

Lecture 2:

Game Description Language

- Games as State Machines
- A Logical Game Description Language (GDL)

Finite Games

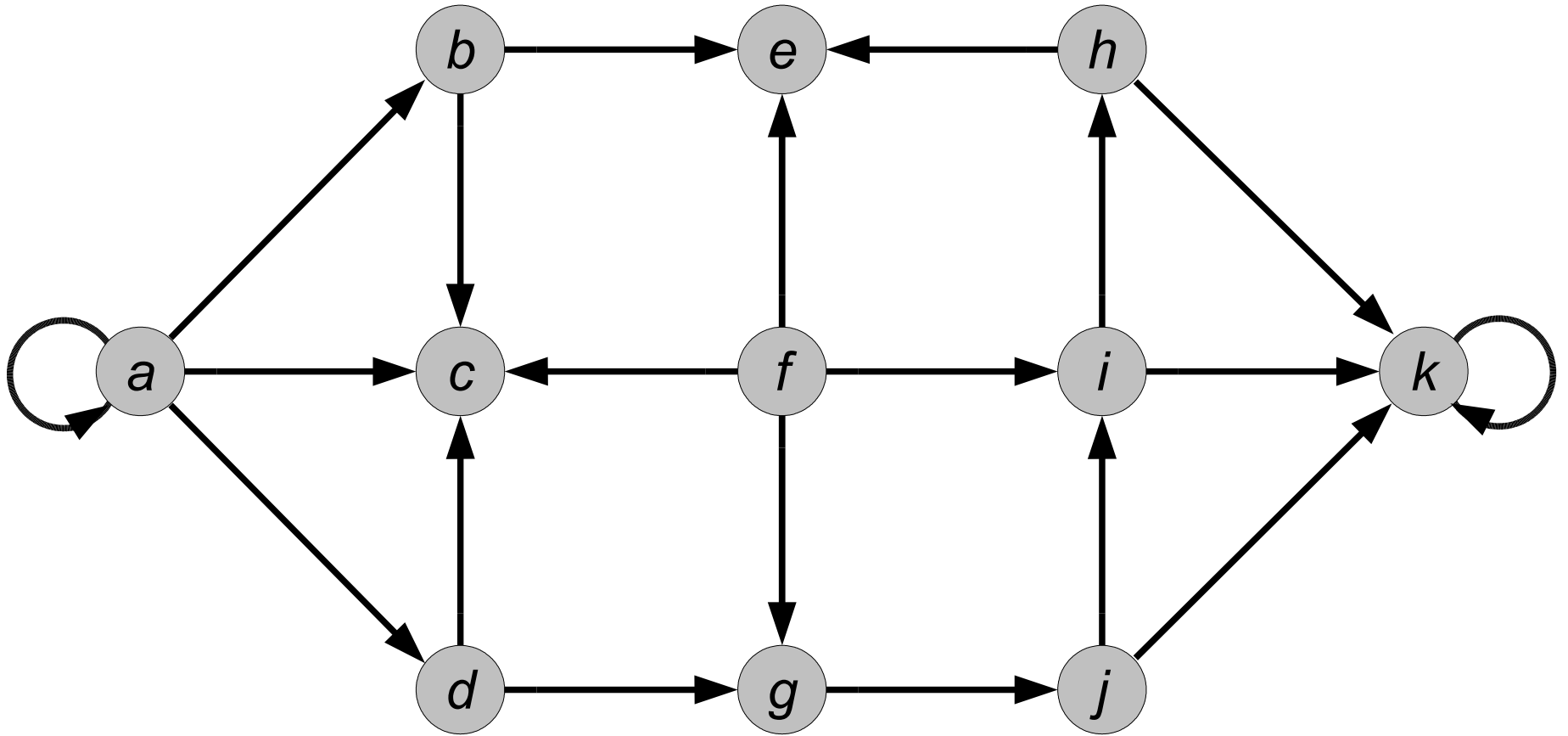
Finite Environment

- Game “world” with finitely many states
- One initial state and one or more terminal states
- Fixed finite number of players
- Each with finitely many “percepts” and “actions”
- Each with one or more goal states

