Learning programs with magic values

Céline Hocquette, Andrew Cropper
University of Oxford

UK Research and Innovation

UNIVERSITY OF OXFORD

Motivation

A *magic value* is a constant symbol in a program which has no clear explanation for its choice: it *magically* works.

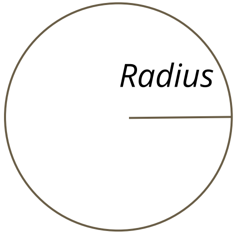
Positive examples	**Negative examples**
[a,e,6,7,q,2]	[6,e,a,2,q,6,e]
[p,3,9,y,5,r,a,q,7]	[u,k,a,b,c,z,r,t,5,e,t]

$f(A) \leftarrow head(A,\mathbf{7})$

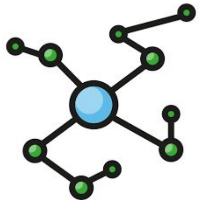
$f(A) \leftarrow tail(A,B), f(B)$

Motivation

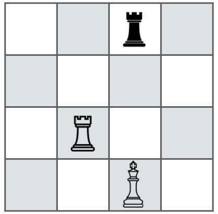
area(Radius,Area) ←
square(Radius,SqRadius),
*mult(SqRadius, **3.14**, Area).*



drug(Drug) ←
atom(Drug,Atom1),
atom(Drug,Atom2),
*atomtype(Atom1,**oxygen**),*
*atomtype(Atom2,**hydrogen**),*
*distance(Atom1,Atom2,**0.53**)*



rookprotected(State) ←
*piece(State1,Piece1,**white**,**rook**),*
*piece(State,Piece2,**white**,**king**),*
*distance(Piece1,Piece2,**1**).*



Existing approaches

Bottom clause (Progol, Aleph):

- bottom clause can grow large
- limited recursion and lack of predicate invention

Existing approaches

Precompute every possible rule in the hypothesis space (ASPAL, ILASP, HEXMIL, ProSynth)

H1: $f(A) \leftarrow head(A,1)$

H2: $f(A) \leftarrow head(A,2)$

H3: $f(A) \leftarrow head(A,3)$

H4: $f(A) \leftarrow head(A,4)$

H5: $f(A) \leftarrow head(A,5)$

H6: $f(A) \leftarrow head(A,6)$

...

Existing approaches

Unary predicate symbols, one for each constant symbol (Popper, δILP)

H1: $f(A) \leftarrow head(A,B), \text{c1}(B)$

H2: $f(A) \leftarrow head(A,B), \text{c2}(B)$

H3: $f(A) \leftarrow head(A,B), \text{c3}(B)$

H4: $f(A) \leftarrow head(A,B), \text{c4}(B)$

H5: $f(A) \leftarrow head(A,B), \text{c5}(B)$

H6: $f(A) \leftarrow head(A,B), \text{c6}(B)$

...

Existing approaches: limitations

Enumeration of constant symbols

- cannot scale to large or infinite domains
- suffer from performance issue

Our approach

<table>
<tr><th colspan="2">Existing approaches</th><th>Our approach</th></tr>
<tr><td>$f(List) \leftarrow head(List,1).$</td><td>$f(List) \leftarrow head(List,E), c1(E).$</td><td rowspan="7">$f(A) \leftarrow head(List,\mathbf{E}),$
$@magic(\mathbf{E}).$</td></tr>
<tr><td>$f(List) \leftarrow head(List,2).$</td><td>$f(List) \leftarrow head(List,E), c2(E).$</td></tr>
<tr><td>$f(List) \leftarrow head(List,3).$</td><td>$f(List) \leftarrow head(List,E), c3(E).$</td></tr>
<tr><td>$f(List) \leftarrow head(List,4).$</td><td>$f(List) \leftarrow head(List,E), c4(E).$</td></tr>
<tr><td>$f(List) \leftarrow head(List,5).$</td><td>$f(List) \leftarrow head(List,E), c5(E).$</td></tr>
<tr><td>$f(List) \leftarrow head(List,6).$</td><td>$f(List) \leftarrow head(List,E), c6(E).$</td></tr>
<tr><td>..</td><td>...</td></tr>
</table>

