Probabilistic Logic Programming with Fusemate: Main Ideas and Recent Developments

Peter Baumgartner

CSIRO/Data61 (StatML)
ANU CECC

About

- PhD in 1996 in Germany, on Automated Reasoning
- NICTA 2005, CSIRO since 2014

Research Interest

Knowledge representation and reasoning

Designing inference systems

Applications

Recently

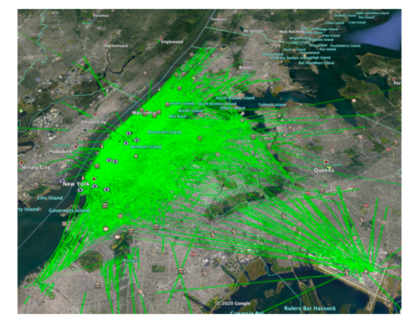
Probabilitistic Logic Programming (PLP)

Combination with LLM (with Lachlan McGinness)

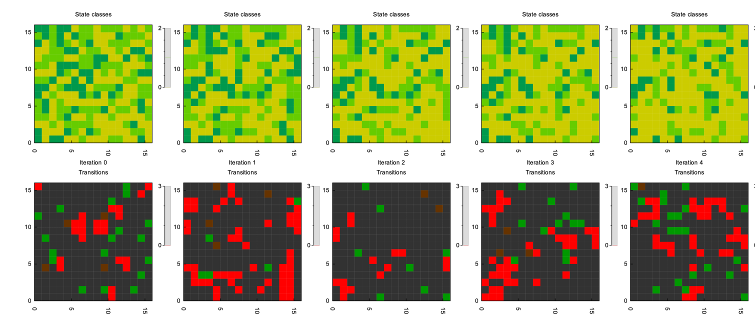
D61 Applications

Computer Factory

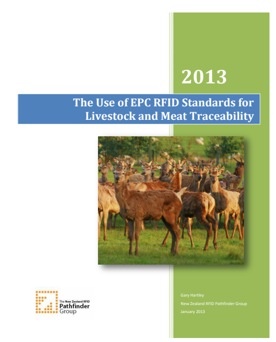
Factory floor



Taxi rides in NYC

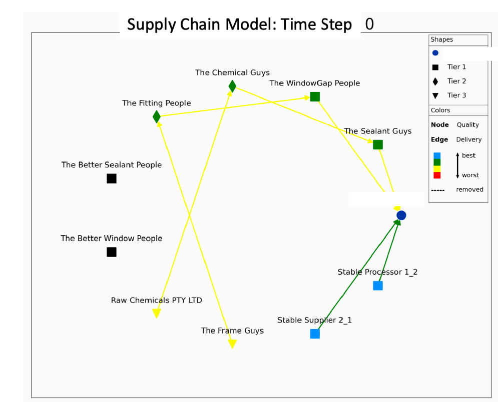


Valuing Sustainability - Future states



Food supply chain

Beef supply chain



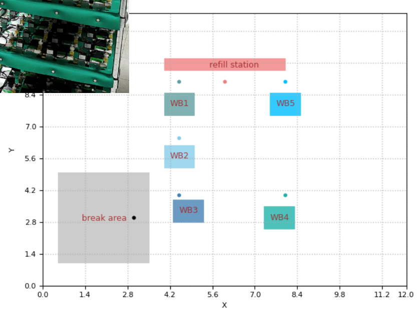
Factory supply chain

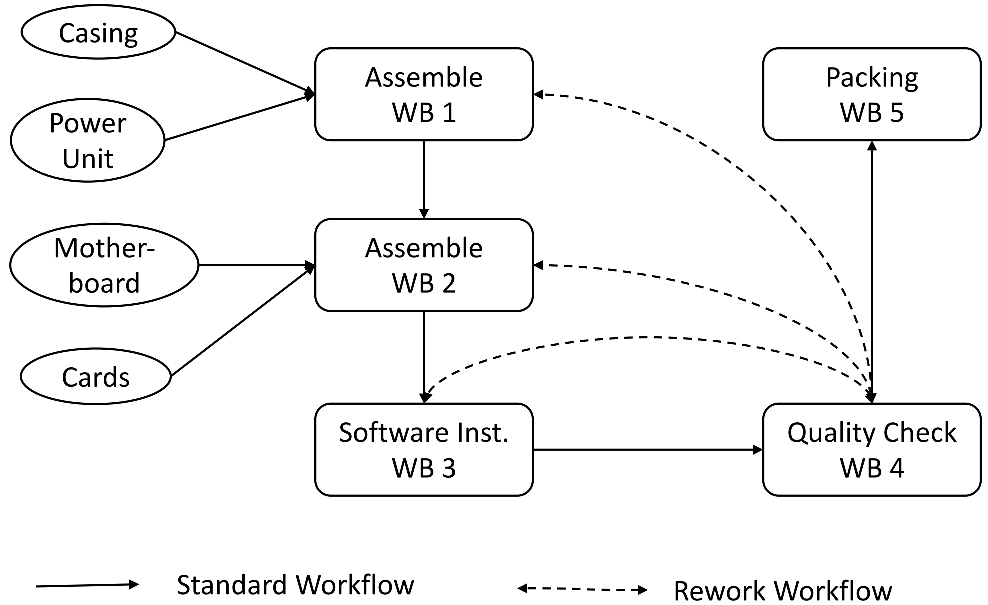
States - Transitions - Uncertainty

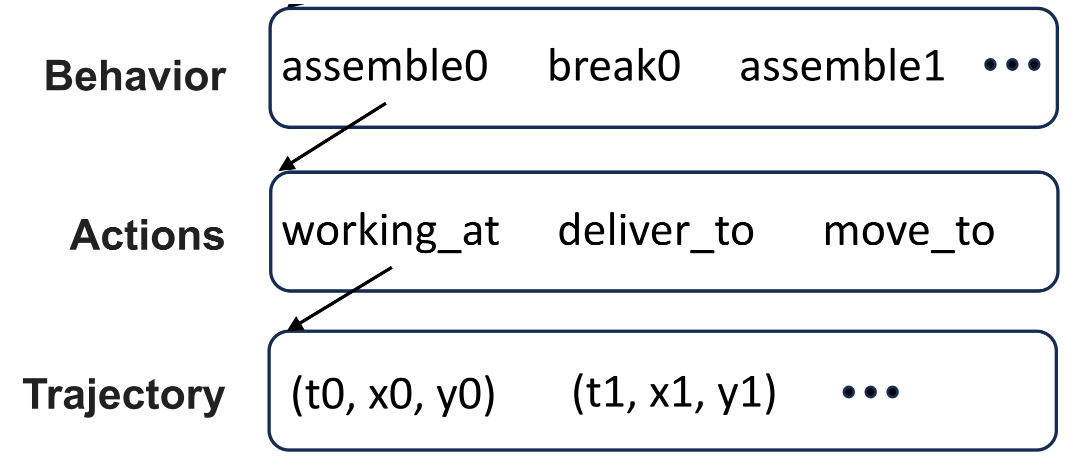
TLDR; Computer Factory Example (FDMF)

