General Game Playing

Game Factoring

Michael Genesereth
Logic Group
Stanford University

Game Factoring

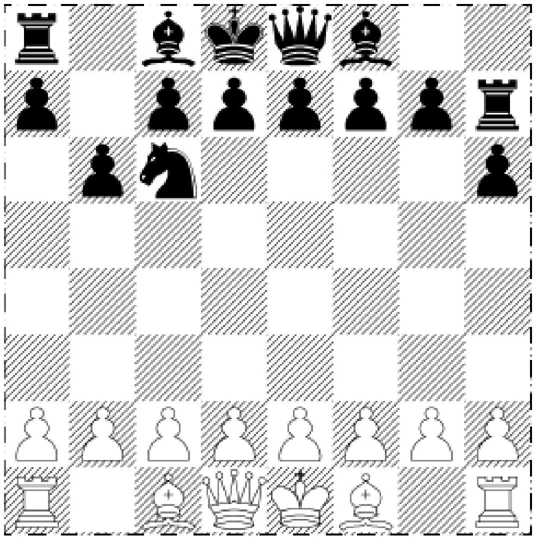
Game factoring (aka *game decomposition*) is the process of discovering independent subgames inside larger games.

Techniques so far provide polynomial improvement
Factoring can provide exponential improvement

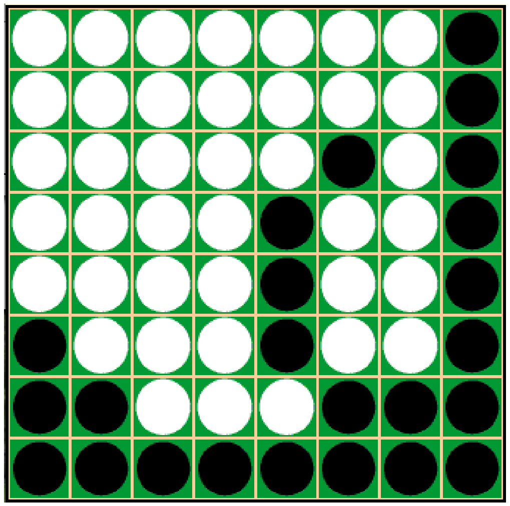
Trade-off - cost of factoring vs savings
    Sometimes cost proportional to size of description
    Sometimes savings proportional to size of game tree

Hodgepodge

Hodgepodge = Chess + Othello



Branching factor: *a*

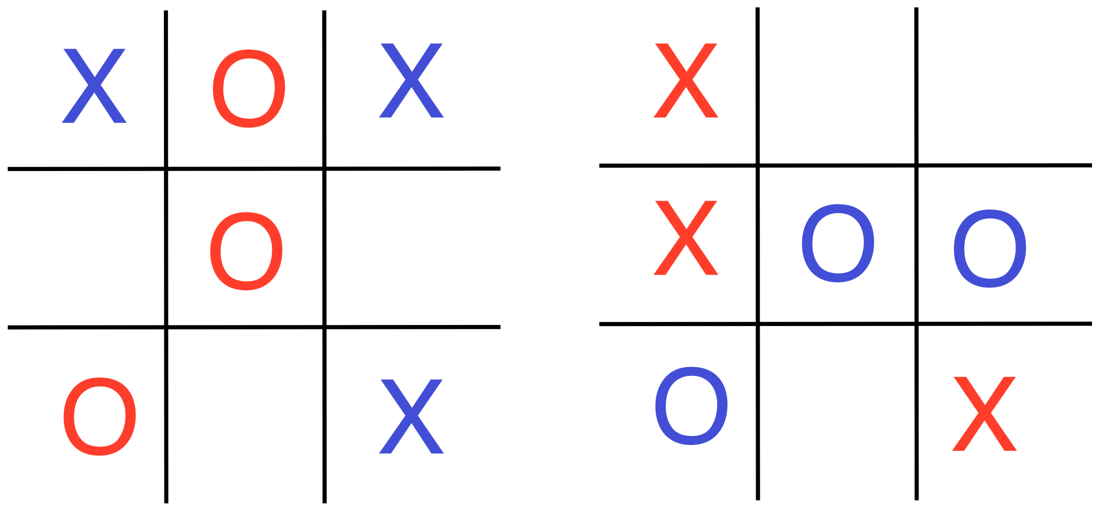


Branching factor: *b*

Analysis of joint game:

- Branching factor as given to players: $a*b$
- Fringe of tree at depth n as given: $(a*b)^n$
- Fringe of tree at depth n factored: a^n+b^n

