Learning and Playing Board Games from the FMT Perspective

Łukasz Kaiser

LIAFA, CNRS & Université Paris 7

FMT Workshop, Les Houches, 2012

Motivation

LFP+C captures **PTIME**

- on **ordered structures** **(Immerman, Vardi ’82)**
- on **planar graphs** **(Grohe ’98)**
- on classes **excluding a minor** **(Grohe ’10)**

Motivation

LFP+C captures **PTIME**

- on **ordered structures (Immerman, Vardi ’82)**
- on **planar graphs (Grohe ’98)**
- on classes **excluding a minor (Grohe ’10)**

LFP+C

PTIME

```
int main() {
  for (i=0; i<64; i++) {
    a[i] = f(i);
  }
  return (0);
}
```

Motivation

LFP+C captures **PTIME**

- on **ordered structures** **(Immerman, Vardi '82)**
- on **planar graphs** **(Grohe '98)**
- on classes **excluding a minor** **(Grohe '10)**

LFP+C

```
DgA(x,y) = ex u (R(x,u) and C(u,y))
DgB(x,y) = ex u (R(x,u) and C(y,u))
Row3(x,y,z) = R(x,y) and R(y,z)
Col3(x,y,z) = C(x,y) and C(y,z)
DgA3(x,y,z) = DgA(x,y) and DgA(y,z)
DgB3(x,y,z) = DgB(x,y) and DgB(y,z)
```

PTIME

```
int main() {
  for (i=0; i<64; i++) {
    a[i] = f(i);
  }
  return (0);
}
```

Motivation

LFP+C captures **PTIME**

- on **ordered structures (Immerman, Vardi '82)**
- on **planar graphs (Grohe '98)**
- on classes **excluding a minor (Grohe '10)**

LFP+C

```
DgA(x,y) = ex u (R(x,u) and C(u,y))
DgB(x,y) = ex u (R(x,u) and C(y,u))
Row3(x,y,z) = R(x,y) and R(y,z)
Col3(x,y,z) = C(x,y) and C(y,z)
DgA3(x,y,z) = DgA(x,y) and DgA(y,z)
DgB3(x,y,z) = DgB(x,y) and DgB(y,z)
```

PTIME

```
int main() {
  for (i=0; i<64; i++) {
    a[i] = f(i);
  }
  return (0);
}
```

When does **LFP+C programming** make sense?

Two Board Game Problems from AI

General Game Playing

- **Input:** game in Game Description Language (**GDL**)
  GDL: a variant of Datalog for games **(Genesereth, Love, '05)**
- **Goal:** playing the game **competitively** a few minutes later
- **GGP Competition:** yearly event in which GGP programs compete

Learning from Videos

- **Goal:** a robot watches a game and learns the rules
- **Tic-Tac-Toe** solved in **(Barbu, Narayanaswamy, Siskind '10)**
- **ILP** for rule induction using **Progol (Muggleton '95)**

Two Board Game Problems from AI

General Game Playing

- **Input:** game in Game Description Language (**GDL**)
  GDL: a variant of Datalog for games **(Genesereth, Love, '05)**
- **Goal:** playing the game **competitively** a few minutes later
- **GGP Competition:** yearly event in which GGP programs compete

Learning from Videos

- **Goal:** a robot watches a game and learns the rules
- **Tic-Tac-Toe** solved in **(Barbu, Narayanaswamy, Siskind '10)**
- **ILP** for rule induction using **Progol (Muggleton '95)**
  Problem: hand-crafted **state representation**, **what** does it learn?

Representing Game Boards

Representing Game Boards

```
inc(X,Y):-Y is X+1.

dec(X,Y):-Y is X-1.

player(player_x).
player(player_o).

opponent(player_x,player_o).
opponent(player_o,player_x).

piece(x).
piece(o).
piece(none).

empty(none).

owns(player_x,x).
owns(player_o,o).

win_outcome(x_wins).
win_outcome(o_wins).

owns_outcome(player_x,x_wins).
owns_outcome(player_o,o_wins).

owns_piece(x_wins,x).
owns_piece(o_wins,o).

row(r0).
row(r1).
row(r2).

col(c0).
col(c1).
col(c2).

row_to_int(r0,0).
row_to_int(r1,1).
row_to_int(r2,2).
```