Computer

Factory



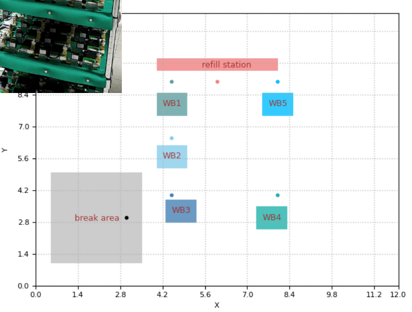




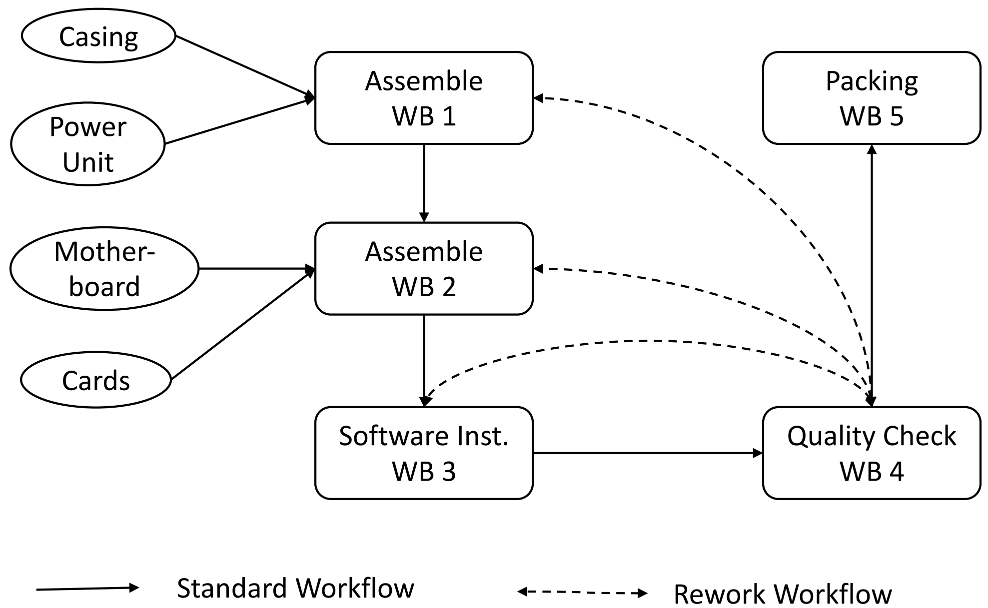
Problem: Trajectory classification: what actions/behaviours exhibited by a trajectory?

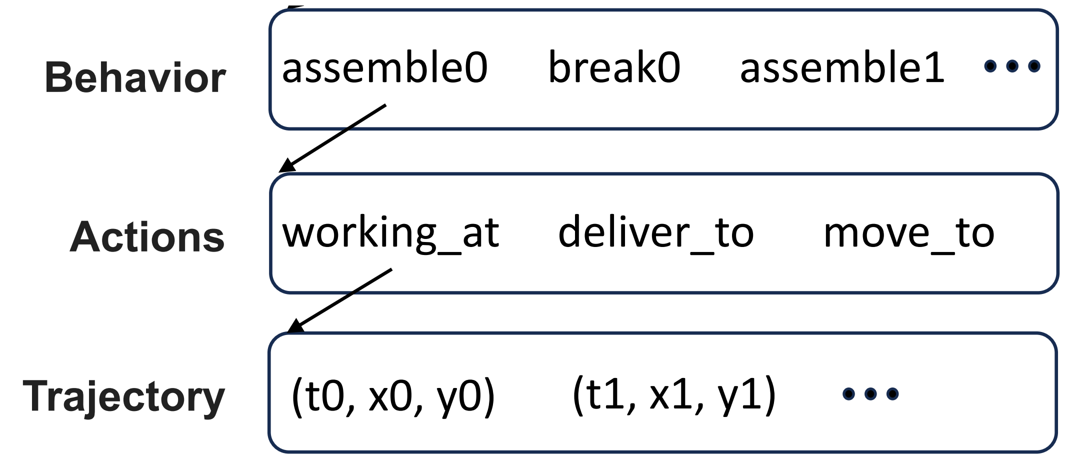
Daniel Smith et al. *Activity Recognition within a Manufacturing System: A Comparison of Logic Programming, Machine Learning, and Combinatorial Optimization Based Methods*. 2023

TLDR; Computer Factory Example (FDMF)



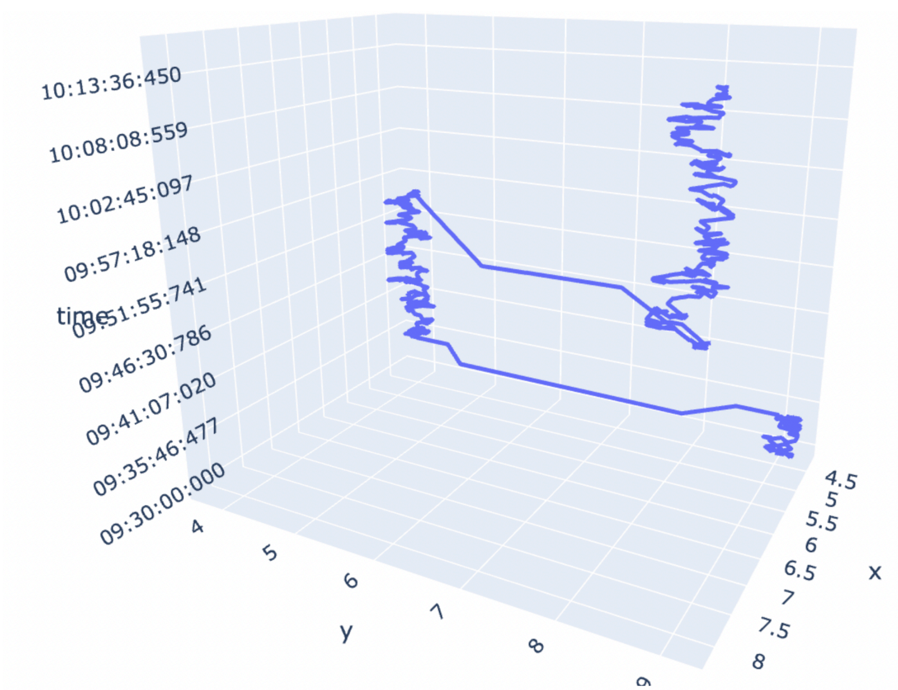
Computer Factory





Problem: Trajectory classification: what actions/behaviours exhibited by a trajectory?

Given trajectory

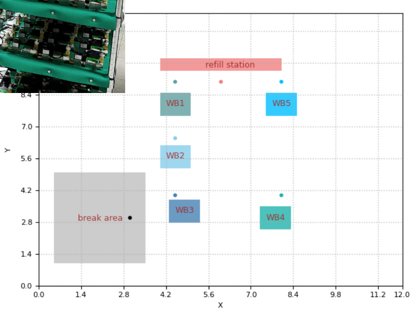


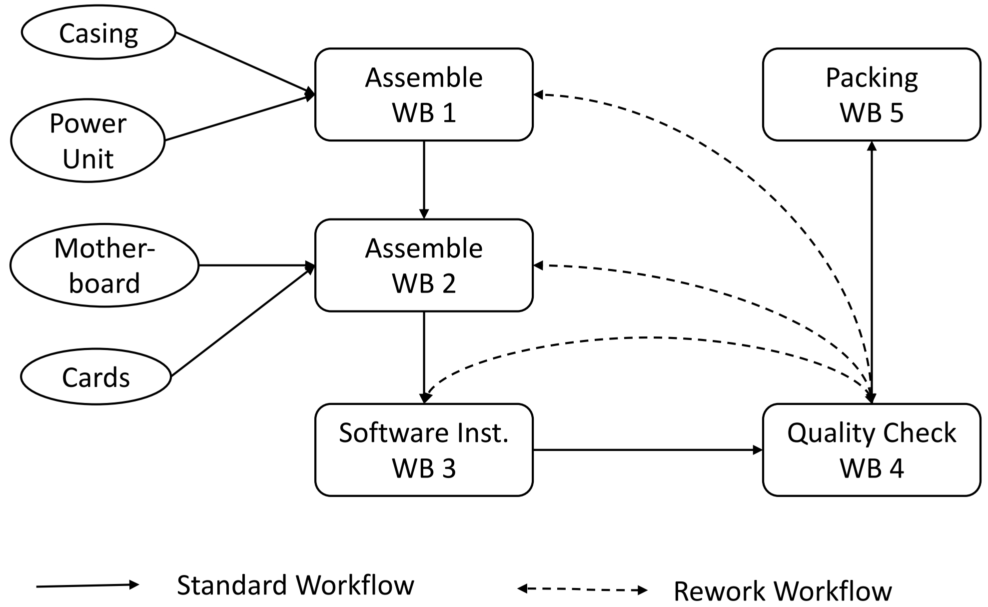
Daniel Smith et al. *Activity Recognition within a Manufacturing System: A Comparison of Logic Programming, Machine Learning, and Combinatorial Optimization Based Methods*. 2023