Related Work

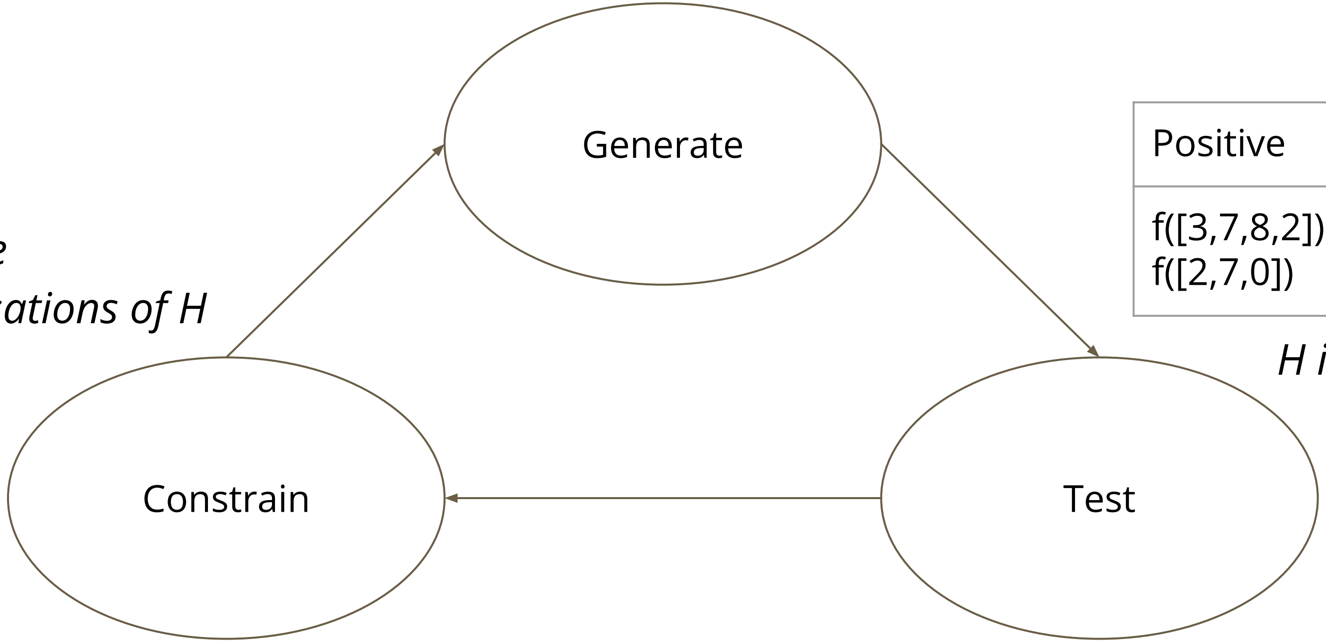
Our approach is inspired by Aleph’s lazy evaluation procedure:

- can learn constants from reasoning from multiple examples
- limited learning of recursion and lack of predicate invention
- requires strong user bias

Learning From Failures

H : f(List) ← head(List,E), c1(E).