Double Tic Tac Toe



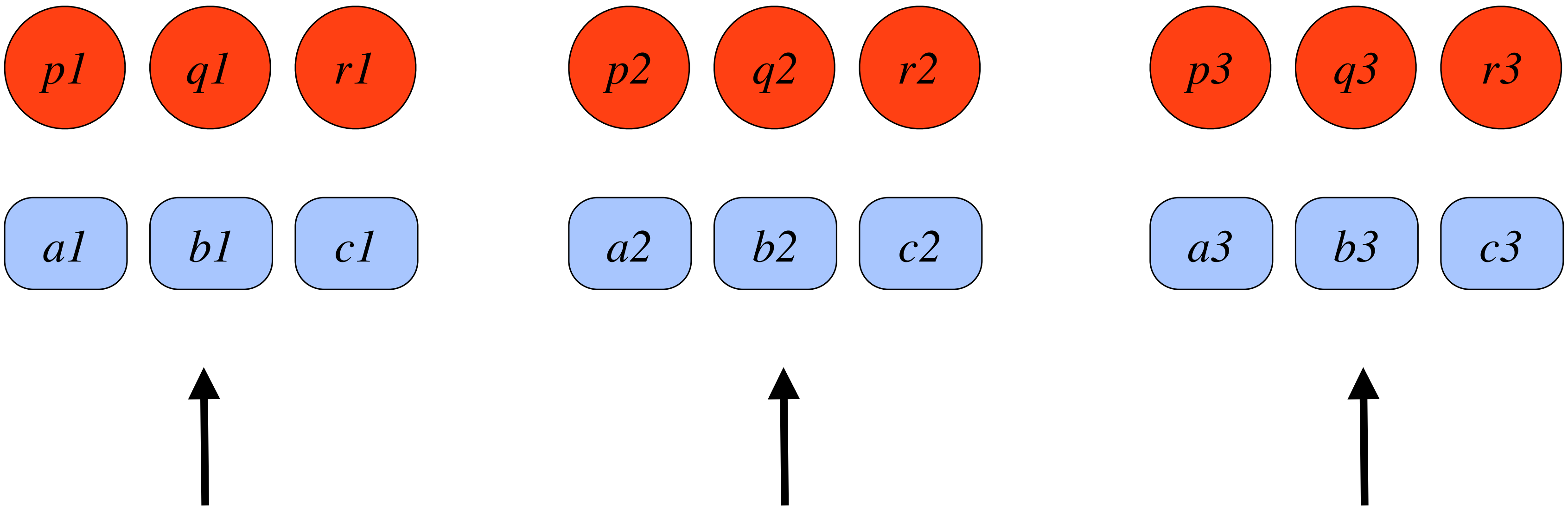
Analysis of joint game:

Branching factor: 81, 64, 49, 36, 25, 16, 9, 4, 1

Branching factor: 9, 8, 7, 6, 5, 4, 3, 2, 1

Best Buttons and Lights

Overall terminal and goals defined as disjunctions of individual terminals and goals.



Terminating one group terminates entire game.

Playing Factored Best Games

Technique:

(1) Play each factor to get a move and a score.

(2) Select subgame/move with best score.

Cost:

$\mathit{cost}(\text{game}_1) + \ldots + \mathit{cost}(\text{game}_k) \ll \mathit{cost}(\text{game})$

Nodes searched by Minimax on 9 board BB&L

with factoring: 3276

without factoring: 14348907

Grounding

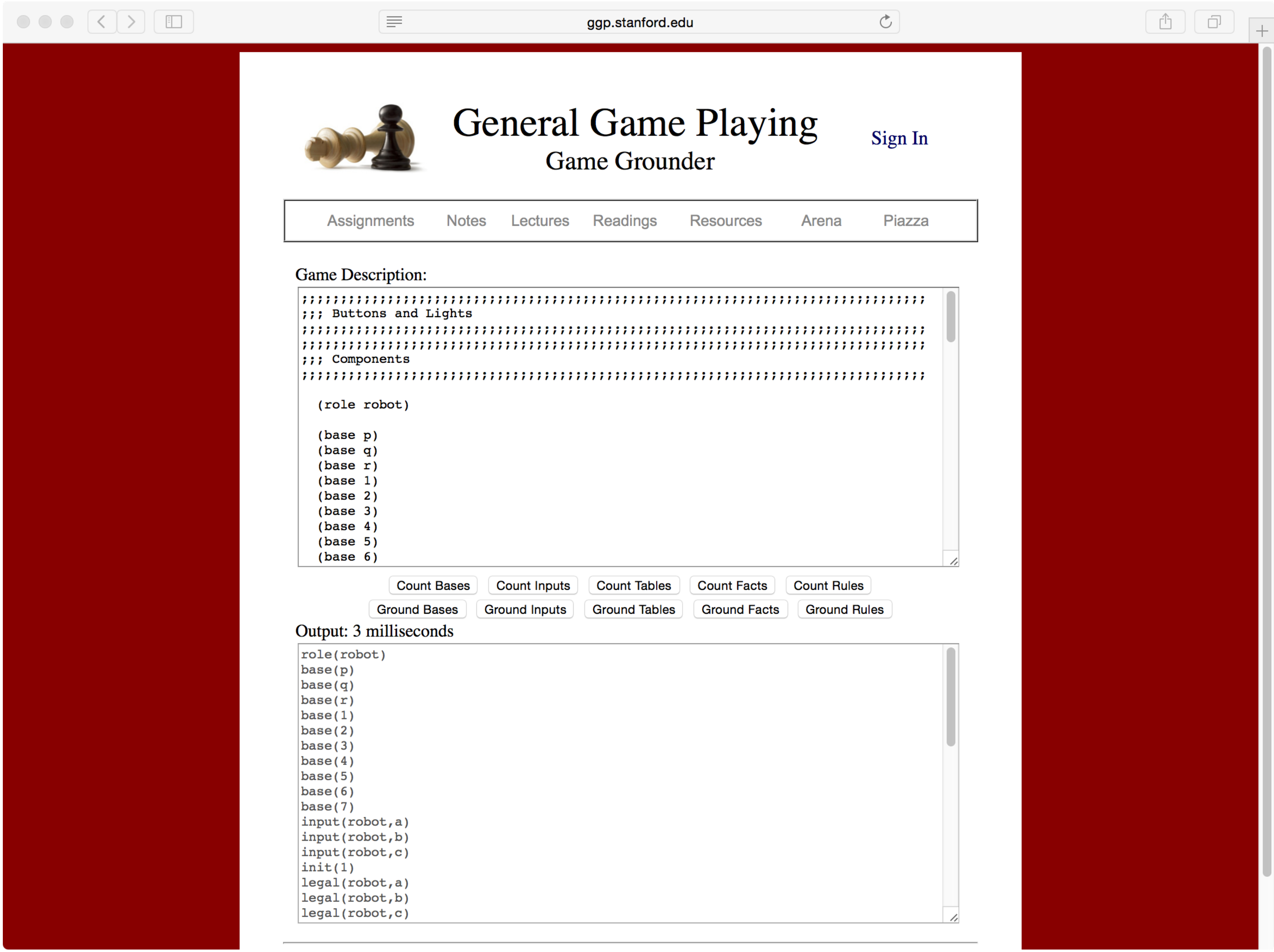
Grounding

```
next(p(X)) :- does(robot,a(X)) & ~true(p(X))
next(p(X)) :- does(robot,b(X)) & true(q(X))
next(p(X)) :- true(p(X)) & ~does(robot,a(X)) & ~does(robot,b(X))
```

