Finite Deterministic Games

Environment
Game “world” with finitely many states
One initial state and one or more terminal states

Players
Finite number of players
Each with finitely many “percepts” and “actions”
Each with one or more goal “states”

Deterministic Operation
Environment changes only in response to moves
Synchronous and asynchronous actions
Single Player Game as a State Machine

Composite Actions for Multiple Players
Problem

Since all of the games that we are considering are finite, it is possible in principle to communicate game information in the form of transition graphs.

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

Solution

Different Conceptualizations:
- State Machines
- Propositional Nets
- Object Nets

Different Linearizations:
- Direct Encoding
- Logical Encoding
Game Description Language
Vocabulary

Components:
Object Variables: $X, Y, Z$
Object Constants: $a, b, c$
Function Constants: $f, g, h$
Relation Constants: $p, q, r$
Logical Operators: $\neg, \land, :-$
Punctuation: $(, , )$

The *arity* of a function or relation is the number of arguments that can be supplied to the function or relation. NB: Variadic functions and relations are possible, but in our work here we fix the arity for each function and relation in advance.

Syntax

Terms:
Variables: $x, y, z$
Object Constants: $a, b, c$
Functional Terms: $f(a), g(x,b), h(x,f(y),c)$

Sentences:
Atoms: $p(a,b), p(a,f(a)), p(g(f(a),b),c)$
Negations: $\neg p(a,b)$
Rules: $r(x,y) :- p(x,y,c) \land \neg q(y)$
Logic Programs

A logic program is a finite set of atoms and rules.

Example:
\[ p(X,Y) :- f(X,Y) \]
\[ p(X,Y) :- m(X,Y) \]

Example:
\[ g(X,Z) :- p(X,Y) \& p(Y,Z) \]

Example:
\[ a(X,Z) :- p(X,Z) \]
\[ a(X,Z) :- p(X,Y) \& a(Y,Z) \]

Example:
\[ ra(X,Y) :- a(X,Y) \& \neg p(X,Y) \]

Terminology

A literal is either an atom or a negation of an atom.
\[ p(a,b), \neg p(a,b) \]

Rules:
\[ p(x_1,...,x_n) :- p_1(x_{i1},...,x_{in}) \& ... \& p_k(x_{k1},...,x_{kn}) \]

An expression is ground if and only if it contains no variables.
Safety

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

Safe Rule:

\[ r(X,Z) \iff p(X,Y) \land q(Y,Z) \land \neg r(X,Y) \]

Unsafe Rule:

\[ r(X,Z) \iff p(X,Y) \land q(Y,X) \]

Unsafe Rule:

\[ r(X,Z) \iff p(X,Y) \land \neg q(Y,Z) \]

In GDL, we require all rules to be safe.

Dependency Graph

The *dependency graph* for a set of rules is a directed graph in which (1) the nodes are the relations mentioned in the head and bodies of the rules and (2) there is an arc from a node \( p \) to a node \( q \) whenever \( p \) occurs with the body of a rule in which \( q \) is in the head.

\[ r(X,Y) \iff p(X,Y) \land q(X,Y) \]
\[ S(X,Y) \iff r(X,Y) \]
\[ S(X,Z) \iff r(X,Y) \land t(Y,Z) \]
\[ t(X,Z) \iff s(X,Y) \land s(Y,X) \]

A set of rules is *recursive* if it contains a cycle. Otherwise, it is *non-recursive*. 
Stratified Negation

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

Negation that is not stratified:
\[ r(X, Z) :- p(X, Y, Z) \]
\[ r(X, Z) :- p(X, Y, Z) \land \neg r(Y, Z) \]

Stratified Negation:
\[ t(X, Y) :- q(X, Y) \land \neg r(X, Y) \]
\[ r(X, Z) :- p(X, Y) \]
\[ r(X, Z) :- r(X, Y) \land r(Y, Z) \]

In GDL, we require that all negations be stratified.

Stratified Recursion

The recursion in a set of rules is said to be *stratified* if and only if every variable in a subgoal relation occurs in a subgoal with a relation at a lower stratum.

Stratified Recursion:
\[ r(X, Z) :- p(X, Y) \land q(Z) \land r(Y, Z) \]

Recursion that is not stratified:
\[ r(X, Z) :- r(X, Y) \land r(Y, Z) \]

In GDL, we require that all recursions be stratified.
**Herbrand Universe**

The *Herbrand universe* for a logic program is the set of all ground terms in the language.

Example 1:
- Object Constants: \( a, b \)
- Herbrand Universe: \( a, b \)

Example 2:
- Object Constants: \( a \)
- Unary Function Constants: \( f \)
- Herbrand Universe: \( a, f(a), f(f(a)), \ldots \)

**Herbrand Base**

The *Herbrand base* for a logic program is the set of all ground atoms in the language.

Object Constants: \( a, b \)
- Unary Relation Constant: \( p \)
- Binary Relation Constant: \( q \)
- Herbrand Universe: \( a, b \)

Herbrand Base:
- \( \{ p(a), p(b), q(a,a), q(a,b), q(b,a), q(b,b) \} \)
Herbrand Interpretations

An interpretation for a logic program is an arbitrary subset of the Herbrand base for the program.

