	38000	10500

Experimental Results

Game	Depth	Result	Terms	Nodes	Normal Normal	Normal Ground	Ground Ground	Symbol Symbol
buttonsandlights	6	100	729	1093	20	25	9	8
tictactoe7	7	50	3468	7332	650	800	250	130
tictoctoe7	7	50	5040	13700	550	800	240	150
connectfour	4	0	4096	4681	2700	3200	870	600
nineboard	4	50	6120	6914	9300	38000	10500	960

Potentially BIG Problem

Why / Why Not?

Benefits

- Ground descriptions are simpler
- Interpreting ground descriptions can be more efficient

Disadvantages:

- Ground descriptions are more verbose / larger
- Need for different interpreter to realize benefits

- Grounding takes time
- Not all descriptions can be grounded in available time

Game	Bases	Actions	Facts	Rules	Milliseconds
alquerque	129	193	996	4470	107
badconnectfour	98	8	310	262645	1670
badskirmish	2100	2048	16596	153534	116378
badtictactoe	29	9	123	5380993	78024
battleofnumbers	305	144	1139	20924	917
bestbuttonsandlights	90	27	280	413	8
besttictactoe	56	18	218	258	12
breakthrough	130	308	935	2796	117
buttonsandlights	11	3	42	63	2
capturetheflag	335	237	1607	68106	859
cbnk	142	211	1348	1954	128
chinesecheckers1	83	268	962	6545	277
chinesecheckers3	176	268	1543	20590	1030
connectfour	98	8	308	573	266
crusade	214	339	1639	4324	180
donttouch	50	16	188	304	10
dualconnectfour	194	16	598	799	135
hamilton	81	20	347	1486	25
hex	6968	163	14722	632994	33452
hex7x7	2648	99	5770	153994	5354
hunter	77	40	361	1564	26
jointbuttonsandlights	31	10	101	155	4
knights	96	160	742	8089	172
knightstour	92	124	682	4587	123
knightthrough	130	336	1354	3383	126
knightthroughsmall	74	160	694	1663	50
leafymcfallface	149	49	475	5561	89
lightboard	65	65	272	272	6
majorities	644	81	1986	7967	3165
multiplebuttonsandlights	90	27	265	431	10
multipleknightthrough	179	229	1070	2067	72
multiplesuicide	245	81	939	1156	109
multipletictactoe	245	81	939	1156	119
nineboardtictactoe	254	81	966	1495	115
parallelbuttonsandlights	23	7	86	110	3
parallelknightthrough	179	229	1070	2067	72
pentago	76	44	455	814	97
rainbow	25	24	119	4278	104
reversi	74	36	2942	4411	1524
simplebuttonsandlights	90	27	241	407	8
simplelightboard	10	10	47	47	1
simpleswitches	81	81	269	280	8
skirmish	2174	2048	16904	151626	9590
suicide	29	9	123	163	7
sukoshi	36	27	172	245	7
switches	81	81	341	352	10
threepuzzle	115	4	254	591	10
tictactoe	29	9	123	163	7
tictactoe3	29	9	123	163	5
tictactoe5	29	9	123	163	5
tictactoe7	29	9	123	163	5
trifecta	29	9	123	163	5
ttcc4	215	313	2337	7474	1073

Solution

Solution:

Pass deadline into grounder based on start clock.

Terminate if deadline passed.

If no grounding, use original.

Responsibility:

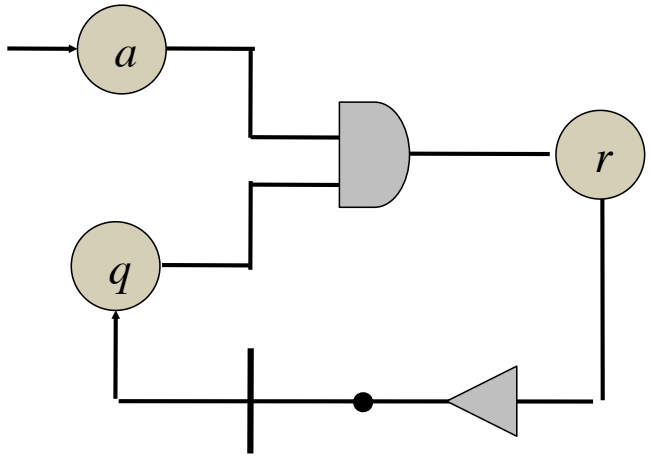
Grounder on website does not do this.