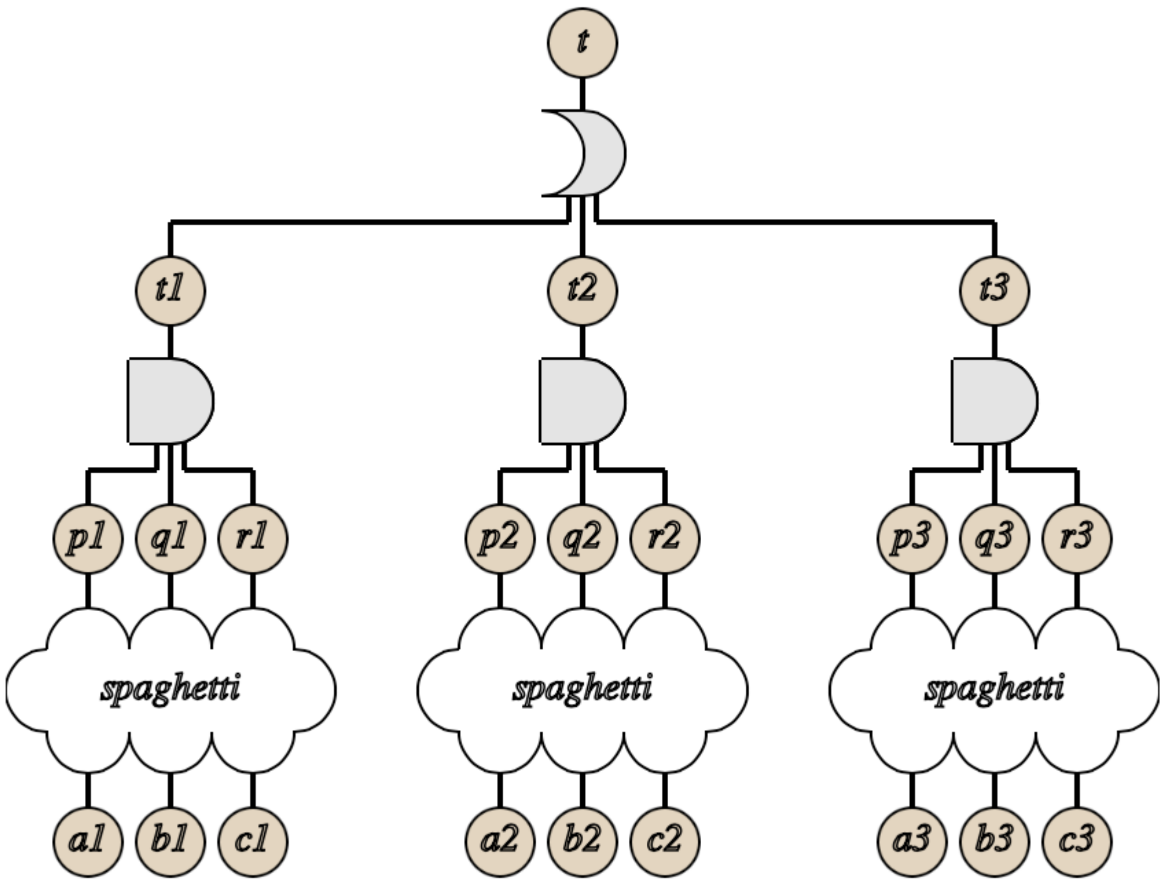
↓

```
next(p(1)) :- does(robot,a(1)) & ~true(p(1))
next(p(1)) :- does(robot,b(1)) & true(q(1))
next(p(1)) :- true(p(1)) & ~does(robot,a(1)) & ~does(robot,b(1))

next(p(2)) :- does(robot,a(2)) & ~true(p(2))
next(p(2)) :- does(robot,b(2)) & true(q(2))
next(p(2)) :- true(p(2)) & ~does(robot,a(2)) & ~does(robot,b(2))

next(p(3)) :- does(robot,a(3)) & ~true(p(3))
next(p(3)) :- does(robot,b(3)) & true(q(3))
next(p(3)) :- true(p(3)) & ~does(robot,a(3)) & ~does(robot,b(3))
```