Herbrand Base:
\{\text{p}(a), \text{p}(b), \text{q}(a,a), \text{q}(a,b), \text{q}(b,a), \text{q}(b,b)\}

Interpretation 1:
\{\text{p}(a), \text{q}(a,b), \text{q}(b,a)\}

Interpretation 2:
\{\text{p}(a), \text{p}(b), \text{q}(a,a), \text{q}(a,b), \text{q}(b,a), \text{q}(b,b)\}

Interpretation 3:
{}  

Herbrand Models

An interpretation \(\Delta\) satisfies a ground sentence (i.e. it is a model) under the following conditions:

\(\Delta\) satisfies an atom \(\varphi\) iff \(\varphi \in \Delta\).

\(\Delta\) satisfies \(\neg \varphi\) iff \(\Delta\) does not satisfy \(\varphi\).

\(\Delta\) satisfies \(\varphi_1 \land \ldots \land \varphi_n\) iff \(\Delta\) satisfies every \(\varphi_i\).

\(\Delta\) satisfies \(\varphi_2 :- \varphi_1\) iff
\(\Delta\) satisfies \(\varphi_2\) whenever it satisfies \(\varphi_1\).

An interpretation satisfies a non-ground sentence if and only if it satisfies every ground instance of that sentence.
Multiple Models

In general, a logic program can have more than one model.

Logic Program:
\{p(a, b), q(X, Y) :- p(X, Y)\}

Model 1: \{p(a,b), q(a,b)\}

Model 2: \{p(a,b), q(a,b), q(b,a)\}

Minimality

To eliminate ambiguity, we adopt a minimal model semantics. A model D is a \textit{minimal model} of a program O iff no proper subset of D is a model of O.

Logic Program:
\{p(a,b), q(X, Y) :- p(X, Y)\}

\textbf{Good} Model 1: \{p(a,b), q(a,b)\}

\textbf{Bad} Model 2: \{p(a,b), q(a,b), q(b,a)\}

Theorem: Given our various restrictions, every logic program has a unique minimal model. Therefore, unambiguous answers. Also, all models are finite!
Open versus Closed Logic Programs

Logic programs as just defined are closed in that they fix the meaning of all relations.

In open logic programs, some of the relations are given as inputs (rather than being defined) and other relations are produced as outputs (defined in terms of the inputs). In other words, open logic programs can be used with different input datasets to produce different output datasets.

Open Logic Programs

An open logic program is a logic program together with a partition of the relations into two types - base relations (the inputs) and view relations (the defined relations).

View relations can appear anywhere in the program, while base relations can appear only in the bodies of rules (never in the heads of rules).
Inputs and Outputs

An overall model of an open logic program $O$ with an input model $D$ is the minimal model of $O \cup D$.

The output of an open logic program for input $D$ is the subset of the overall model of $O$ and $D$ that mentions only output relations.

Example

Base Relations: $p$, $q$
View Relations: $r$
Rules: \{ $r(X,Y) :\neg p(X,Y) \& \neg q(Y)$ \}

Input 1: \{ $p(a,b), p(b,b), q(b)$ \}
Output 1: \{ $r(a,b)$ \}

Input 2: \{ $p(a,b), p(b,b)$ \}
Output 2: \{ $r(a,b), r(b,b)$ \}
A GDL description is an open logic program in which we treat does and true as inputs. For each choice of inputs, the description tells us init, legal, next, goal, and terminal.

Upshot is that each GDL program corresponds to one and only one state machine.

Initial state is the state in which all init facts are true. Initial legal actions are defined in terms of this state.

Given initial facts and choice of legal actions, next tells us what is true in the next state.

And so forth.
Game-Independent Vocabulary

Object Constants:
$$0, 1, 2, 3, \ldots, 100$$

Relation Constants:
- role(player)
- proposition(proposition)
- action(action)
- init(proposition)
- true(proposition)
- next(proposition)
- legal(player, action)
- does(player, action)
- goal(proposition)
- terminal

Game-Specific Vocabulary

Object constants:
- white, black - players
- x, o, b - marks

Function Constants:
- mark(number, number) --> action
- cell(number, number, mark) --> proposition
- control(player) --> proposition

Relation Constants:
- row(number, player)
- column(number, player)
- diagonal(player)
- line(player)
- open
No Built-in Assumptions

What we see:

\[
\text{next(cell(M,N,x)) :-}
\text{does(white,mark(M,N)) &}
\text{true(cell(M,N,b))}
\]

What the player sees:

\[
\text{next(welcoul(M,N,himenoing)) :-}
\text{does(himenoing,dukepse(M,N)) &}
\text{true(welcoul(M,N,lorenchise))}
\]

Tic-Tac-Toe

```
X

O

X
```
State Representation

X

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

cell(1,1,x)  
cell(1,2,b)  
cell(1,3,b)  
control(black)