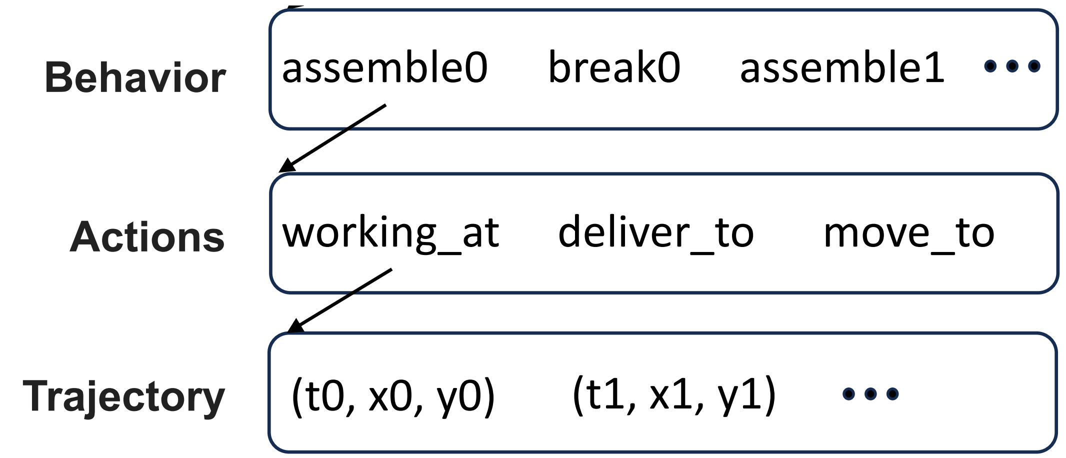
TLDR; Computer Factory Example (FDMF)

Computer

Factory

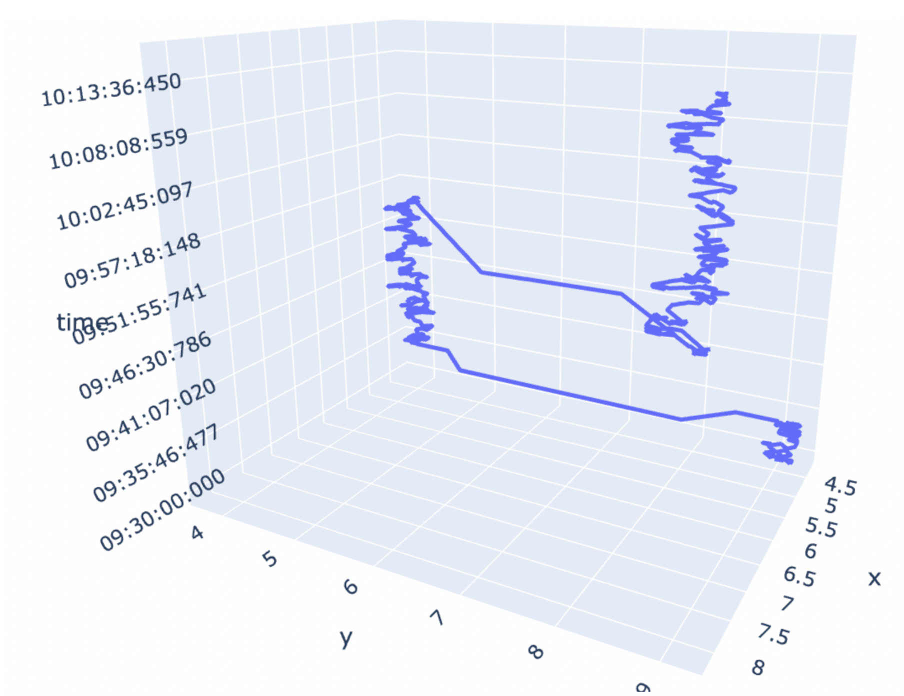






Problem: Trajectory classification: what actions/behaviours exhibited by a trajectory?

Given trajectory



Probabilistic logic program

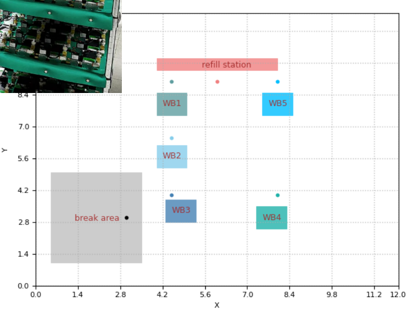
```
behaviour ~ [assemble, break ...]. %% Distribution
worker ~ [1,2,3,4,5].              %% Distribution

action = working_at(wb(W))   @ 0 :-
    behaviour = assemble,
    worker = W.

action = deliver_to(wb(W+1)) @ 1 :-
    behaviour = assemble,
    worker = W.

loc = L       @ T :- action = working_at(L) @ T.
dur ~ [1..10] @ T :- action = working_at(_) @ T.
```

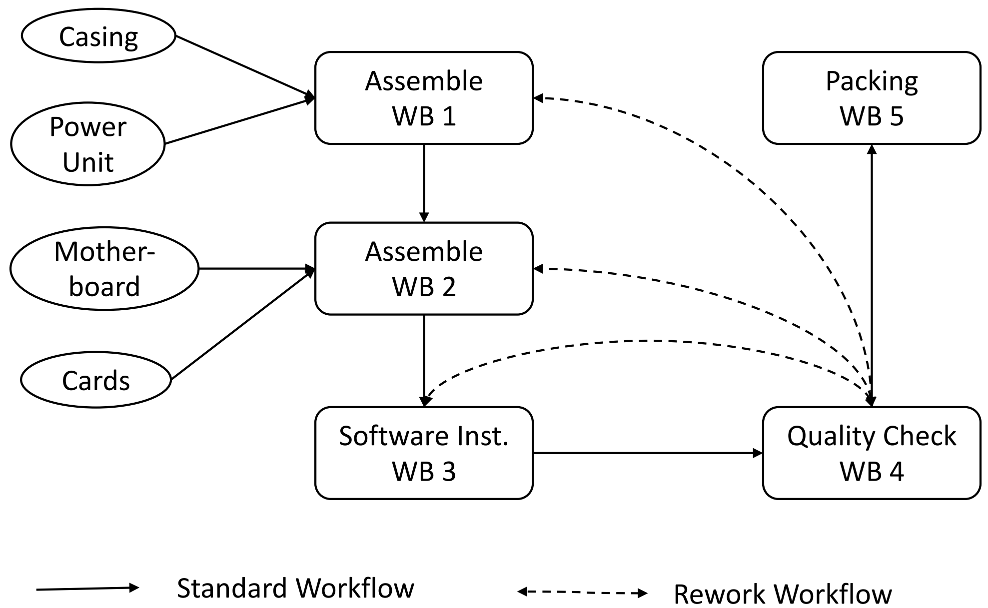

Daniel Smith et al. *Activity Recognition within a Manufacturing System: A Comparison of Logic Programming, Machine Learning, and Combinatorial Optimization Based Methods*. 2023

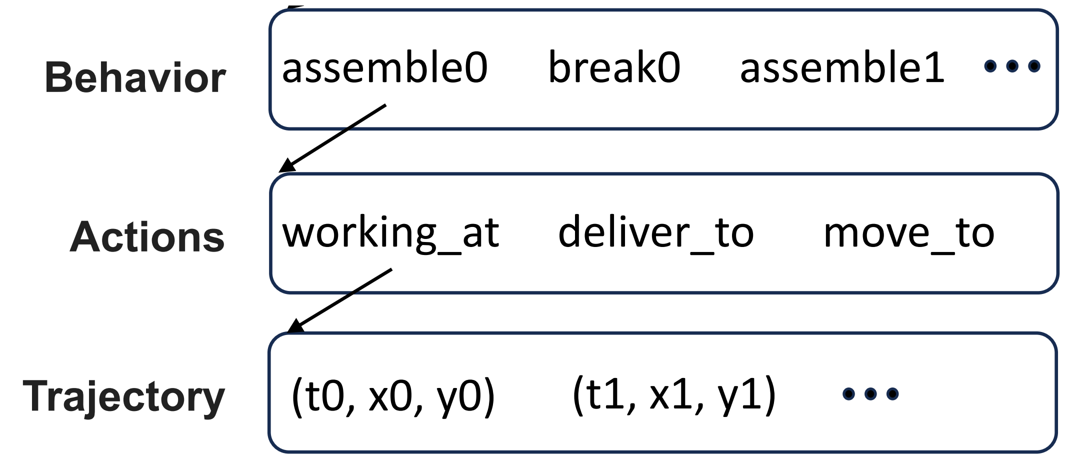
TLDR; Computer Factory Example (FDMF)