Grounding Tool

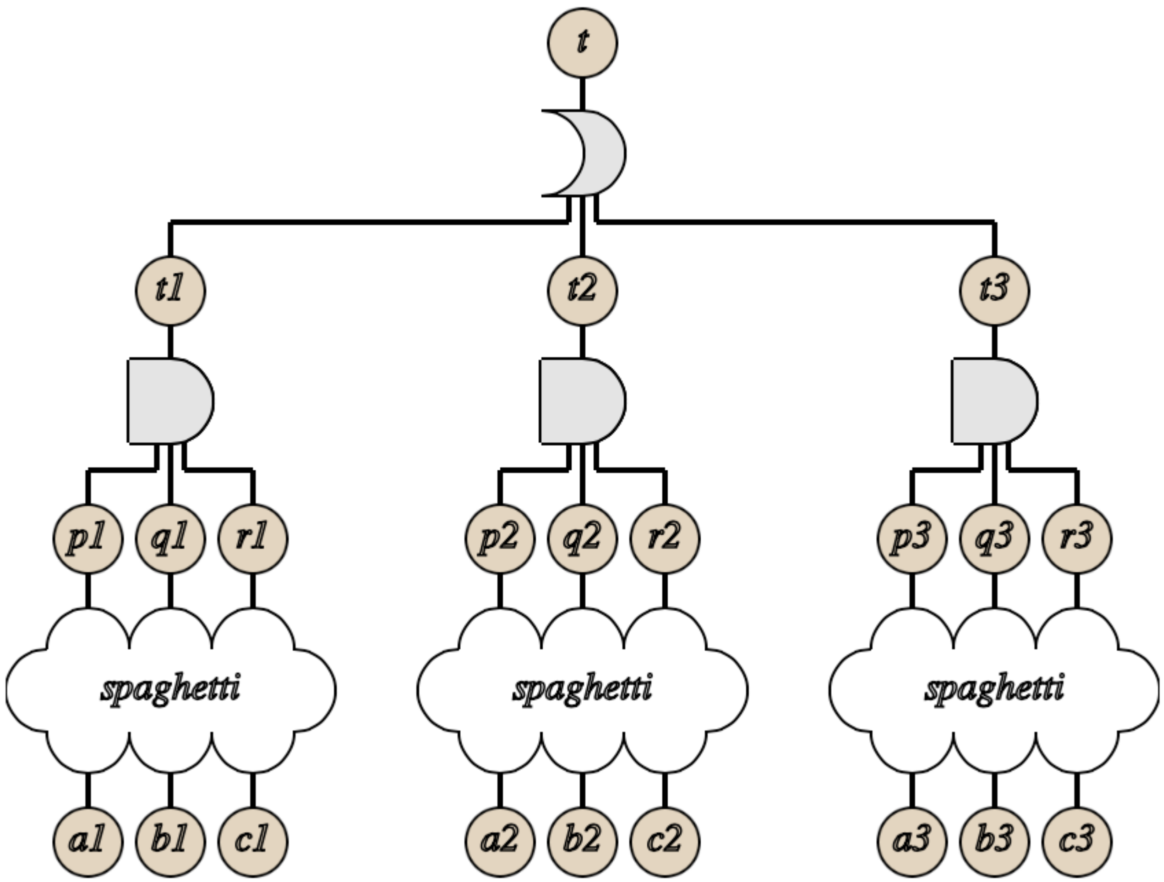


Propnet for Best Buttons and Lights



Factoring

Propnet for Best Buttons and Lights



Base and Input Propositions

Bases:

```
base(p(X)) :- index(X)
base(q(X)) :- index(X)
base(r(X)) :- index(X)
index(1)
index(2)
index(3)
```

Inputs:

```
input(robot,a(X)) :- index(X)
input(robot,b(X)) :- index(X)
input(robot,c(X)) :- index(X)
```

Results:

```
true(p(1))
true(p(2))
true(p(3))
true(q(1))
true(q(2))
true(q(3))
true(r(1))
true(r(2))
true(r(3))
```

```
does(robot(a(1)))
does(robot(a(2)))
does(robot(a(3)))
does(robot(b(1)))
does(robot(b(2)))
does(robot(b(3)))
does(robot(c(1)))
does(robot(c(2)))
does(robot(c(3)))
```

Compute Residues

Compute all `true(`*p*`)` and `does(`*role*`,`*a*`)` factoids needed to compute `next(`*q*`)` and `legal(`*role*`,`*a*`)` for each base proposition *q* and each input *a*. These sets are called *residues*.

Rules for `next(p(1))`:

```
next(p(1)) :- does(robot,a(1)) & ~true(p(1))
next(p(1)) :- does(robot,b(1)) & true(q(1))
next(p(1)) :- true(p(1)) & ~does(robot,a(1)) & ~does(robot,b(1))
```

Residue for `next(p(1))`:

```
{true(p(1)), true(q(1), does(robot,a(1))), does(robot,b(1))}
```

Merge Overlapping Residues

```
{true(p(1)), true(q(1),
 does(robot,a(1))), does(robot,b(1))}
{true(q(1)), true(p(1)), true(r(1)),
 does(robot,b(1))), does(robot,c(1))}
{true(r(1)), true(q(1)), does(robot,c(1))}

{true(p(2)), true(q(2),
 does(robot,a(2))), does(robot,b(2))}
{true(q(2)), true(p(2)), true(r(2)),
 does(robot,b(2))), does(robot,c(2))}
{true(r(2)), true(q(2)), does(robot,c(2))}

{step(1), step(2),..., step(7)}


{true(p(1)), true(q(1), true(r(1))
 does(robot,a(1))), does(robot,b(1), does(robot,c(1)))}
{true(p(2)), true(q(2), true(r(2))
 does(robot,a(2))), does(robot,b(2), does(robot,c(2)))}
{step(1), step(2),..., step(7)}
```

Disjoint Action Sets

```
{does(robot,a(1))), does(robot,b(1), does(robot,c(1)))}

{does(robot,a(2))), does(robot,b(2), does(robot,c(2)))}

{does(robot,a(3))), does(robot,b(3), does(robot,c(3)))}
```

Rules Defining Legality

```
legal(robot,a(1)) :- index(1)
legal(robot,b(1)) :- index(1)
legal(robot,c(1)) :- index(1)
legal(robot,a(2)) :- index(2)
legal(robot,b(2)) :- index(2)
legal(robot,c(2)) :- index(2)
legal(robot,a(3)) :- index(3)
legal(robot,b(3)) :- index(3)
legal(robot,c(3)) :- index(3)
```

Partition into Different Subgames

```
legal(robot,a(1)) :- index(1)
legal(robot,b(1)) :- index(1)
legal(robot,c(1)) :- index(1)

legal(robot,a(2)) :- index(2)
legal(robot,b(2)) :- index(2)
legal(robot,c(2)) :- index(2)

legal(robot,a(3)) :- index(3)
legal(robot,b(3)) :- index(3)
legal(robot,c(3)) :- index(3)
```

Playing Factored Best Games

Technique:

(1) Play each factor to get a move and a score.