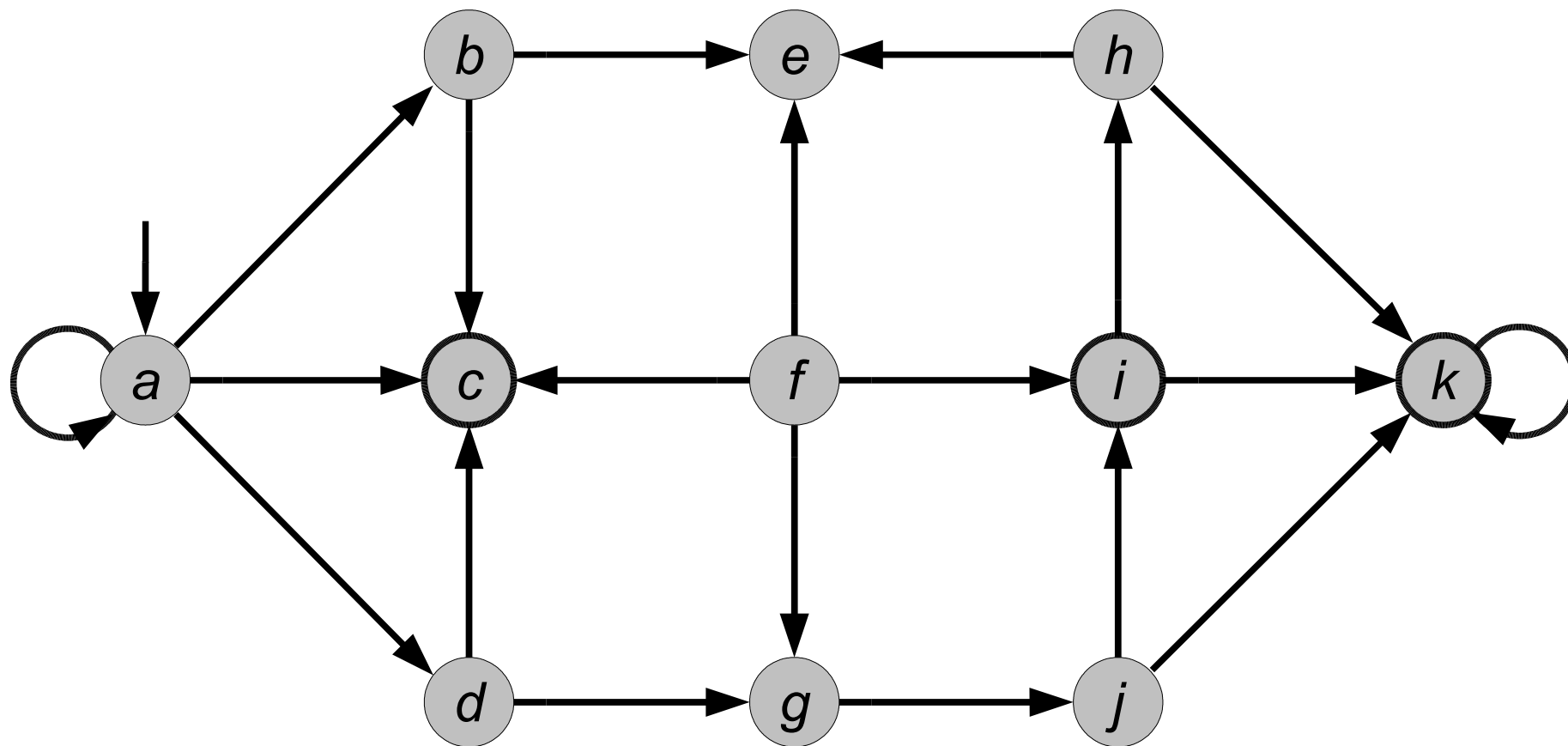
Causal Model

- Environment changes only in response to moves
- Synchronous actions

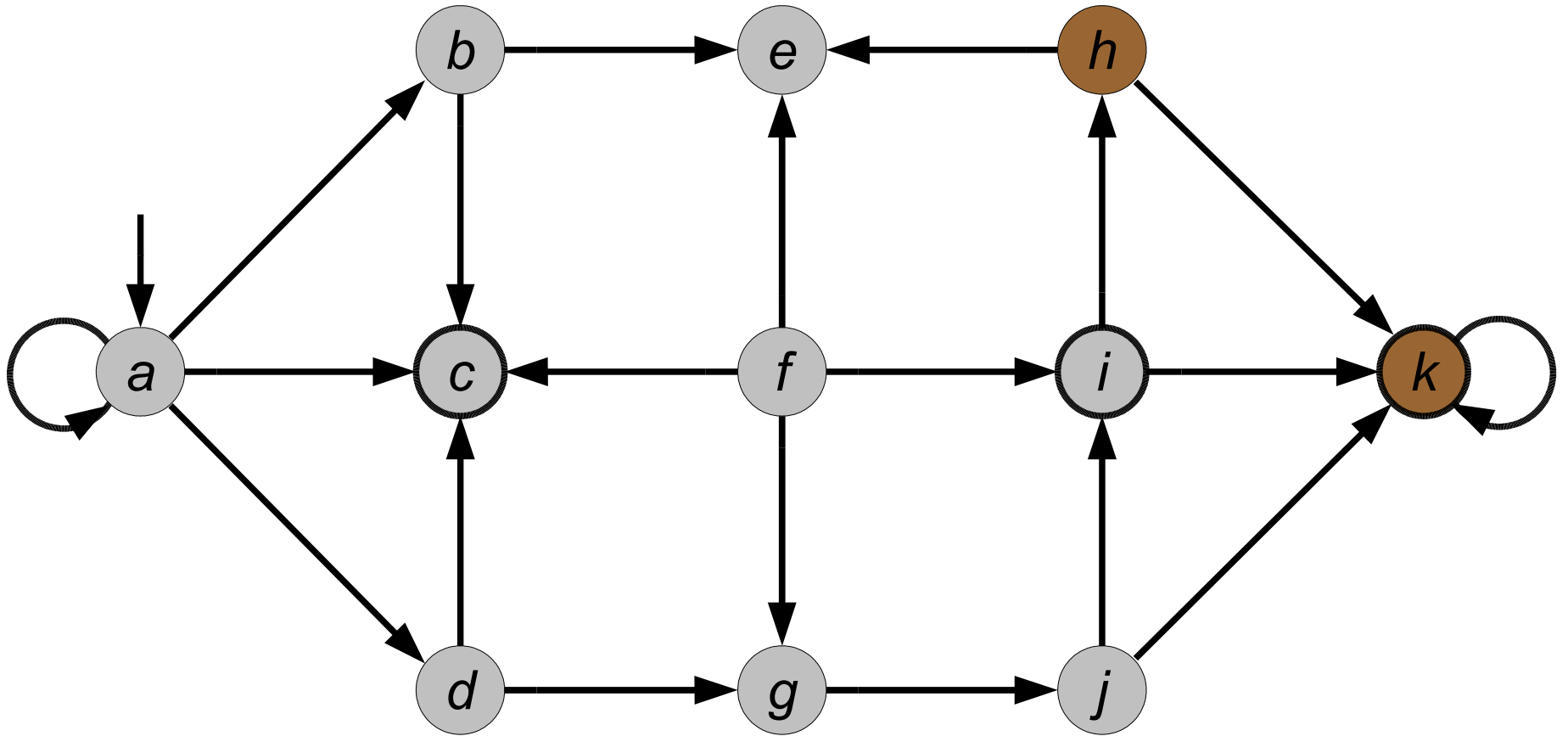
Games as State Machines



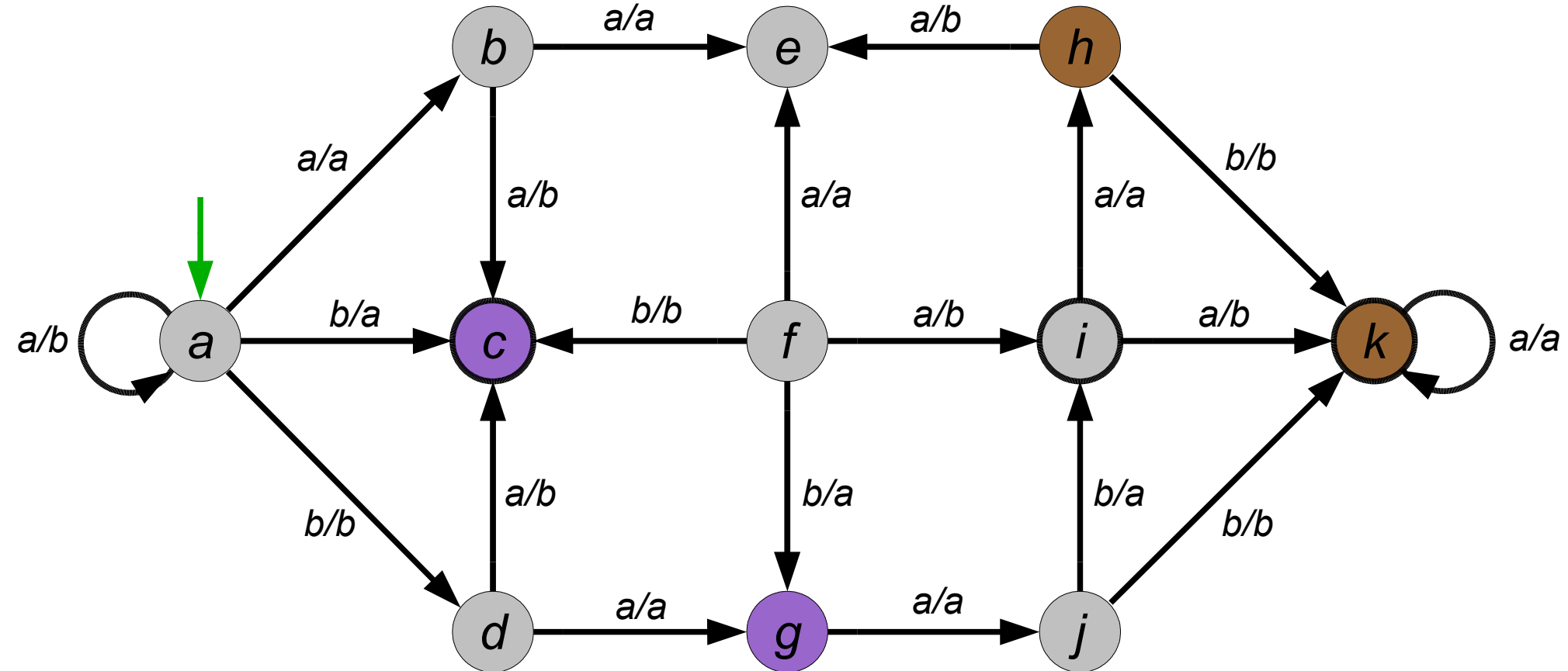
Initial State and Terminal States



Goal States



Simultaneous Actions



Game Model

An n -player game is a structure with components:

S – set of states

A_1, \dots, A_n – n sets of actions, one for each player

I_1, \dots, I_n – where $I_i \subseteq A_i \times S$, the legality relations

$u: S \times A_1 \times \dots \times A_n \rightarrow S$ – update function