Computer

Factory



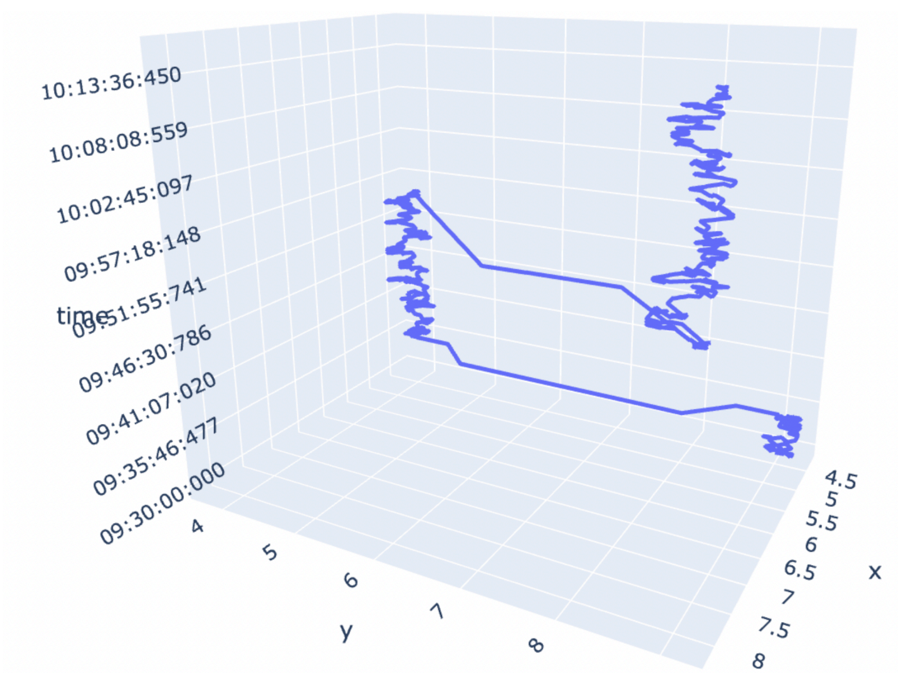




Problem: Trajectory classification: what actions/behaviours exhibited by a trajectory?

Given trajectory → **Probabilistic logic program** → **Most likely behaviour seq.**

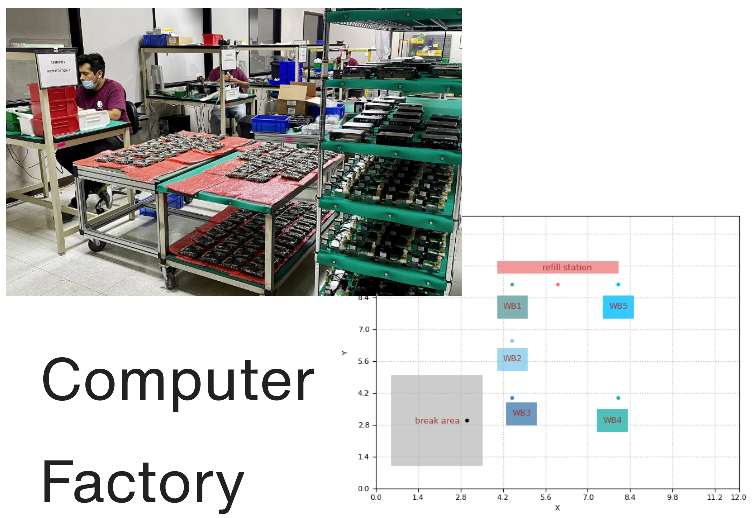
assemble -> break -> ...




```
behaviour ~ [assemble, break ...]. %% Distribution
worker ~ [1,2,3,4,5].              %% Distribution

action = working_at(wb(W))   @ 0 :-
    behaviour = assemble,
    worker = W.

action = deliver_to(wb(W+1)) @ 1 :-
    behaviour = assemble,
    worker = W.

loc = L       @ T :- action = working_at(L) @ T.
dur ~ [1..10] @ T :- action = working_at(_) @ T.
```

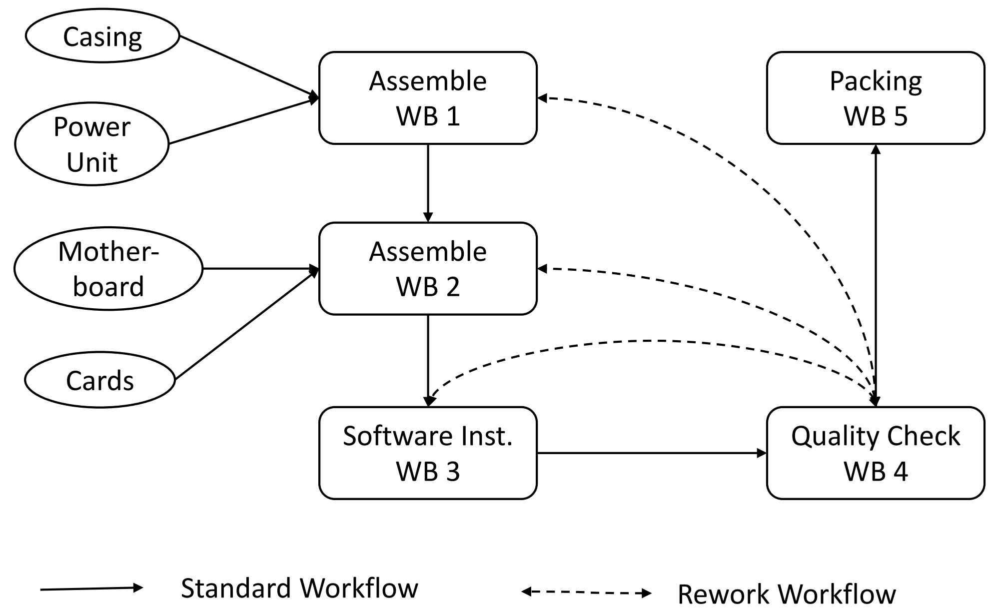

Daniel Smith et al. *Activity Recognition within a Manufacturing System: A Comparison of Logic Programming, Machine Learning, and Combinatorial Optimization Based Methods*. 2023

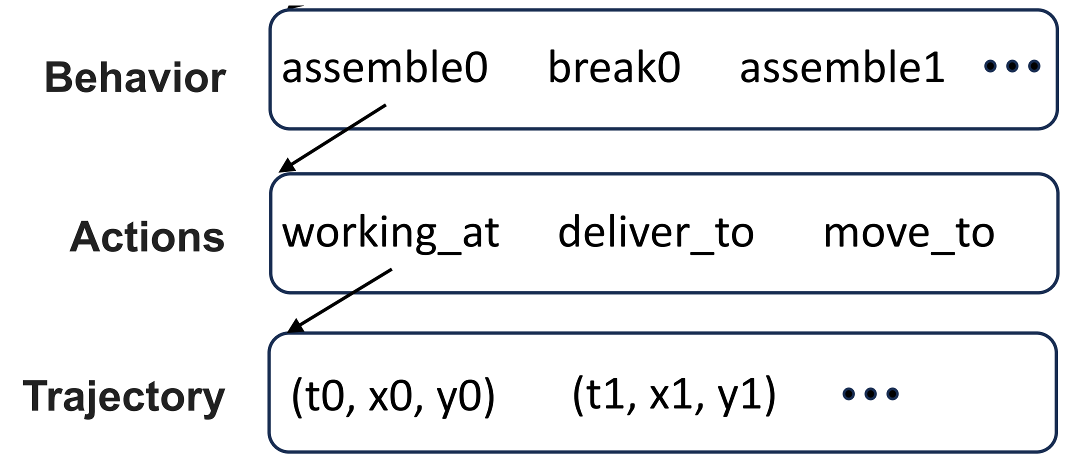
TLDR; Computer Factory Example (FDMF)