Initial State

\[
\begin{align*}
\text{init}(\text{cell}(1,1,b)) \\
\text{init}(\text{cell}(1,2,b)) \\
\text{init}(\text{cell}(1,3,b)) \\
\text{init}(\text{cell}(2,1,b)) \\
\text{init}(\text{cell}(2,2,o)) \\
\text{init}(\text{cell}(2,3,b)) \\
\text{init}(\text{cell}(3,1,b)) \\
\text{init}(\text{cell}(3,2,b)) \\
\text{init}(\text{cell}(3,3,x)) \\
\text{init}(\text{control}(x))
\end{align*}
\]
Legality

\[
\text{legal}(W, \text{mark}(X,Y)) :\neg \\
\text{true}(\text{cell}(X,Y,b)) \land \\
\text{true}(\text{control}(W))
\]

\[
\text{legal}(\text{white}, \text{noop}) :\neg \\
\text{true}(\text{cell}(X,Y,b)) \land \\
\text{true}(\text{control}(\text{black}))
\]

\[
\text{legal}(\text{black}, \text{noop}) :\neg \\
\text{true}(\text{cell}(X,Y,b)) \land \\
\text{true}(\text{control}(\text{white}))
\]

Conclusions:

\[
\text{legal}(\text{white}, \text{noop})
\]

\[
\text{legal}(\text{black}, \text{mark}(1,2))
\]

\[
\text{legal}(\text{black}, \text{mark}(3,2))
\]

Physics

\[
\text{cell}(1,1,x) \\
\text{cell}(1,2,b) \\
\text{cell}(1,3,b) \\
\text{cell}(2,1,b) \\
\text{cell}(2,2,o) \\
\text{cell}(2,3,b) \\
\text{cell}(3,1,b) \\
\text{cell}(3,2,b) \\
\text{cell}(3,3,x) \\
\text{control}(\text{black})
\]

\[
\text{cell}(1,1,x) \\
\text{cell}(1,2,b) \\
\text{cell}(1,3,o) \\
\text{cell}(2,1,b) \\
\text{cell}(2,2,o) \\
\text{cell}(2,3,b) \\
\text{cell}(3,1,b) \\
\text{cell}(3,2,b) \\
\text{cell}(3,3,x) \\
\text{control}(\text{white})
\]

\[
\text{next}(\text{cell}(M,N,o)) :\neg \\
\text{does}(\text{black}, \text{mark}(M,N)) \land \\
\text{true}(\text{cell}(M,N,b))
\]
Physics

next(cell(M,N,x)) :-
does(white,mark(M,N))

next(cell(M,N,o)) :-
does(black,mark(M,N))

next(cell(M,N,Z)) :-
does(W,mark(M,N)) &
true(cell(M,N,Z)) & Z#b

next(cell(M,N,b)) :-
does(W,mark(J,K)) &
true(cell(M,N,b)) & (M#J | N#K)

next(control(white)) :-
true(control(black))

next(control(black)) :-
true(control(white))

Supporting Concepts

row(M,W) :-
true(cell(M,1,W)) &
true(cell(M,2,W)) &
true(cell(M,3,W))

diagonal(W) :-
true(cell(1,1,W)) &
true(cell(2,2,W)) &
true(cell(3,3,W))

column(N,W) :-
true(cell(1,N,W)) &
true(cell(2,N,W)) &
true(cell(3,N,W))

diagonal(W) :-
true(cell(1,3,W)) &
true(cell(2,2,W)) &
true(cell(3,1,W))

line(W) :- row(M,W)
line(W) :- column(N,W)
line(W) :- diagonal(W)

open :- true(cell(M,N,b))
Goals and Termination

\[\text{goal(white,100)} :- \text{line(x)}\]
\[\text{goal(white,50)} :- \neg\text{line(x)} \& \neg\text{line(o)}\]
\[\text{goal(white,0)} :- \text{line(o)}\]

\[\text{goal(black,100)} :- \text{line(o)}\]
\[\text{goal(black,50)} :- \neg\text{line(x)} \& \neg\text{line(o)}\]
\[\text{goal(black,0)} :- \text{line(x)}\]

\[\text{terminal} :- \text{line(W)}\]
\[\text{terminal} :- \neg\text{open}\]

Roles

\[\text{role(white)}\]
\[\text{role(black)}\]
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. Operators are renamed. Case-independent. Variables are prefixed with ?.

\[
\begin{align*}
   r(X,Y) & :- p(X,Y) & \& \neg q(Y) \\
   (\leq (r \ ?x \ ?y) \ (\text{and} \ (p \ ?x \ ?y) \ (\text{not} \ (q \ ?y)))) \\
   (\leq (r \ ?x \ ?y) \ (p \ ?x \ ?y) \ (\text{not} \ (q \ ?y)))
\end{align*}
\]

Semantics is the same.
Agent Communication Language

Start Message

\[(\text{start id role } (s_1 \ldots s_n) \text{ startclock playclock})\]

Play Message

\[(\text{play id } (a_1 \ldots a_k))\]

Stop Message

\[(\text{stop id } (a_1 \ldots a_k))\]

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 primitive moves and the does relation.

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 is weakly winnable, and all single player games are strongly winnable.