(2) Select subgame/move with best score.

Cost:

$cost(\text{game}_1) + \ldots + cost(\text{game}_k) \ll cost(\text{game})$

Nodes searched by Minimax on 9 board BB&L

with factoring: 3276

without factoring: 14348907

Nine Board Best Buttons and Lights

Depth	Unfactored	Factored	Time (msec)
1	28	36	14
2	757	117	45
3	20,440	360	140
4	551,881	1089	420
5	14,369,347	3276	1300
6		3437	1350

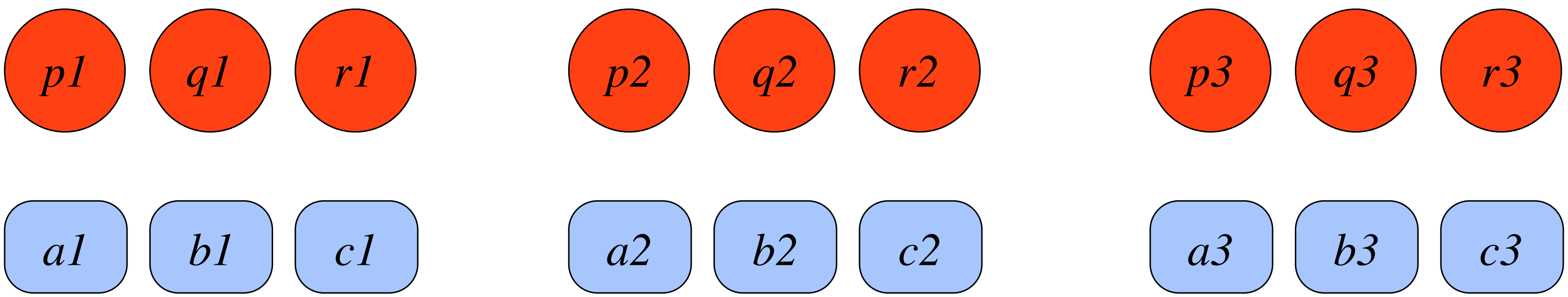
Unfactored: `1 + 27 + 27^2 + 27^3 + 27^4 + 27^5`

Factored: (1 + 3 + 3^2 + 3^3 + 3^4 + 3^5) x 9

Conditions

Inertiality

Overall terminal and goals defined as
disjunctions of individual terminals and goals.



Suppose one group changes on action in another group.

Inertiality

A subgame is *inertial* iff the state of the subgame does not change when no action in the subgame is performed.

Technique

Inertiality is computed by checking `next(p)` for every base proposition *p*.

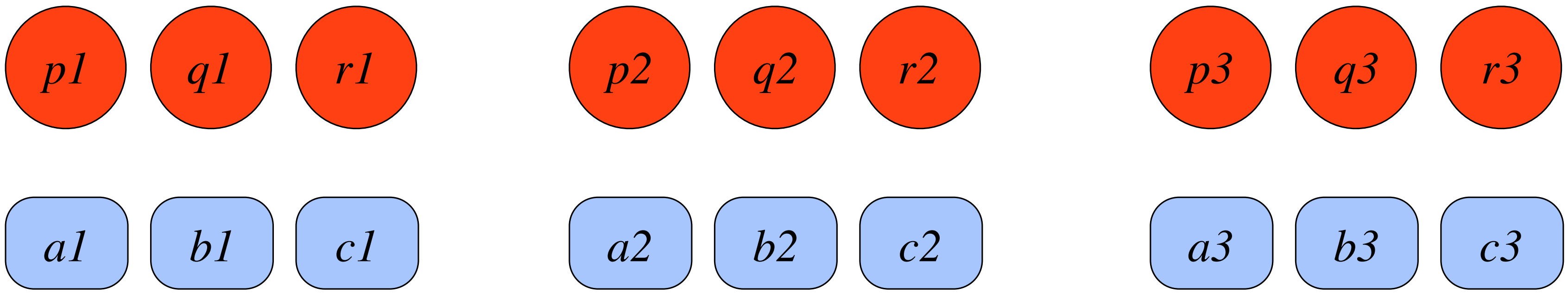
Reduce all rules for `next(p)` to rules defining `next(p)` in terms of `true(p)` and `does(r,p)`, resolve with each other, and filter out subsumed rules.

There must be a rule `next(p) :- true(p) & junk` where junk does not depend on base propositions.

There must not be any rule of the form `next(p):-junk` where *junk* does not include `true(p)`.

No Termination

Overall terminal and goals defined as
disjunctions of individual terminals and goals.



Is it possible for a group to have no terminal state?