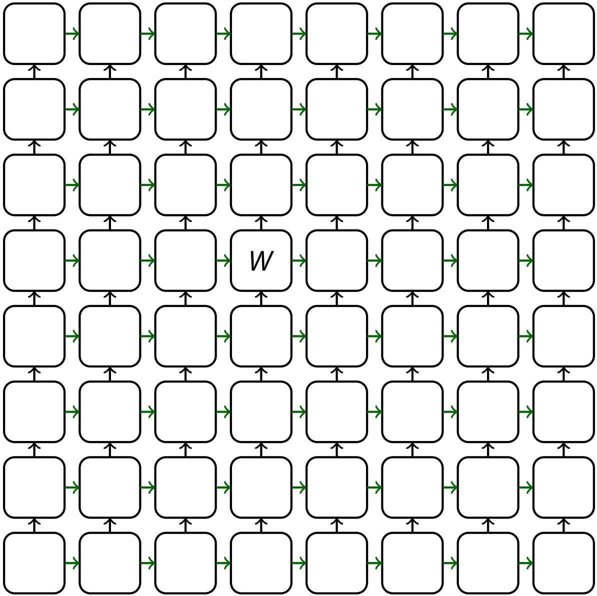
```
col_to_int(c0,0).
col_to_int(c1,1).
col_to_int(c2,2).

board([[A,B,C],[D,E,F],[G,H,I]]):-
        piece(A),piece(B),piece(C),
        piece(D),piece(E),piece(F),
        piece(G),piece(H),piece(I).

ref(0,[H|_],H).
ref(0,_,_):-!,fail.
ref(1,[_,B,_],B).
ref(1,_,_):-!,fail.
ref(2,[_,_,C],C).
ref(2,_,_):-!,fail.
ref(I,[H|T],H2):-dec(I,J),ref(J,T,H2).

at(ROW,COLUMN,BOARD,PIECE):-
        row_to_int(ROW,RI),
        col_to_int(COLUMN,CI),!,
        ref(RI,BOARD,L),!,ref(CI,L,PIECE).
at(ROW,COLUMN,BOARD,PIECE,
    [ROW,COLUMN,PIECE]):-
        at(ROW,COLUMN,BOARD,PIECE).

replace(0,A,[_|T],[A|T]):-!.
replace(0,_,_,_):-!,fail.
replace(1,A,[X,_|T],[X,A|T]):-!.
replace(1,_,_,_):-!,fail.
replace(2,A,[X,Y,_|T],[X,Y,A|T]):-!.
replace(2,_,_,_):-!,fail.
replace(I,H1,[H2|T1],[H2|T2]):-
        dec(I,J),replace(J,H1,T1,T2).

frame(RROW,CCOLUMN,BOARD1,BOARD2):-
        row(RROW),
        col(CCOLUMN),
        row_to_int(RROW,ROW),
        col_to_int(CCOLUMN,COLUMN),
        ref(ROW,BOARD1,L1),
        replace(ROW,L2,BOARD1,BOARD3),
        replace(COLUMN,ignore,L1,L2),
        ref(ROW,BOARD2,L3),
        replace(ROW,L4,BOARD2,BOARD3),
        replace(COLUMN,ignore,L3,L4),!,
        board(BOARD2).
```

```
frame_obj([R1,C1,P1],
          [R2,C2,P2],
          [R1,C1,P3],
          [R2,C2,P4],
          B1,
          B2):-
        frame(R1,C1,B1,B3),
        at(R1,C1,B1,P1),
        at(R2,C2,B1,P2),
        frame(R2,C2,B3,B2),
        at(R1,C1,B2,P3),
        at(R2,C2,B2,P4).

forward(player_x,R1,R2):-
        row_to_int(R1,RI1),
        inc(RI1,RI2),
        row_to_int(R2,RI2).
forward(player_o,R1,R2):-
        row_to_int(R1,RI1),
        dec(RI1,RI2),
        row_to_int(R2,RI2).

sideways(C1,C2):-
        col_to_int(C1,CI1),
        inc(CI1,CI2),
        col_to_int(C2,CI2).
sideways(C1,C2):-
        col_to_int(C1,CI1),
        dec(CI1,CI2),
        col_to_int(C2,CI2).

linear_test(X1,Y1,X2,Y2,X3,Y3,S):-
        S is X1*(Y2-Y3)+X2*(Y3-Y1)+
             X3*(Y1-Y2).

linear(R1,C1,R2,C2,R3,C3):-
        [R1,C1]\=[R2,C2],
        [R3,C3]\=[R2,C2],
        [R1,C1]\=[R3,C3],
        row_to_int(R1,RI1),
        row_to_int(R2,RI2),
        row_to_int(R3,RI3),
        col_to_int(C1,CI1),
        col_to_int(C2,CI2),
        col_to_int(C3,CI3),
        linear_test(RI1,CI1,RI2,
                    CI2,RI3,CI3,0).
```