Computer

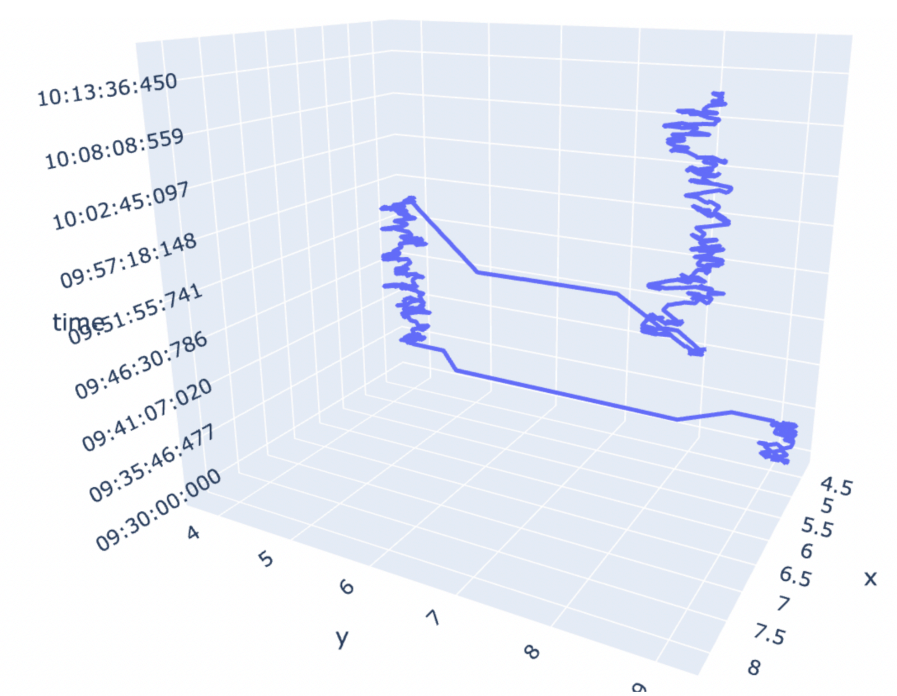
Factory





Problem: Trajectory classification: what actions/behaviours exhibited by a trajectory?

Given trajectory → **Probabilistic logic program** → **Most likely behaviour seq.**

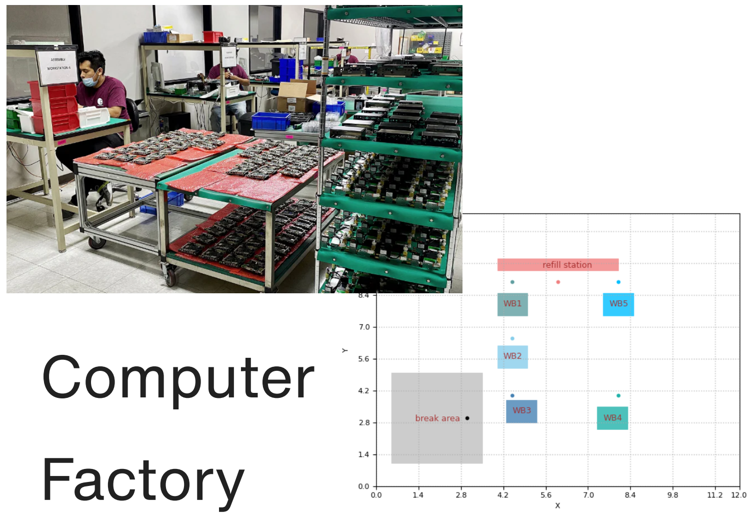



```
behaviour ~ [assemble, break ...].  %% Distribution
worker ~ [1,2,3,4,5].               %% Distribution

action = working_at(wb(W))   @ 0 :-
    behaviour = assemble,
    worker = W.

action = deliver_to(wb(W+1)) @ 1 :-
    behaviour = assemble,
    worker = W.

loc = L       @ T :- action = working_at(L) @ T.
dur ~ [1..10] @ T :- action = working_at(_) @ T.
```

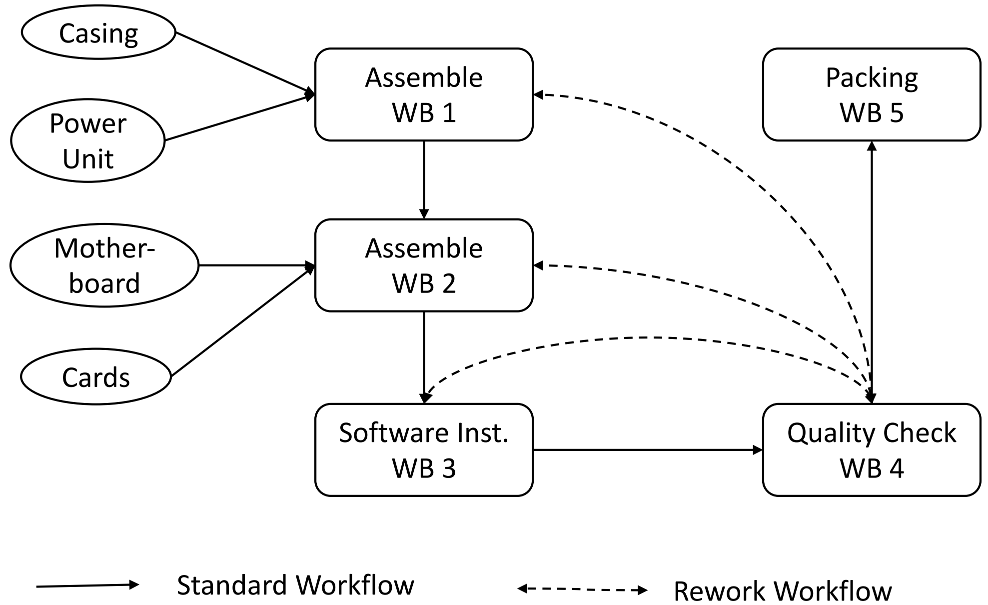
assemble -> break -> ...

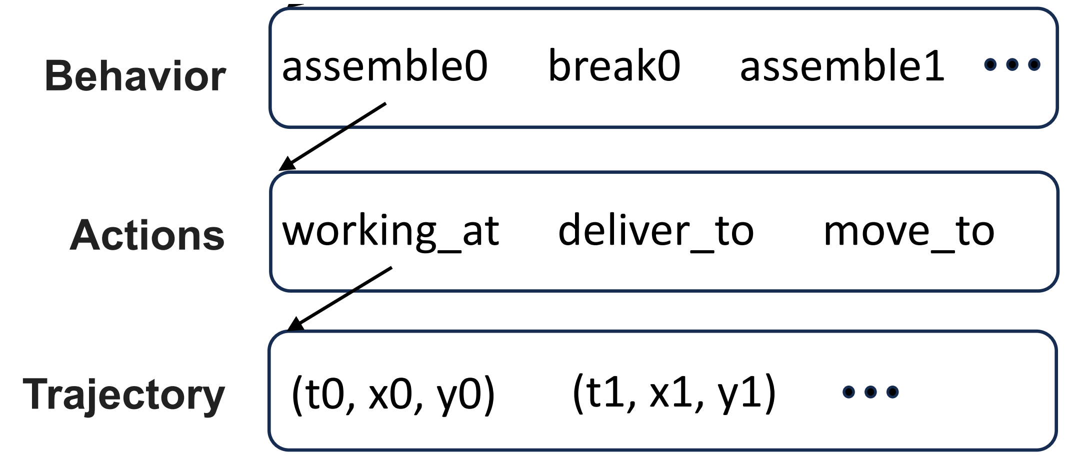
Daniel Smith et al. *Activity Recognition within a Manufacturing System: A Comparison of Logic Programming, Machine Learning, and Combinatorial Optimization Based Methods.* 2023

TLDR; Computer Factory Example (FDMF)