Termination

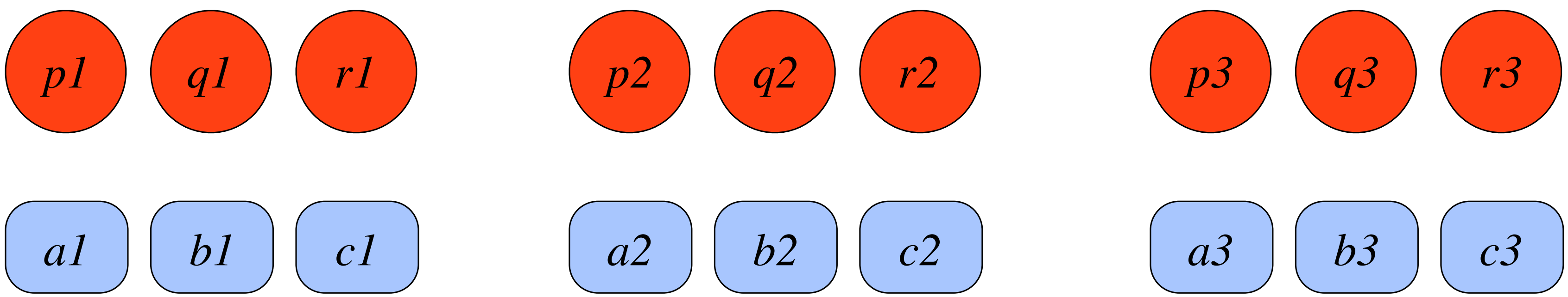
Possible for a subgame not to have a termination condition at all?

Not possible. All games must terminate. If game is inertial and some subgame does not have a termination condition, then it would be possible to play repeatedly in the game forever, contradicting the requirement for termination of all games.

Upshot: No check necessary.

Goal Coverage

Overall terminal and goals defined as
disjunctions of individual terminals and goals.



Is it possible for one group to terminate with low goal
while another is non-terminal with higher goal?

Goal Coverage

In general, it is possible for a subgame to have a high goal and no termination while another has termination and low goal.

Problem resolved if all non-zero goal states are also terminal states.

Can be checked in a manner similar to checking for inertiality.

Validation Problem

Check that the game satisfies all of these conditions.

Almost as difficult or even more difficult than factoring.

Brian’s Method

(1) Factor

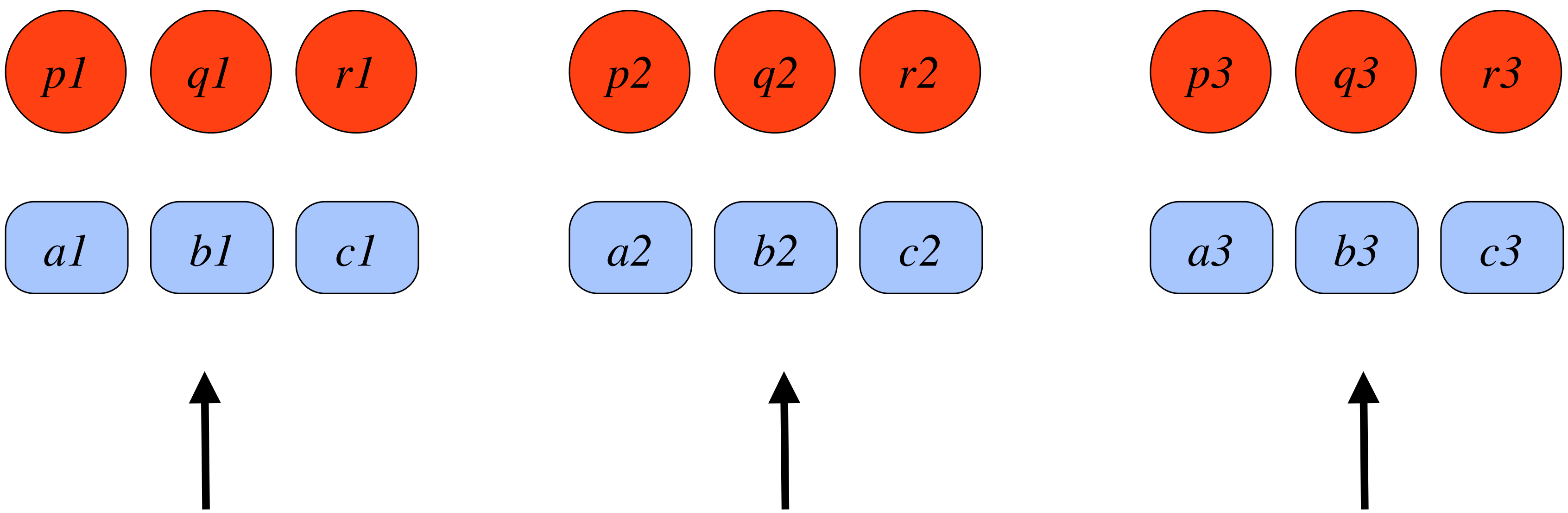
(2) Get a solution

(3) See if it works in the unfactored game.