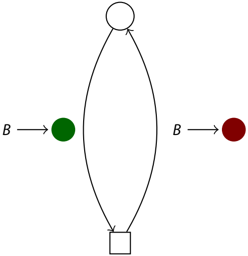
Fig. 4. Background knowledge encoded in PROLOG. PROGOL-specific settings and mode, type, and pruning declarations have been omitted.

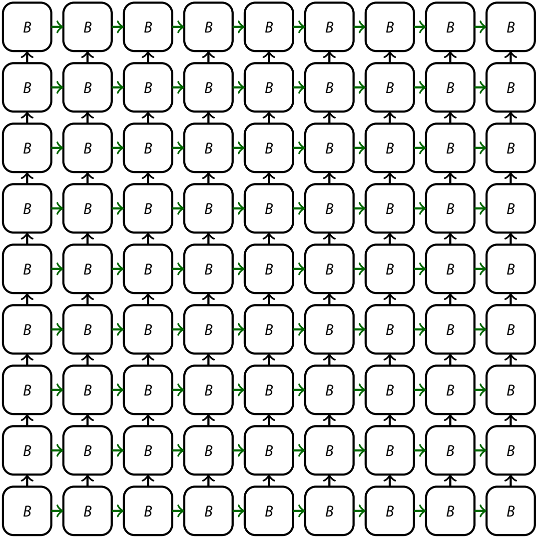
Representing Game Boards

Representing Game Boards

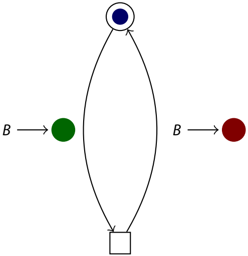


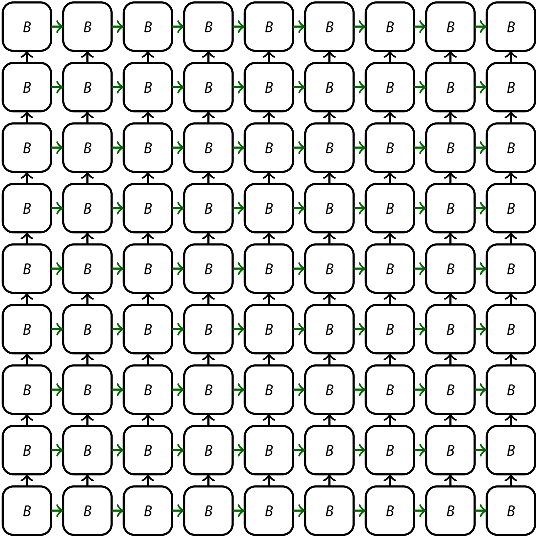
Example Game: Gomoku (Connect–5)



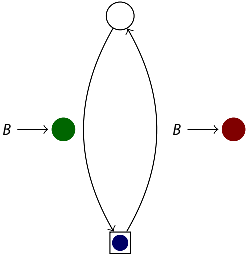


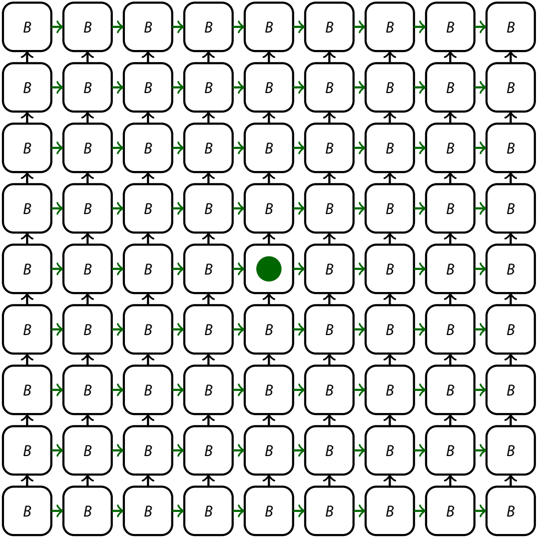
Example Game: Gomoku (Connect–5)