$s_1 \in S$ – initial game state

$t \subseteq S$ – the terminal states

g_1, \dots, g_n – where $g_i \subseteq S \times \mathbb{N}$, the goal relations

Direct Description

Since all of the games that we are considering are finite, it is possible in principle to communicate game information in the form of lists (of states and actions), and tables (for legality, update, etc.)

Problem: Size of description. Even though everything is finite, the necessary tables can be large

Solutions:

- Reformulate in modular fashion
- Use compact encoding

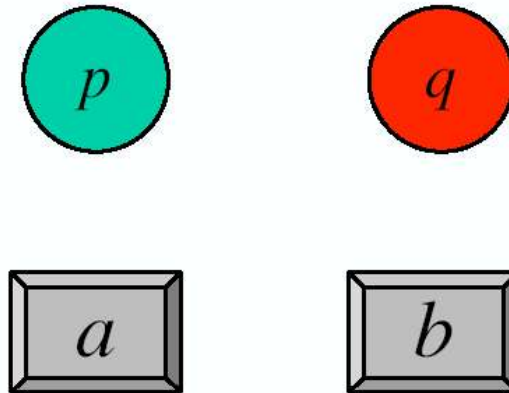
States versus Fluents

In many cases, worlds are best thought of in terms of **fluents**, i.e. features that may change; e.g. “position-of-white-queen”, “black-can-castle”. Actions affect subsets of fluents.

States represent all possible ways the world can be. As such, the number of states is exponential in the number of fluents of the world, and the action tables are correspondingly large.

Solution: Represent fluents directly and describe how actions change individual fluents rather than entire states

Buttons and Lights



Pressing button a toggles p .