Other Types of Game Analysis

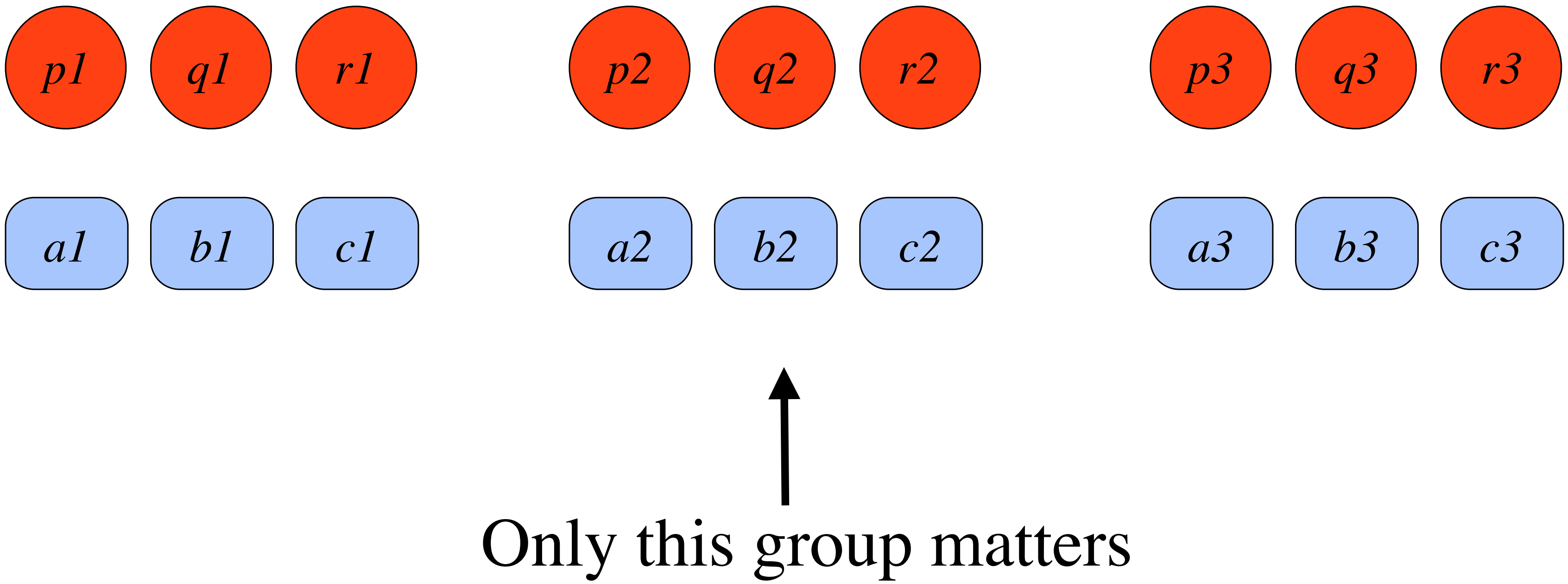
Best Buttons and Lights

Overall terminal and goals defined as disjunctions of individual terminals and goals.

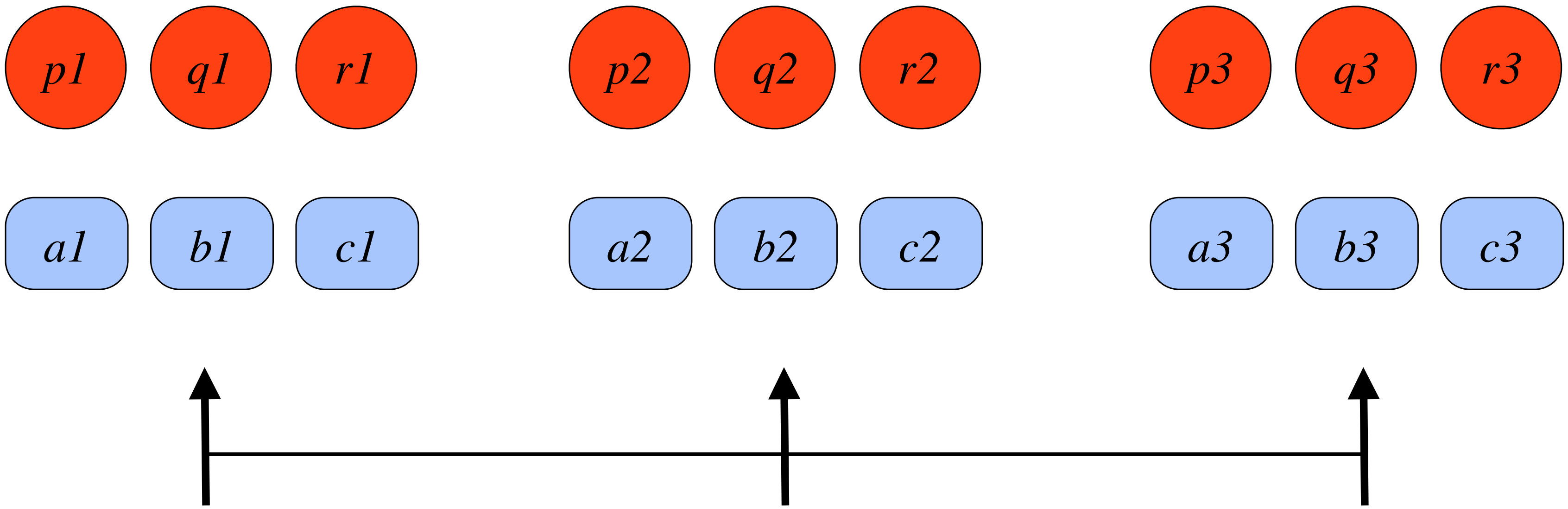


Terminating one group terminates entire game.