Your job (if you want to use it).

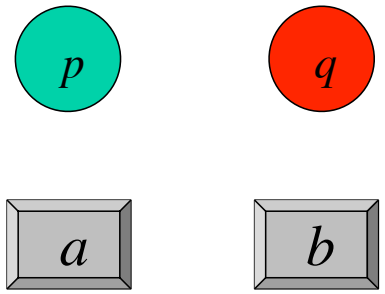
Propositional Nets

Grounded State Description

```
row1x :- cell11x & cell12x & cell13x
row2x :- cell21x & cell22x & cell23x
row3x :- cell31x & cell32x & cell33x
row1o :- cell11o & cell12o & cell13o
row2o :- cell21o & cell22o & cell23o
row3o :- cell31o & cell32o & cell33o
```

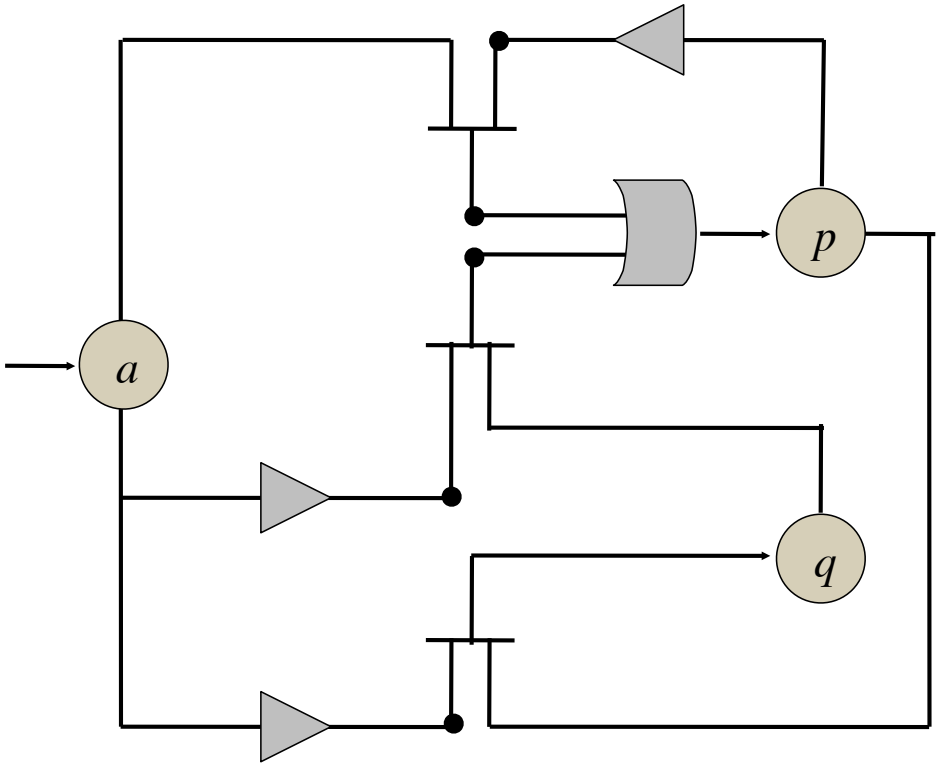


Simple Buttons and Lights



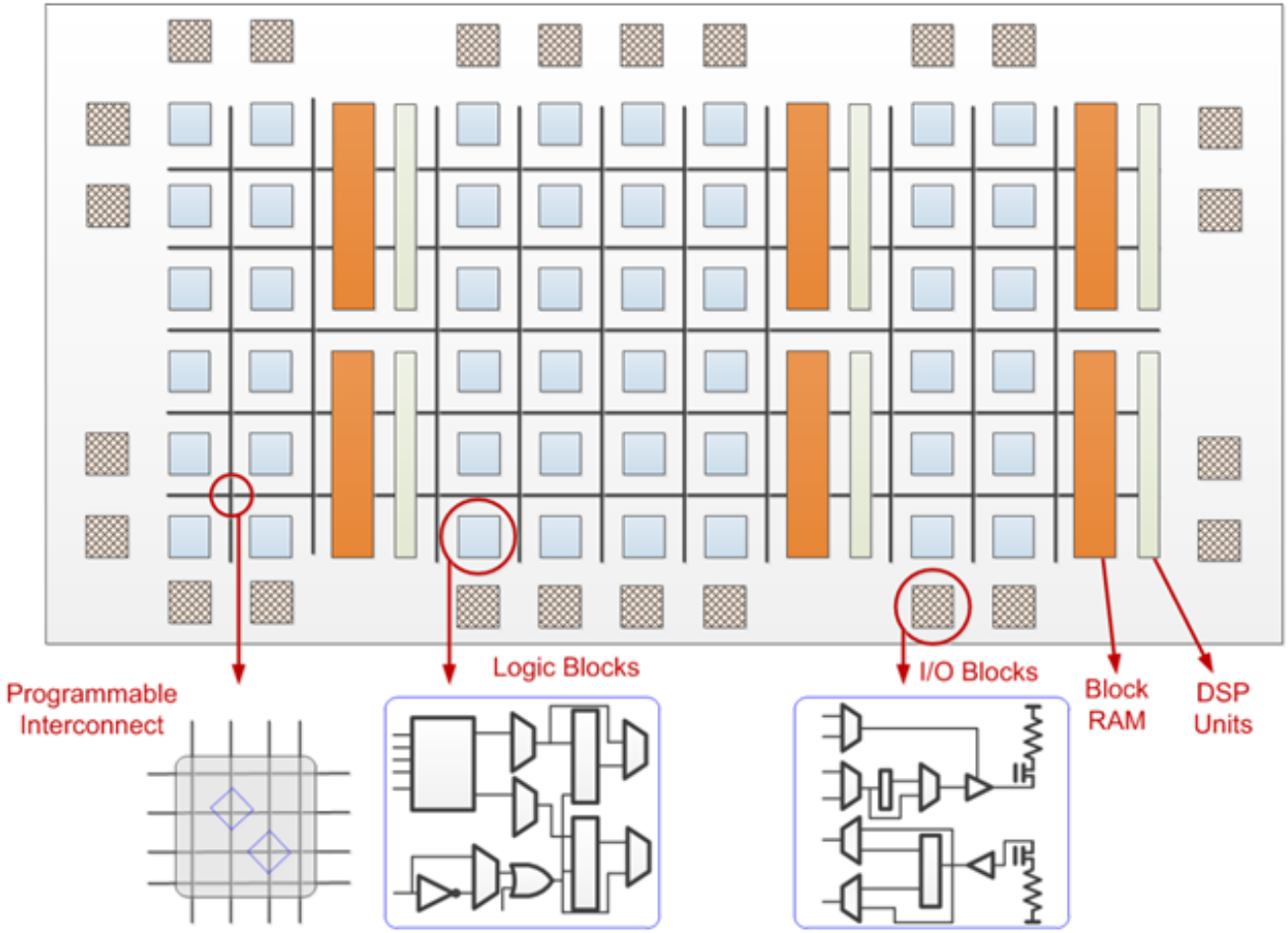
Pressing button a toggles p.
Pressing button b interchanges p and q.

Simple Buttons and Lights Propnet



Field Programmable Gate Arrays (FPGAs)

Field Programmable Gate Arrays (FPGAs)




GENERAL
GAME
PLAYING