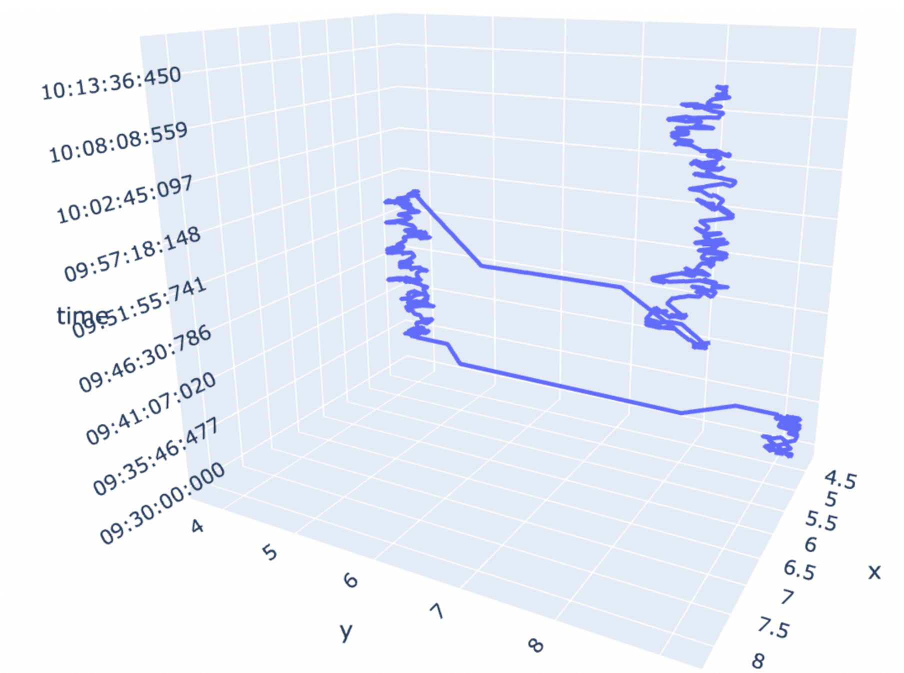
Computer Factory





Problem: Trajectory classification: what actions/behaviours exhibited by a trajectory?

Given trajectory



Probabilistic logic program

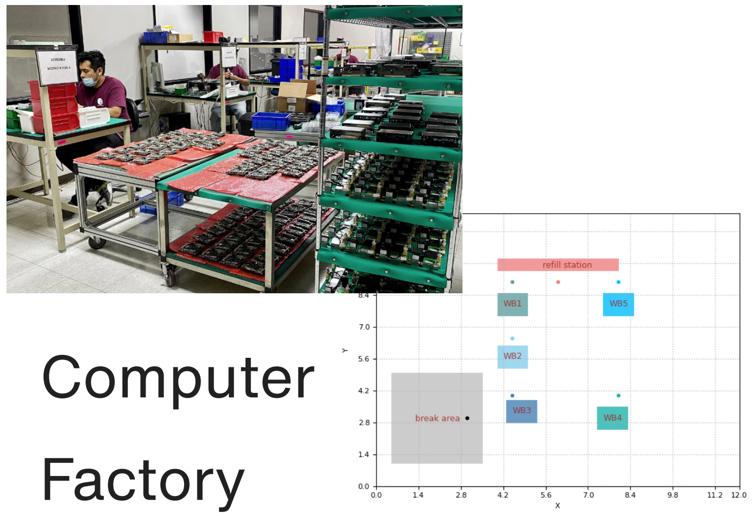
```
behaviour ~ [assemble, break ...]. %% Distribution
worker ~ [1,2,3,4,5].              %% Distribution

action = working_at(wb(W))  @ 0 :-
    behaviour = assemble,
    worker = W.

action = deliver_to(wb(W+1)) @ 1 :-
    behaviour = assemble,
    worker = W.

loc = L       @ T :- action = working_at(L) @ T.
dur ~ [1..10] @ T :- action = working_at(_) @ T.
```

Most likely behaviour seq.

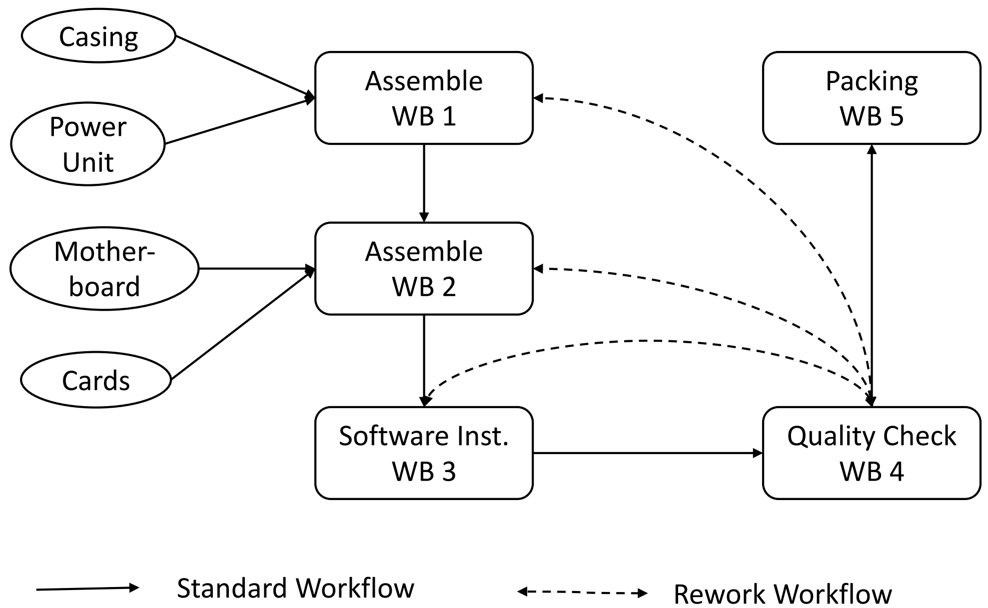
assemble -> break -> ...

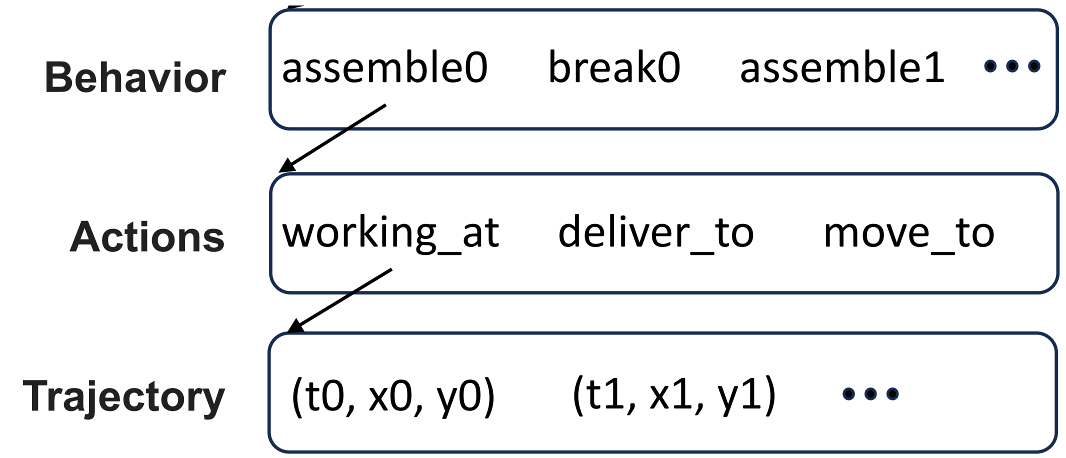
Daniel Smith et al. *Activity Recognition within a Manufacturing System: A Comparison of Logic Programming, Machine Learning, and Combinatorial Optimization Based Methods*. 2023

TLDR; Computer Factory Example (FDMF)