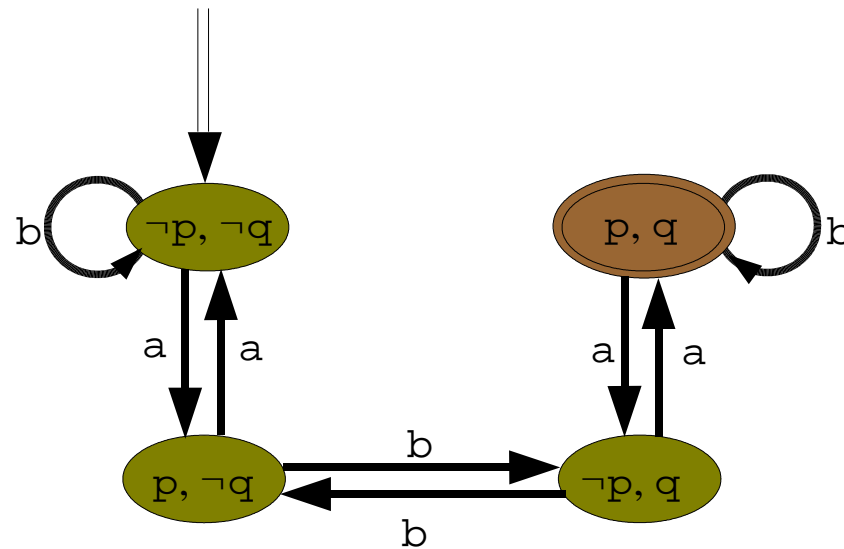
Pressing button b interchanges p and q .

Initially, p and q are off. Goal: p and q are on.

State Machine for Buttons and Light

Fluents: p , q

Actions: a , b



Game Description for Buttons and Light

Legality

```
legal(robot, a)
```

```
legal(robot,b)
```

Update

$$\begin{aligned} \text{next}(p) \leq & \text{does}(\text{robot}, a) \wedge \neg \text{true}(p) \\ & \vee \text{does}(\text{robot}, b) \wedge \text{true}(q) \end{aligned}$$
$$\text{next}(q) \leq \text{does}(\text{robot}, a) \wedge \text{true}(q) \vee \text{does}(\text{robot}, b) \wedge \text{true}(p)$$

Termination and Goal

```
terminal <= true(p)  ^  true(q)
```

```
goal(robot,100) <= true(p) ^ true(q)
```

Relational Logic

Object Variables: x, y, z

Object Constants: a, b, c

Function Constants: f, g, h

Relation Constants: $p, q, r, \text{distinct}$

Logical Operators: $\neg, \wedge, \vee, \leq, \leq\Rightarrow$

Terms: $x, y, z, a, b, c, f(a), g(a,x), h(a,b,f(y))$

Relational Sentences: $p(a,b)$

Logical Sentences: $r(x,y) \leq p(x,y) \wedge \neg q(y)$

Standard first-order semantics

Clausal Logic

Literals: relational sentences, negated relational sentences

Clauses

- Facts: relational sentences
- Rules: $\text{Head} \leq \text{Body}$

Head: relational sentence

Body: logical sentence built from \wedge , \vee , literal

Standard completion semantics (“negation-as-failure”)

Completion

Let P be a finite set of clauses.

For a relation $p(x_1, \dots, x_n)$ let

$$\begin{aligned} p(t_{11}, \dots, t_{1n}) &\leq B_1 \\ &\vdots \\ p(t_{m1}, \dots, t_{mn}) &\leq B_m \end{aligned}$$

be all clauses in P with head p .

The **completion of p in P** is the formula

$$\begin{aligned} p(x_1, \dots, x_n) \Leftrightarrow & x_1 = t_{11} \wedge \dots \wedge x_n = t_{1n} \wedge B_1 \\ & \vee \dots \vee \\ & x_1 = t_{m1} \wedge \dots \wedge x_n = t_{mn} \wedge B_m \end{aligned}$$

(If $m=0$ then the completion is $\neg p(x_1, \dots, x_n)$.)