we prune generalisations of H

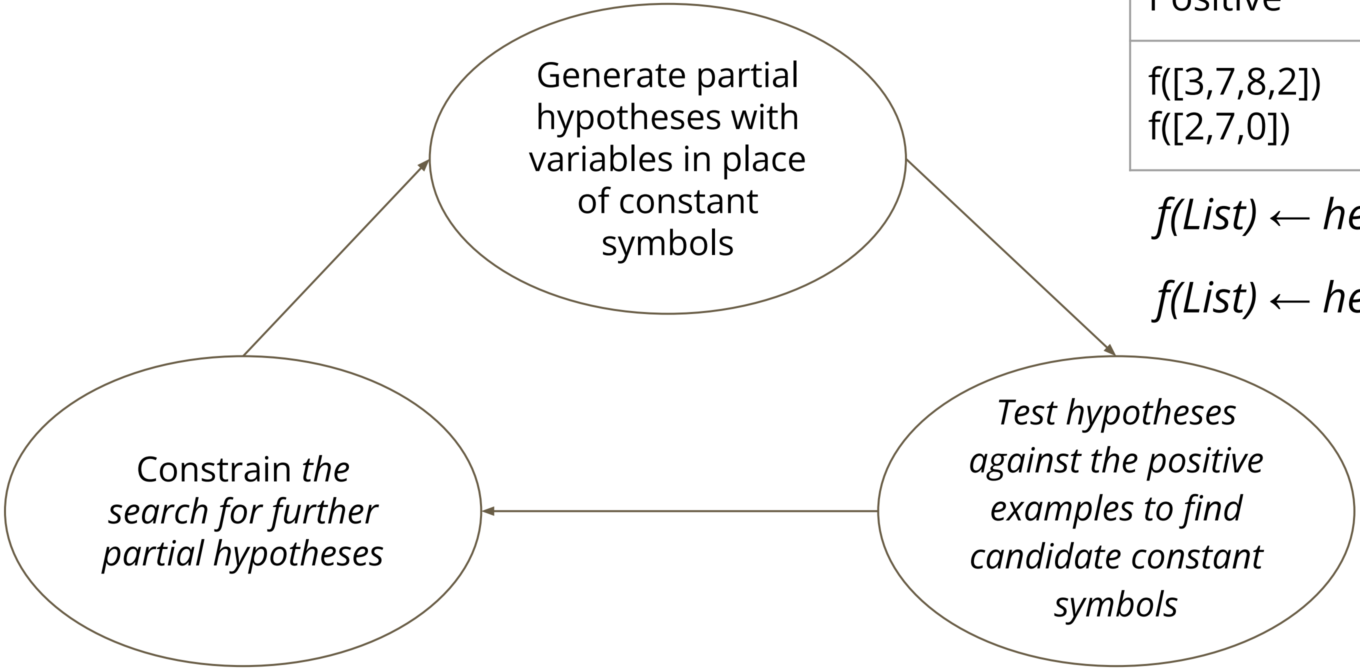


Positive	Negative
f([3,7,8,2]) f([2,7,0])	f([8,0,4]) f([1,2])

H is inconsistent

Our approach

H : *f(List) ← head(List,E), @magic(E)*



Positive	Negative
f([3,7,8,2]) f([2,7,0])	f([8,0,4]) f([1,2])

f(List) ← head(List,3).

f(List) ← head(List,2).

Implementation

We implement our approach in MagicPopper

Based on the LFF learner Popper

Experiments

Q1: How well does MagicPopper perform compared to other approaches?

Q1: comparison with other approaches

Task	Aleph	Metagol	Popper	MagicPopper
md	**100 ± 0**	50 ± 0	**100 ± 0**	**100 ± 0**
buttons-next	81 ± 0	50 ± 0	**100 ± 0**	**100 ± 0**
coins-next	50 ± 0	50 ± 0	**100 ± 0**	**100 ± 0**
buttons-goal	**100 ± 0**	50 ± 0	98 ± 1	**100 ± 0**
coins-goal	50 ± 0	50 ± 0	**100 ± 0**	**100 ± 0**
gt-centipede-goal	**99 ± 0**	50 ± 0	75 ± 0	75 ± 0
gt-centipede-legal	**100 ± 0**	50 ± 0	**100 ± 0**	**100 ± 0**
gt-centipede-next	**100 ± 0**	50 ± 0	**100 ± 0**	**100 ± 0**
krk	**100 ± 0**	54 ± 4	96 ± 1	99 ± 0
list	50 ± 0	**100 ± 0**	49 ± 0	**100 ± 0**
powerof2	86 ± 1	58 ± 5	84 ± 1	**100 ± 0**
append	95 ± 1	**99 ± 0**	96 ± 1	96 ± 1

Predictive accuracies

Q1: comparison with other approaches

Task	Aleph	Metagol	Popper	MagicPopper
md	**0 ± 0**		1 ± 0	**0 ± 0**
buttons-next	32 ± 1		**3 ± 0**	4 ± 0
coins-next			**53 ± 0**	99 ± 1
buttons-goal	**0 ± 0**		1 ± 0	**0 ± 0**
coins-goal			**0 ± 0**	**0 ± 0**
gt-centipede-goal	**0 ± 0**		23 ± 0	6 ± 0
gt-centipede-legal	**0 ± 0**		4 ± 0	1 ± 0
gt-centipede-next	**0 ± 0**		10 ± 0	**0 ± 0**
krk	**0 ± 0**		35 ± 6	6 ± 0
list		36 ± 8		**2 ± 0**
powerof2	**0 ± 0**		18 ± 0	**0 ± 0**
append	1 ± 0	**0 ± 0**	298 ± 49	**0 ± 0**

Learning times

Q1: comparison with other approaches

- **MagicPopper can outperform existing approaches in terms of learning times and predictive accuracies**