Computer Factory

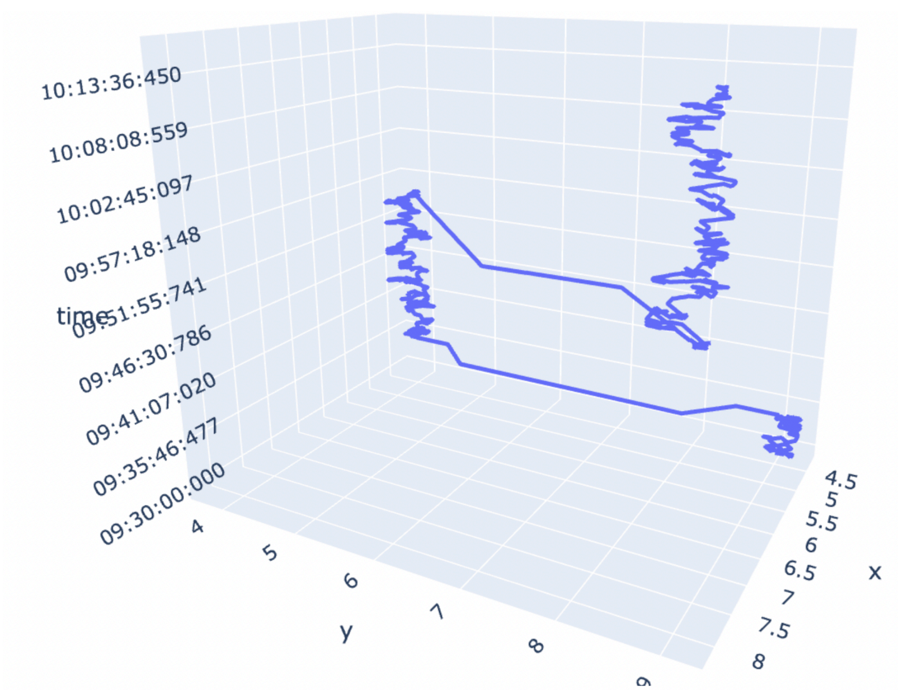




Problem: Trajectory classification: what actions/behaviours exhibited by a trajectory?

Given trajectory → **Probabilistic logic program** → **Most likely behaviour seq.**

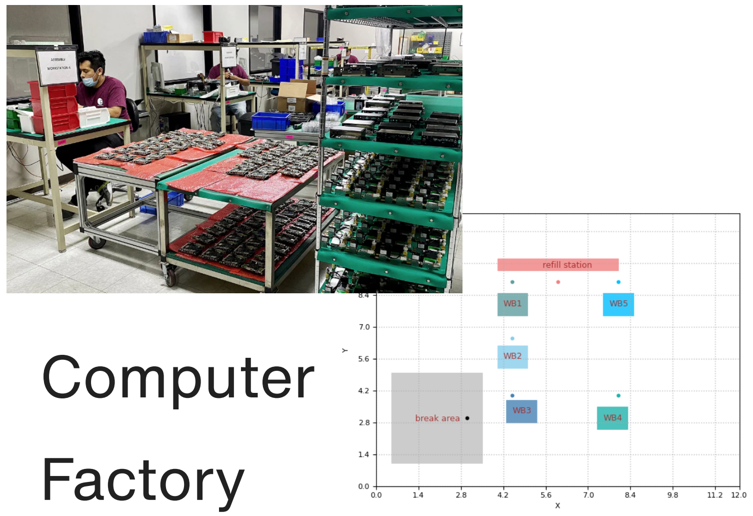
assemble -> break -> ...



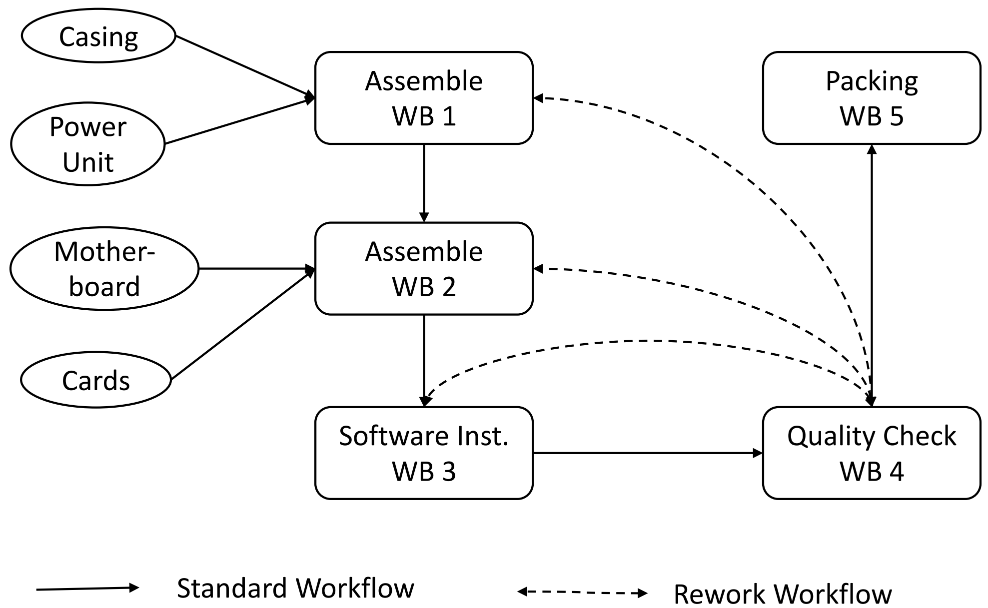

```
behaviour ~ [assemble, break ...].  %% Distribution
worker ~ [1,2,3,4,5].               %% Distribution

action = working_at(wb(W))  @ 0 :-
    behaviour = assemble,
    worker = W.

action = deliver_to(wb(W+1)) @ 1 :-
    behaviour = assemble,
    worker = W.

loc = L      @ T :- action = working_at(L) @ T.
dur ~ [1..10] @ T :- action = working_at(_) @ T.
```


Daniel Smith et al. *Activity Recognition within a Manufacturing System: A Comparison of Logic Programming, Machine Learning, and Combinatorial Optimization Based Methods.* 2023

TLDR; Computer Factory Example (FDMF)



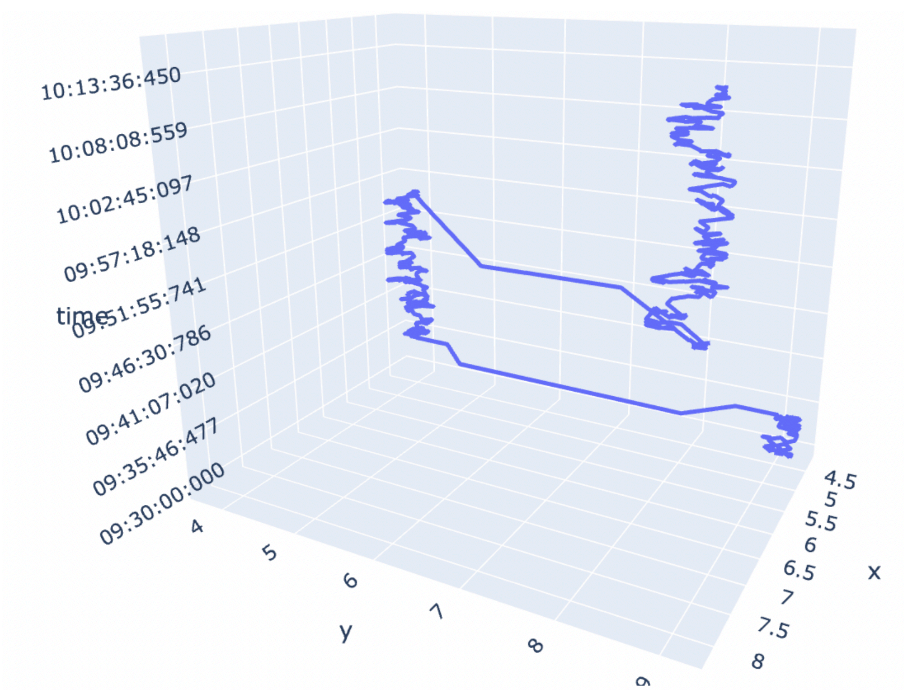
Computer Factory



Behavior	assemble0 break0 assemble1 •••
Actions	working_at deliver_to move_to
Trajectory	(t0, x0, y0) (t1, x1, y1) •••

Problem: Trajectory classification: what actions/behaviours exhibited by a trajectory?