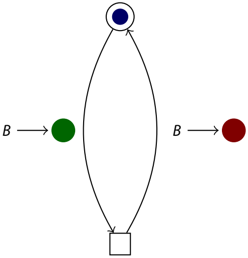


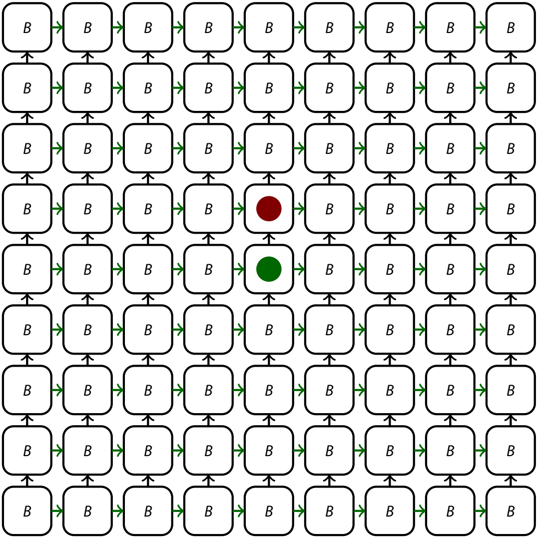
Example Game: Gomoku (Connect–5)



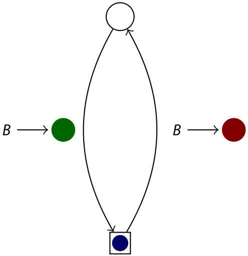


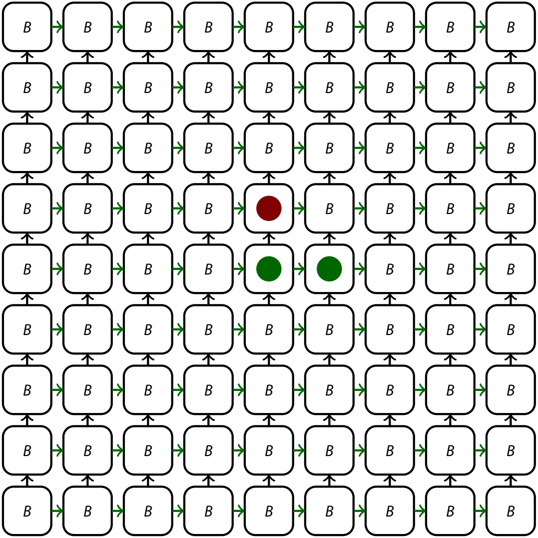
Example Game: Gomoku (Connect–5)



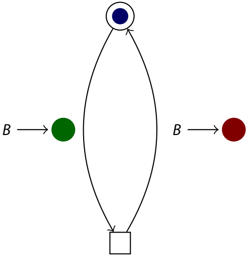


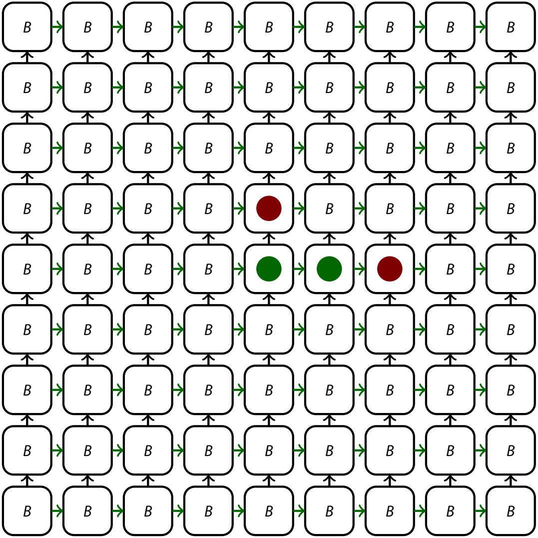
Example Game: Gomoku (Connect–5)