Completion for Buttons and Light

$\neg \text{init}(F)$

$\text{legal}(P,A) \iff P=\text{robot} \wedge A=a$

$\vee P=\text{robot} \wedge A=b$

$\text{next}(F) \iff F=p \wedge (\text{does}(\text{robot},a) \wedge \neg \text{true}(p)$

$\vee \text{does}(\text{robot},b) \wedge \text{true}(q))$

\vee

$F=q \wedge (\text{does}(\text{robot},a) \wedge \text{true}(q)$

$\vee \text{does}(\text{robot},b) \wedge \text{true}(p))$

$\text{terminal} \iff \text{true}(p) \wedge \text{true}(q)$

$\text{goal}(P,V) \iff P=\text{robot} \wedge V=100 \wedge \text{true}(p) \wedge \text{true}(q)$

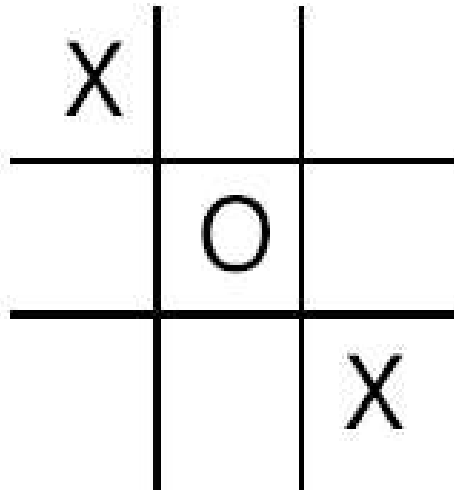
Game-Independent Vocabulary

- Object constants
0, 1, 2, 3, ...

- Relations

`init(fluent)`
`true(fluent)`
`does(player, action)`
`next(fluent)`
`legal(player, action)`
`goal(player, value)`
`terminal`

Tic-Tac-Toe




Tic-Tac-Toe Vocabulary


- **Object constants**
`xplayer, oplayer` Players
`x, o, b` Marks
- **Functions**
`cell(number,number,mark)` Fluent
`control(player)` Fluent
`mark(number,number)` Move
- **Relations**
`row(number,mark)`
`column(number,mark)`
`diagonal(mark)`
`line(mark)`
`open`

Initial State

```
init(cell(1,1,b))  
init(cell(1,2,b))  
init(cell(1,3,b))  
init(cell(2,1,b))  
init(cell(2,2,b))  
init(cell(2,3,b))  
init(cell(3,1,b))  
init(cell(3,2,b))  
init(cell(3,3,b))  
init(control(xplayer))
```

Legality

 $\text{legal}(P, \text{mark}(X, Y)) \leq$
 $\text{true}(\text{cell}(X, Y, b)) \wedge$
 $\text{true}(\text{control}(P))$

$\text{legal}(\text{xplayer}, \text{noop}) \leq$
 $\text{true}(\text{cell}(X, Y, b)) \wedge$
 $\text{true}(\text{control}(\text{oplayer}))$

$\text{legal}(\text{oplayer}, \text{noop}) \leq$
 $\text{true}(\text{cell}(X, Y, b)) \wedge$
 $\text{true}(\text{control}(\text{xplayer}))$

Update

```
next(cell(M,N,x)) <=
    does(xplayer,mark(M,N))
```

```
next(cell(M,N,o)) <=
    does(oplayer,mark(M,N))
```

```
next(cell(M,N,W)) <=
    true(cell(M,N,W)) ∧
    distinct(W,b)
```

```
next(cell(M,N,b)) <=
    true(cell(M,N,b)) ∧
    does(P,mark(J,K)) ∧
    (distinct(M,J) ∨ distinct(N,K))
```

Update (continued)

```
next(control(xplayer)) <=  
    true(control(oplayer))
```

```
next(control(oplayer)) <=  
    true(control(xplayer))
```

Termination

`terminal <= line(x) ∨ line(o)`

`terminal <= ¬open`

`line(W) <= row(M,W)`

`line(W) <= column(N,W)`

`line(W) <= diagonal(W)`

`open <= true(cell(M,N,b))`

Supporting Concepts

```
row(M,W) <=
    true(cell(M,1,W)) ^
    true(cell(M,2,W)) ^
    true(cell(M,3,W))
```

```
column(N,W) <=
    true(cell(1,N,W)) ^
    true(cell(2,N,W)) ^
    true(cell(3,N,W))
```

```
diagonal(W) <=
    true(cell(1,1,W)) ^
    true(cell(2,2,W)) ^
    true(cell(3,3,W))
```

```
diagonal(W) <=
    true(cell(1,3,W)) ^
    true(cell(2,2,W)) ^
    true(cell(3,1,W))
```

Goals

```
goal(xplayer,100) <= line(x)
goal(xplayer,50)  <= ¬line(x) ∧ ¬line(o) ∧ ¬open
goal(xplayer,0)   <= line(o)

goal(oplayer,100) <= line(o)
goal(oplayer,50)  <= ¬line(x) ∧ ¬line(o) ∧ ¬open
goal(oplayer,0)   <= line(x)
```

Guaranteeing Finite Derivability (I): Safety

A clause is *safe* if and only if every variable in the clause appears in some positive subgoal in the body.