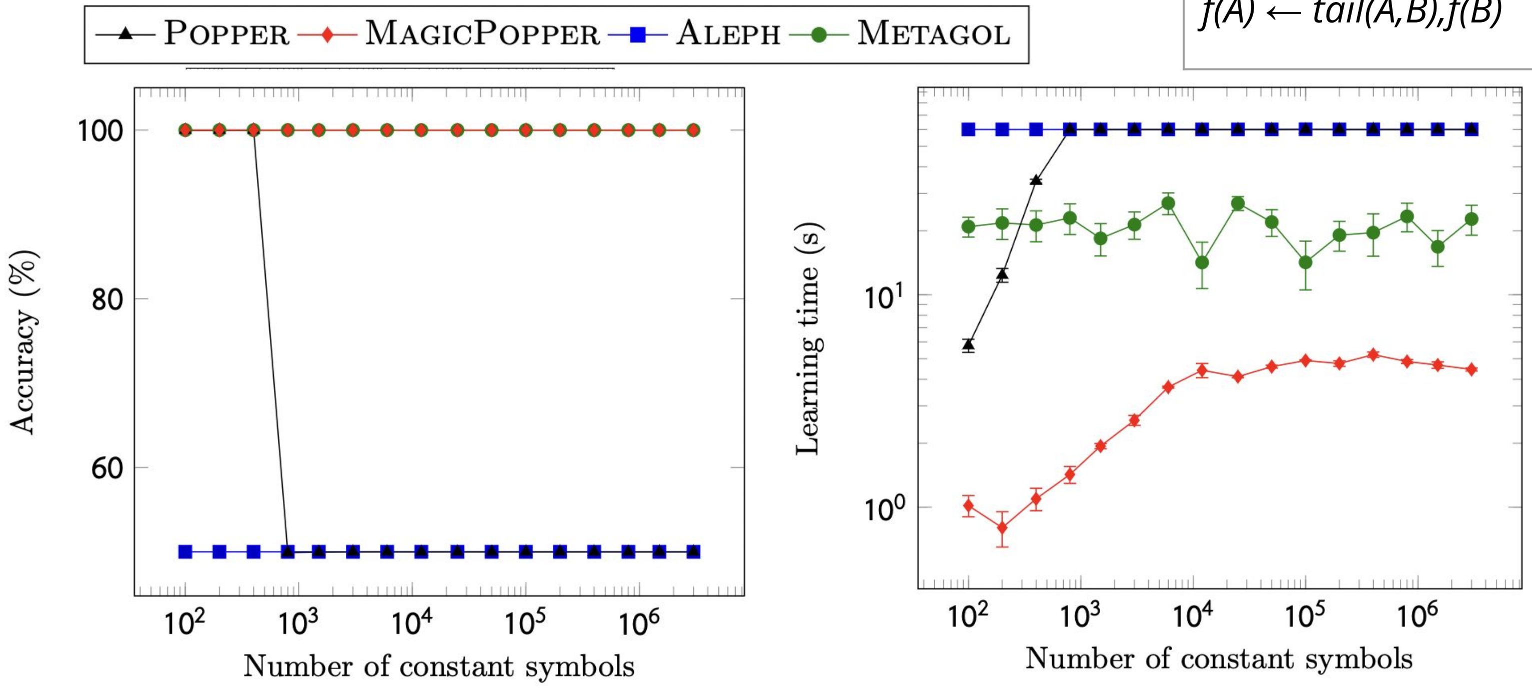
Experiments

Q2: How well does MagicPopper scale with the number of constant symbols?