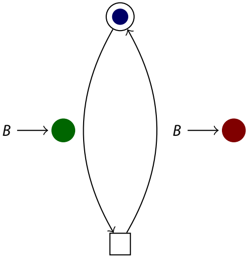


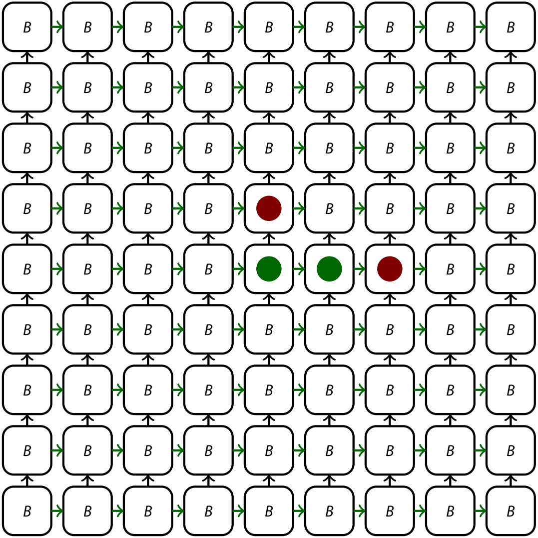
Example Game: Gomoku (Connect–5)





Example Game: Gomoku (Connect–5)





$$\chi\Big[\exists x_1 \dots x_5 \Big(\bigwedge_{1 \le i \le 5} G(x_i) \wedge \Big(\bigwedge_{1 \le i \le 5} R(x_i, x_{i+1}) \vee \bigwedge_{1 \le i \le 5} C(x_i, x_{i+1}) \vee$$

$$\bigwedge_{1 \le i \le 5} \exists y (R(x_i, y) \wedge C(y, x_{i+1})) \vee \bigwedge_{1 \le i \le 5} \exists y (R(x_i, y) \wedge C(x_{i+1}, y)) \Big) \Big) \Big]$$

First-Order Logic with Counting

Syntax

$$\begin{aligned}\varphi \;&:=\; R_i(x_1,\ldots,x_{r_i}) \mid x_i = x_j \mid \rho < \rho \\ &\qquad \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x_i\, \varphi \mid \forall x_i\, \varphi \\ \rho \;&:=\; \frac{n}{m} \mid \rho + \rho \mid \rho \cdot \rho \mid \chi[\varphi] \mid \sum_{\overline{x} \mid \varphi} \rho\end{aligned}$$

Semantics as expected, with

- $\chi[\varphi(\overline{x})] = 1$ iff $\varphi(\overline{x})$ **holds**
- $\sum_{\overline{x}|\varphi} \rho$ sums $\rho(\overline{x})$ over $\overline{x}$ **satisfying** $\varphi(\overline{x})$

First-Order Logic with Counting

Syntax

$$\varphi \;:=\; R_i(x_1, \ldots, x_{r_i}) \mid x_i = x_j \mid \rho < \rho$$
$$\mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x_i\, \varphi \mid \forall x_i\, \varphi$$
$$\rho \;:=\; \frac{n}{m} \mid \rho + \rho \mid \rho \cdot \rho \mid \chi[\varphi] \mid \sum_{\overline{x} \mid \varphi} \rho$$

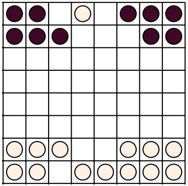
Semantics as expected, with

- $\chi[\varphi(\overline{x})] = 1$ iff $\varphi(\overline{x})$ **holds**
- $\sum_{\overline{x}|\varphi} \rho$ sums $\rho(\overline{x})$ over $\overline{x}$ **satisfying** $\varphi(\overline{x})$

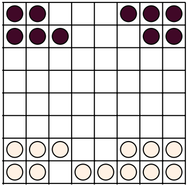
Example from Chess

$$\sum_{x \,\mid\, \mathbf{bBeats}(x)} 1 + \chi[\mathbf{w}(x)] + 3 \cdot \chi[\mathbf{wK}(x)]$$