Safe Rule:

$$r(X, Y) \leq p(X, Y) \wedge \neg q(X, Y)$$

Unsafe Rule:

$$r(X, Z) \leq p(X, Y) \wedge \neg q(Y, Z)$$

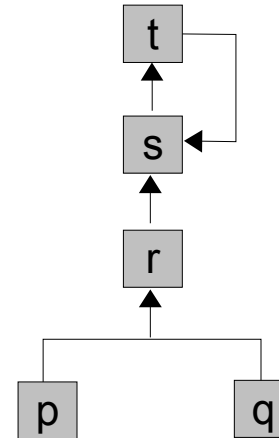
In GDL, we require all rules to be safe.

(Note that this implies all facts to be variable-free.)

Dependency Graph

The *dependency graph* for a set of clauses is a directed graph in which

- the nodes are the relations mentioned in the head and bodies of the clauses
- there is an arc from a node p to a node q whenever p occurs in the body of a clause in which q is in the head.

$$r(X, Y) \leq p(X, Y) \wedge q(X, Y)$$
$$s(X, Y) \leq r(X, Y)$$
$$s(X, Z) \leq r(X, Y) \wedge t(Y, Z)$$
$$t(X, Z) \leq s(X, Y) \wedge s(Y, X)$$


A set of clauses is *recursive* if its dependency graph contains a cycle. Otherwise, it is *non-recursive*.

Guaranteeing Finite Derivability (II): Stratified Negation

A set of rules is said to be *stratified* if there is no recursive cycle in the dependency graph involving a negation.

- Stratified Negation:

$$t(X, Y) \leq q(X, Y) \wedge \neg r(X, Y)$$

$$r(X, Z) \leq p(X, Y)$$

$$r(X, Z) \leq r(X, Y) \wedge r(Y, Z)$$

- Negation that is not stratified:

$$r(X, Z) \leq p(X, Y)$$

$$r(X, Z) \leq p(X, Y) \wedge \neg r(Y, Z)$$

In GDL, we require all sets of rules to be stratified.

Guaranteeing Finite Derivability (III)

If a set of rules contains a clause

$$p(s_1, \dots, s_m) \leq b_1(\vec{t}_1) \wedge \dots \wedge q(v_1, \dots, v_k) \wedge \dots \wedge b_n(\vec{t}_n)$$

such that p and q occur in a cycle in the dependency graph, then for every $i \in \{1, \dots, k\}$

- v_i is variable-free, or
- v_i is one of s_1, \dots, s_m , or
- v_i occurs in some \vec{t}_j such that b_j does not occur in a cycle with p in the dependency graph ($1 \leq j \leq n$).

Completeness

Of necessity, game descriptions are logically incomplete in that they do not uniquely specify the moves of the players.

Every game description contains *complete definitions* for *legality*, *termination*, *goalhood*, and *update* in terms of the relations *true* and *does*.

The upshot is that in every state every player can determine legality, termination, goalhood, and, given a joint move, can update the state.

Playability

A game is *playable* if and only if every player has at least one legal move in every non-terminal state.

(Note that in chess, if a player cannot move, it is a stalemate. Fortunately, this is a terminal state.)

In GGP, we guarantee that every game is playable.

Winnability

A game is *strongly winnable* if and only if, for some player, there is a sequence of individual moves of that player that leads to a terminating goal state for that player.

A game is *weakly winnable* if and only if, for every player, there is a sequence of joint moves of the players that leads to a terminating goal state for that player.

In GGP, every game should be weakly winnable, and all single player games should be strongly winnable.

Knowledge Interchange Format

Knowledge Interchange Format is a standard for programmatic exchange of knowledge represented in relational logic.

Syntax is prefix version of standard syntax.

Some operators are renamed: `not`, `and`, `or`.

Case-independent. Variables are prefixed with `?`.

$r(X,Y) \leq p(X,Y) \wedge \neg q(Y)$

`(<= (r ?x ?y) (and (p ?x ?y) (not (q ?y))))`

or, equivalently,

`(<= (r ?x ?y) (p ?x ?y) (not (q ?y)))`

Semantics is the same.