Q2: scalability with respect to the number of constant symbols

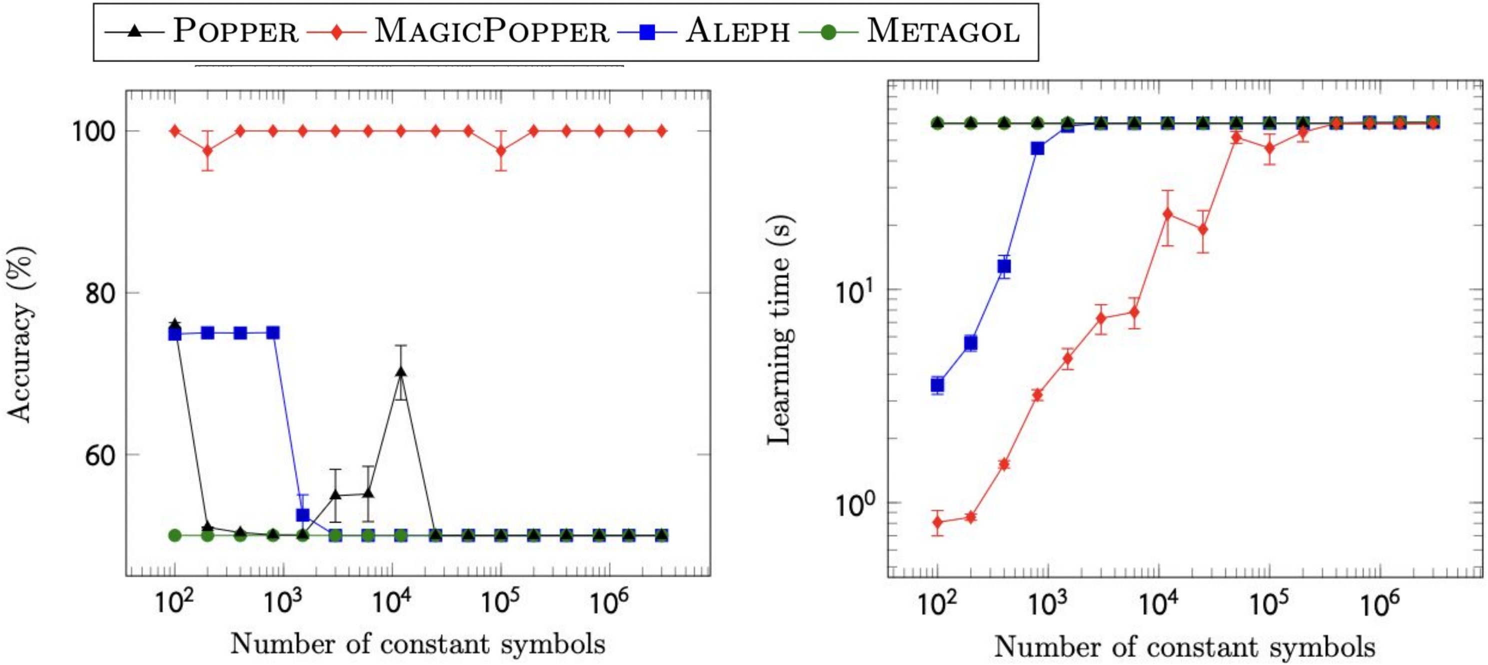
f(A) ← head(A,**7**)

f(A) ← tail(A,B),f(B)



Q2: scalability with respect to the number of constant symbols

*next_val(A,**5**) ← does(A,**player**,**press_button**)*
*next_val(A,B) ← does(A,**player**,**noop**), true_val(A,C), succ(B,C)*



Q2: scalability with respect to the number of constant symbols

- **MagicPopper can scale well with the number of constant symbols, up to millions**

Experiments

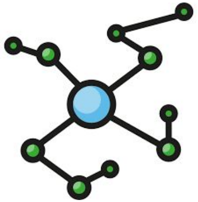
Q3: Can MagicPopper learn in infinite domains?

Q3: learning in infinite domains

Task	Aleph	Metagol	Popper	MagicPopper
pi	**100 ± 0**	50 ± 0	50 ± 0	99 ± 0
equilibrium	**100 ± 0**	50 ± 0	62 ± 1	86 ± 7
drug design	63 ± 7	50 ± 0	50 ± 0	**98 ± 0**
next	50 ± 0	50 ± 0	49 ± 0	**100 ± 0**
sumk	50 ± 0	50 ± 0	50 ± 0	**100 ± 0**

Predictive accuracies

drug(Drug) ←
atom(Drug,Atom1),
atom(Drug,Atom2),
*atomtype(Atom1,**oxygen**),*
*atomtype(Atom2,**hydrogen**),*
*distance(Atom1,Atom2,**0.53**)*

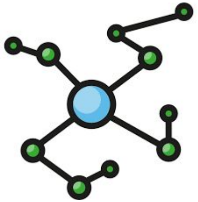


Q3: learning in infinite domains

Task	Aleph	Metagol	Popper	MagicPopper
pi	4 ± 1			1 ± 0
equilibrium	**0 ± 0**			72 ± 17
drug design				6 ± 3
next				25 ± 0
sumk				99 ± 1

Learning times

drug(Drug) ←
atom(Drug,Atom1),
atom(Drug,Atom2),
*atomtype(Atom1,**oxygen**),*
*atomtype(Atom2,**hydrogen**),*
*distance(Atom1,Atom2,**0.53**)*



Q3: learning in infinite domains

- **MagicPopper can learn in infinite domains**

Conclusion

MagicPopper, approach to learn programs with magic values. It can:

- outperform state-of-the art approaches,
- scale to domains with millions of constant symbols,
- learn in continuous domains,
- learn optimal programs and recursive programs.