Multiple Buttons and Lights

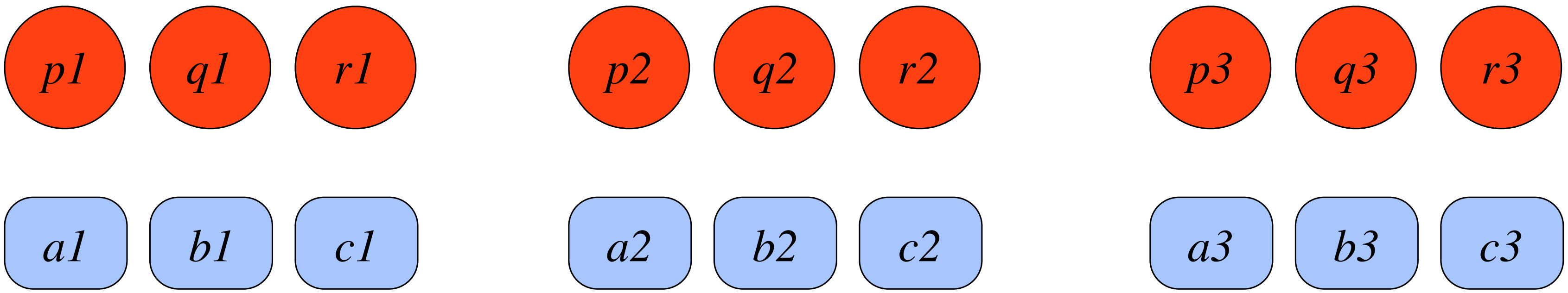


Joint Buttons and Lights



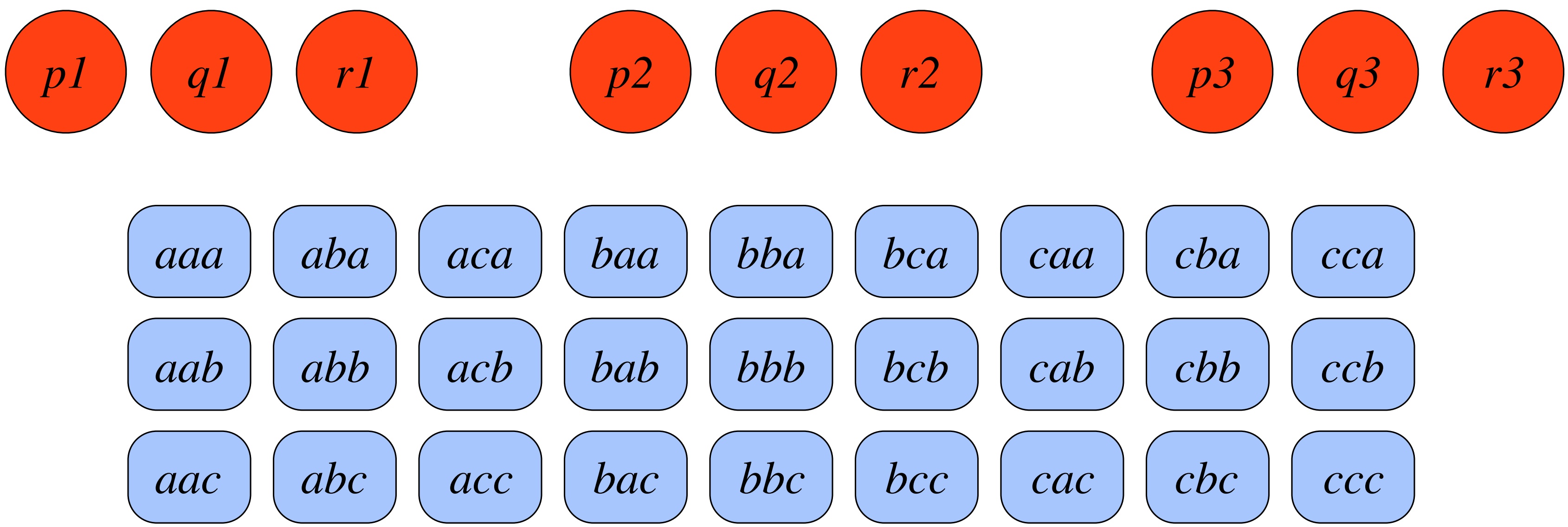
Terminates if and only if *all* groups terminate.

Parallel Buttons and Lights

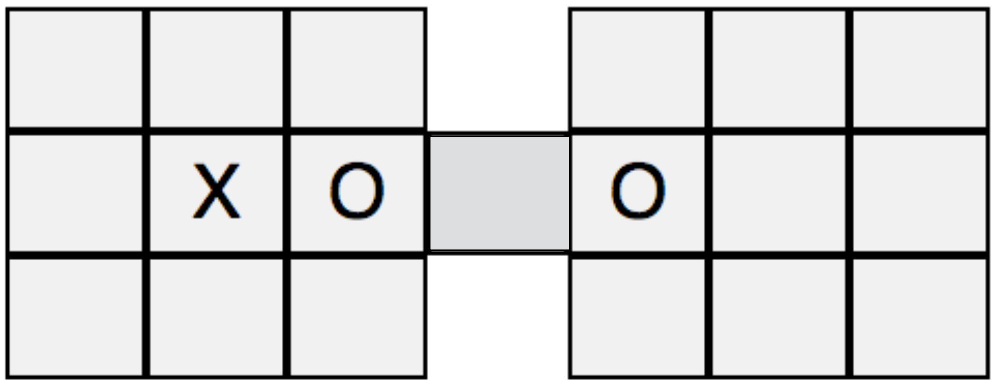


Terminal if and only if *all* groups terminal.
Each group acted upon by different player.

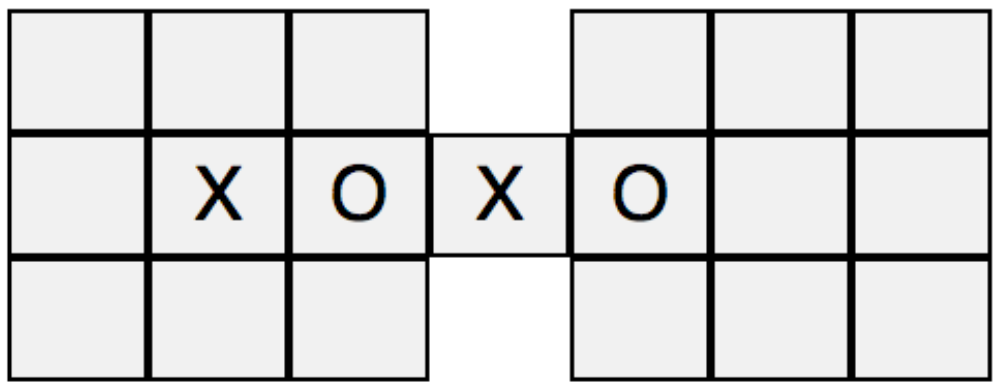
Simultaneous Buttons and Lights



Conditional Factoring



Conditional Factoring



Related Types of Game Analysis

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


GENERAL
GAME
PLAYING