Learning Winning Conditions



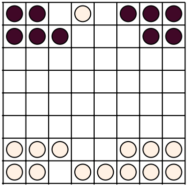
Positive Example $\mathfrak{A}$



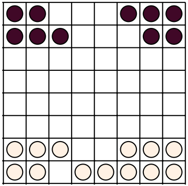
Negative Example $\mathfrak{B}$

Find **minimal** φ such that $\mathfrak{A} \models \varphi, \mathfrak{B} \models \neg\varphi$

Learning Winning Conditions



Positive Example $\mathfrak{A}$



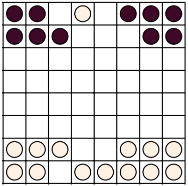
Negative Example $\mathfrak{B}$

Find **minimal** φ such that $\mathfrak{A} \models \varphi, \mathfrak{B} \models \neg\varphi$

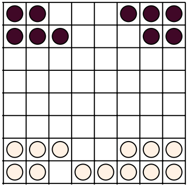
Which Logic?

- Full FO, minimal quantifier rank: **PSPACE-complete (Pezzoli ’98)**
- FO^k, minimal quantifier rank: **PTIME (Grohe ’99)**
- $k = 16$ and $\log(n)$ quantifiers suffice for …**(Pikhurko, Verbitsky ’10)**

Learning Winning Conditions



Positive Example $\mathfrak{A}$



Negative Example $\mathfrak{B}$

Find **minimal** φ such that $\mathfrak{A} \models \varphi$, $\mathfrak{B} \models \neg\varphi$

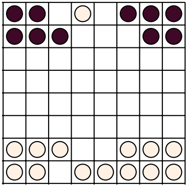
Which Logic?

- Full FO, minimal quantifier rank: **PSPACE-complete (Pezzoli '98)**
- FO^k, minimal quantifier rank: **PTIME (Grohe '99)**
- $k = 16$ and $\log(n)$ quantifiers suffice for …**(Pikhurko, Verbitsky '10)**

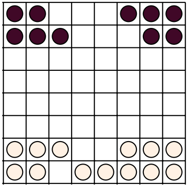
Extensions

- Use the *m*-step TC^m operator **(lower qr, nice formulas)**
- Compute **guarded** formulas first **(complexity)**
- **Shortest** φ with minimal quantifier rank? **(at present: greedy removal)**

Learning Winning Conditions



Positive Example $\mathfrak{A}$



Negative Example $\mathfrak{B}$

Find **minimal** φ such that $\mathfrak{A} \models \varphi, \mathfrak{B} \models \neg\varphi$

Which Logic?

- Full FO, minimal quantifier rank: **PSPACE-complete** **(Pezzoli '98)**
- FO^k, minimal quantifier rank: **PTIME** **(Grohe '99)**
- $k = 16$ and $\log(n)$ quantifiers suffice for … **(Pikhurko, Verbitsky '10)**

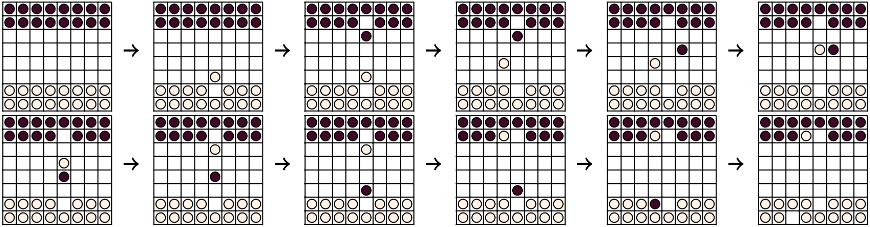
Extensions

- Use the m-step TC^m operator **(lower qr, nice formulas)**
- Compute **guarded** formulas first **(complexity)**
- **Shortest** φ with minimal quantifier rank? **(at present: greedy removal)**

Computed Formula: $\exists x\ (\mathbf{W}(x)\ \wedge\ \forall y\ \neg\mathbf{C}(x, y))$