Future Work and Limitations

- Noise: identify magic values from noisy examples
- Numerical reasoning from multiple examples (eg identify thresholds)

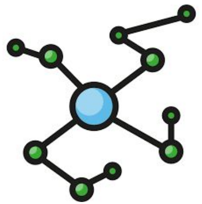
References

- Corapi, D., Russo, A., Lupu, E.: Inductive logic programming in answer set programming. In: Inductive Logic Programming - 21st International Conference, (2011).
- Law, M., Russo, A., Broda, K.: The ILASP system for learning Answer Set Programs. www.ilasp.com (2015).
- Kaminski, T., Eiter, T., Inoue, K.: Exploiting answer set programming with external sources for meta-interpretive learning. Theory and Practice of Logic Programming 18(3-4), (2018).
- Raghothaman, M., Mendelson, J., Zhao, D., Naik, M., Scholz, B.: Provenance-guided synthesis of datalog programs. Proceedings of the ACM on Programming Languages 4(POPL), (2019).
- Cropper, A., Morel, R.: Learning programs by learning from failures. Machine Learning 110(4), 801–856 (2021).
- Evans, R., Grefenstette, E.: Learning explanatory rules from noisy data. Journal of Artificial Intelligence Research 61, (2018).
- Srinivasan, A.: The ALEPH manual. Machine Learning at the Computing Laboratory (2001).
- Srinivasan, A., Camacho, R.: Numerical reasoning with an ILP system capable of lazy evaluation and customised search. The Journal of Logic Programming 40(2), 185–213 (1999).
- Muggleton, S.H., Lin, D., Pahlavi, N., Tamaddoni-Nezhad, A.: Meta- interpretive learning: application to grammatical inference. Machine Learning 94, (2014).

Tasks

equilibrium(Object) ←
forces(Object,Forces),
sum(Forces,Sum), mass(Object,Mass),
mult(Mass, ***9.81****, Sum).*

drug(Drug) ←
atom(Drug,Atom1),
atom(Drug,Atom2),
*atomtype(Atom1,****oxygen****),*
*atomtype(Atom2,****hydrogen****),*
*distance(Atom1,Atom2,****0.53****)*



next(A,**q**) ← my true(A,**q**),does(A,**robot**,**a**)
next(A,**p**) ← my true(A,**q**),does(A,**robot**,**b**)
next(A,**q**) ← my true(A,**r**),does(A,**robot**,**c**)
next(A,**r**) ← my true(A,**r**),does(A,**robot**,**a**)
next(A,**r**) ← my true(A,**r**),does(A,**robot**,**b**)
next(A,**q**) ← my true(A,**p**),does(A,**robot**,**b**)
next(A,**p**) ← my true(A,**p**),does(A,**robot**,**c**)
next(A,B) ← my true(A,C),my succ(C,B)
next(A,**p**) ← not my true(A,B),does(A,**robot**,**a**)

*next(A,B) ← head(A,****4.543****), tail(A,C), head(C,B).*
next(A,B) ← tail(A,C),next(C,B).

sumk(A) ← member(A,B), member(A,C), add(B,C,**612**)

Implementation: Bias

Predicate	Type
head_pred(f,1)	(state)
body_pred(cell,4)	(state,pos,color,type)
body_pred(distance,3)	(pos,pos,int)

Setting	Bias	Example
Arguments	cell 3 distance 3	f(State) ← cell(State,Piece1,**Color1**,Type),cell(State,Piece2,**Color2**,Type),distance(Piece1,Piece2,**Dist**)
Types	integer type	f(State) ← cell(State,Piece1,Color,**Type1**),cell(State,Piece2,Color,**Type2**),distance(Piece1,Piece2,**Dist**)
All		f(State) ← cell(**State**,**Piece1**,**Color1**,**Type1**),cell(**State**,**Piece2**,**Color2**,**Type2**),distance(**Piece1**,**Piece2**,**Dist**)