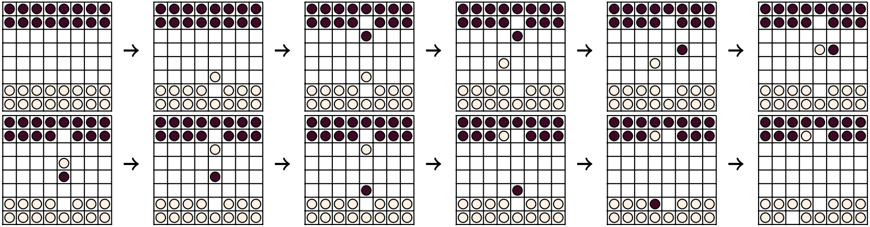
Learning Moves

Example Input: a video of a sequence of Breakthrough moves

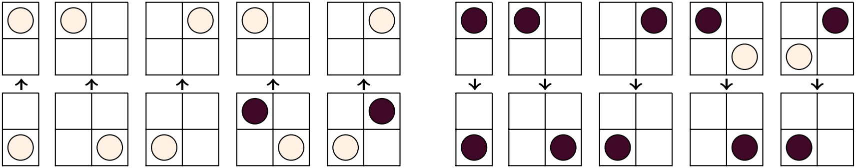


Learning Moves

Example Input: a video of a sequence of Breakthrough moves

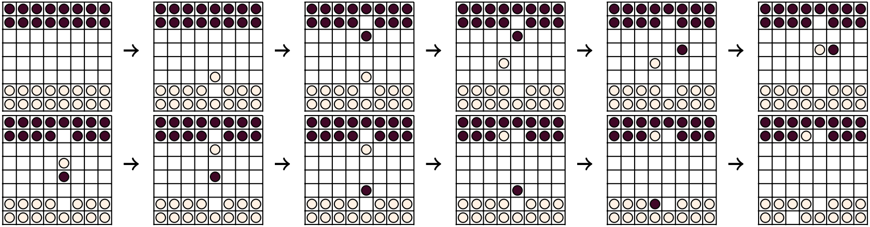


Derived Structure Rewriting Rules correspond **directly** to moves

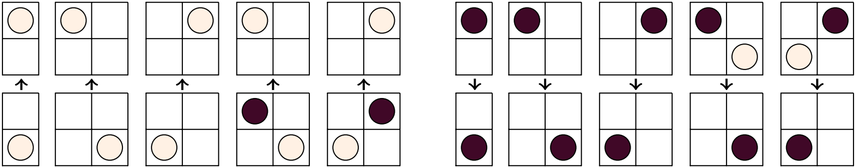


Learning Moves

Example Input: a video of a sequence of Breakthrough moves



Derived Structure Rewriting Rules correspond **directly** to moves



Preconditions are derived from **illegal** moves

Learning Results (K., AAAI-12)

	1 Wins	2 Wins	Not Won	Illegal	Learning Time
Breakthrough	1	1	3	0	168 s
Connect4	4	4	13	4	164 s
Gomoku	4	4	9	0	57 s
Pawn Whopping	1	1	4	6	980 s
Tic-Tac-Toe	4	4	17	0	36 s

Table: Number of videos needed for each game and learning time

See the videos and code at:

`http://toss.sf.net/learn.html`

Deriving Position Evaluation Functions

Methods **(K., Stafiniak, AAAI-11)**

- Goal expansion and Type Normal Form
- Summing over conjuncts in existentially quantified conjunctions
- Retain stable guards and include rule preconditions

Example: Tic-Tac-Toe without Diagonals

$$\big(P(x)\wedge P(y)\wedge P(z)\wedge R(x,y)\wedge R(y,z)\big)\vee\big(P(x)\wedge P(y)\wedge P(z)\wedge C(x,y)\wedge C(y,z)\big)$$

$$\rightsquigarrow$$

$$\sum_{x|P(x)}\Big(\tfrac{1}{8}+\sum_{y|P(y)\wedge R(x,y)}\big(\tfrac{1}{4}+\sum_{z|P(z)\wedge R(y,z)}1\big)\Big)+\sum_{x|P(x)}\Big(\tfrac{1}{8}+\sum_{y|P(y)\wedge C(x,y)}\big(\tfrac{1}{4}+\sum_{z|P(z)\wedge C(y,z)}1\big)\Big)$$

Deriving Position Evaluation Functions

Methods (K., Stafiniak, AAAI-11)

- Goal expansion and Type Normal Form
- Summing over conjuncts in existentially quantified conjunctions
- Retain stable guards and include rule preconditions

Example: Tic-Tac-Toe without Diagonals

$$\big(P(x) \wedge P(y) \wedge P(z) \wedge R(x,y) \wedge R(y,z)\big) \vee \big(P(x) \wedge P(y) \wedge P(z) \wedge C(x,y) \wedge C(y,z)\big)$$

$$\rightsquigarrow$$

$$\sum_{x|P(x)} \Big(\tfrac{1}{8} + \sum_{y|P(y)\wedge R(x,y)} \big(\tfrac{1}{4} + \sum_{z|P(z)\wedge R(y,z)} 1\big)\Big) + \sum_{x|P(x)} \Big(\tfrac{1}{8} + \sum_{y|P(y)\wedge C(x,y)} \big(\tfrac{1}{4} + \sum_{z|P(z)\wedge C(y,z)} 1\big)\Big)$$

Results vs. Fluxplayer

(fixed depth)

	Toss Wins	Fluxplayer Wins	Tie
Breakthrough	95%	5%	0%
Connect4	20%	75%	5%
Connect5	0%	0%	100%
Pawn Whopping	50%	50%	0%

Deriving Position Evaluation Functions

Methods (K., Stafiniak, AAAI-11)

- Goal expansion and Type Normal Form
- Summing over conjuncts in existentially quantified conjunctions
- Retain stable guards and include rule preconditions

Example: Tic-Tac-Toe without Diagonals

$$\big(P(x) \wedge P(y) \wedge P(z) \wedge R(x,y) \wedge R(y,z)\big) \vee \big(P(x) \wedge P(y) \wedge P(z) \wedge C(x,y) \wedge C(y,z)\big)$$

$$\rightsquigarrow$$

$$\sum_{x|P(x)} \Big(\tfrac{1}{8} + \sum_{y|P(y)\wedge R(x,y)} \big(\tfrac{1}{4} + \sum_{z|P(z)\wedge R(y,z)} 1\big)\Big) + \sum_{x|P(x)} \Big(\tfrac{1}{8} + \sum_{y|P(y)\wedge C(x,y)} \big(\tfrac{1}{4} + \sum_{z|P(z)\wedge C(y,z)} 1\big)\Big)$$

Results vs. Fluxplayer

(variable depth)

	Toss Wins	Fluxplayer Wins	Tie
Breakthrough	95%	5%	0%
Connect4	45%	20%	35%
Connect5	0%	0%	100%
Pawn Whopping	60%	40%	0%

Programming in LFP+C?

Conclusions

- **weaker logics** can be useful
- **learning** can be in PTIME

Programming in LFP+C?

Conclusions

- **weaker logics** can be useful
- **learning** can be in PTIME

Open Problems

- Fast **FO solver**

Programming in LFP+C?

Conclusions

- **weaker logics** can be useful
- **learning** can be in PTIME

Open Problems

- Fast **FO solver**

$$\exists x, y, z\big(P(x) \wedge P(y) \wedge P(z) \wedge R(x,y) \wedge R(y,z)\big)$$

$$\rightsquigarrow$$

$$\exists x\,(P(x) \wedge \exists y\,(R(x,y) \wedge P(y) \wedge \exists z\,(R(y,z) \wedge P(z))))$$

Programming in LFP+C?

Conclusions

- **weaker logics** can be useful
- **learning** can be in PTIME

Open Problems

- Fast **FO solver**

$$\exists x, y, z\big(P(x) \wedge P(y) \wedge P(z) \wedge R(x,y) \wedge R(y,z)\big)$$

$$\rightsquigarrow$$

$$\exists x\,(P(x) \wedge \exists y\,(R(x,y) \wedge P(y) \wedge \exists z\,(R(y,z) \wedge P(z))))$$

- How to **distinguish structures** efficiently?
- Better logics to get **nice formulas**

Programming in LFP+C?

Conclusions

- **weaker logics** can be useful
- **learning** can be in PTIME

Open Problems

- Fast **FO solver**

$$\exists x, y, z\big(P(x) \wedge P(y) \wedge P(z) \wedge R(x,y) \wedge R(y,z)\big)$$

$$\rightsquigarrow$$

$$\exists x\,(P(x) \wedge \exists y\,(R(x,y) \wedge P(y) \wedge \exists z\,(R(y,z) \wedge P(z))))$$

- How to **distinguish structures** efficiently?
- Better logics to get **nice formulas**

`www.toss.sf.net`