Given trajectory → **Probabilistic logic program** → **Most likely behaviour seq.**



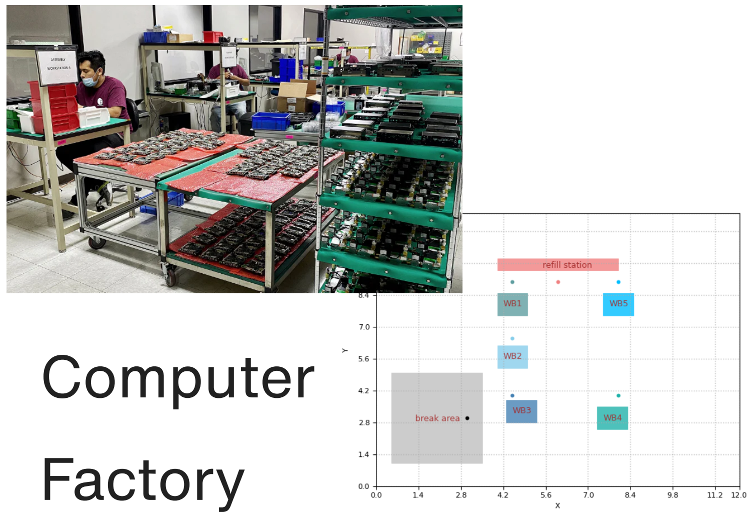

```
behaviour ~ [assemble, break ...].  %% Distribution
worker ~ [1,2,3,4,5].               %% Distribution

action = working_at(wb(W))   @ 0 :-
    behaviour = assemble,
    worker = W.

action = deliver_to(wb(W+1)) @ 1 :-
    behaviour = assemble,
    worker = W.

loc = L        @ T :- action = working_at(L) @ T.
dur ~ [1..10]  @ T :- action = working_at(_) @ T.
```

assemble -> break -> ...

Hidden Markov Model

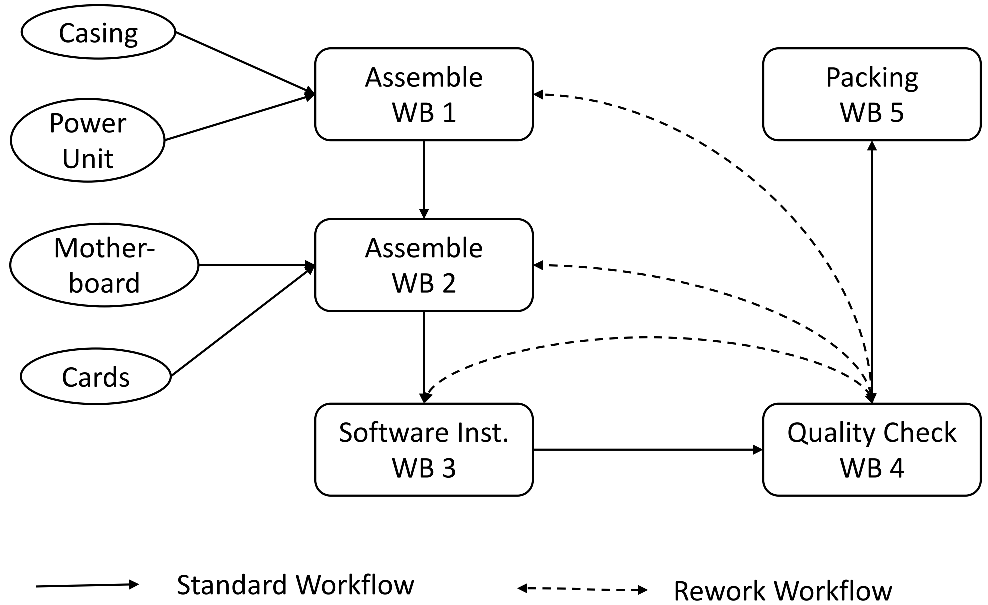
Daniel Smith et al. *Activity Recognition within a Manufacturing System: A Comparison of Logic Programming, Machine Learning, and Combinatorial Optimization Based Methods*. 2023

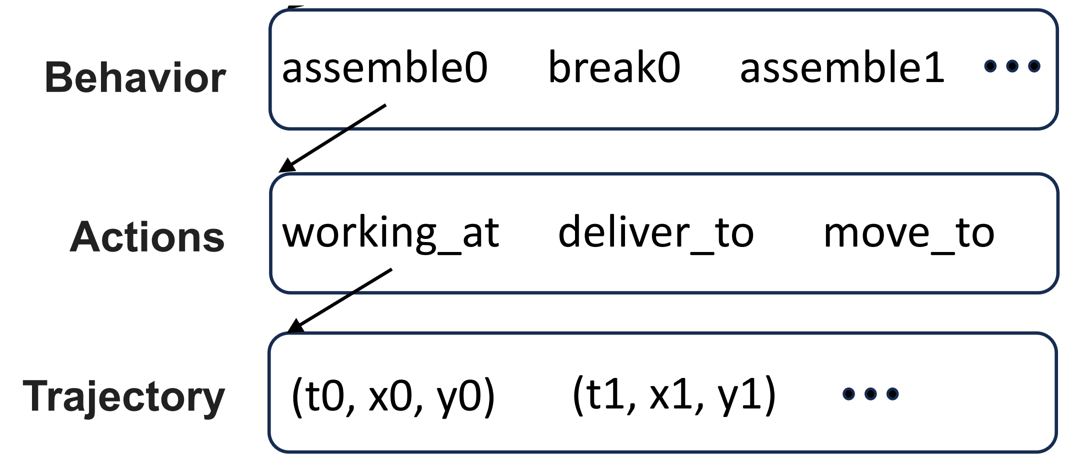
TLDR; Computer Factory Example (FDMF)



Computer

Factory





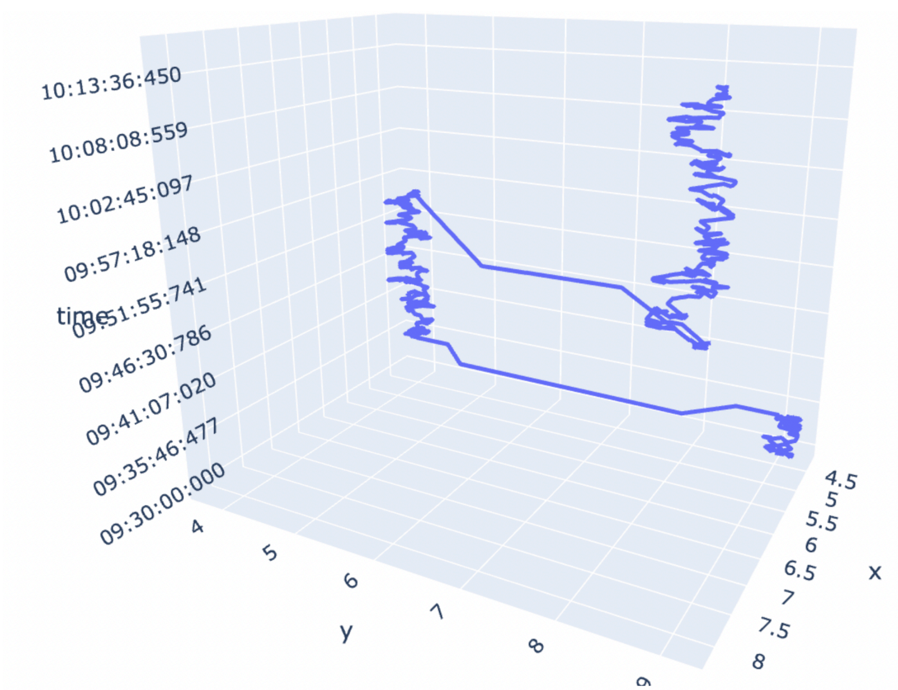
Problem: Trajectory classification: what actions/behaviours exhibited by a trajectory?

Given trajectory

Probabilistic logic program

Most likely behaviour seq.

assemble -> break -> ...




```
behaviour ~ [assemble, break ...].  %% Distribution
worker ~ [1,2,3,4,5].               %% Distribution

action = working_at(wb(W))  @ 0 :-
    behaviour = assemble,
    worker = W.

action = deliver_to(wb(W+1)) @ 1 :-
    behaviour = assemble,
    worker = W.

loc = L       @ T :- action = working_at(L) @ T.
dur ~ [1..10] @ T :- action = working_at(_) @ T.
```

Hidden Markov Model

PLP can do much more!

Daniel Smith et al. *Activity Recognition within a Manufacturing System: A Comparison of Logic Programming, Machine Learning, and Combinatorial Optimization Based Methods.* 2023

Part 1

- Probabilistic
- Logic
- Programming
- Fusemate Implementation

Part 2

- LLMs + Logic (Programming)
- Neural Networks + Logic (Programming

Logic

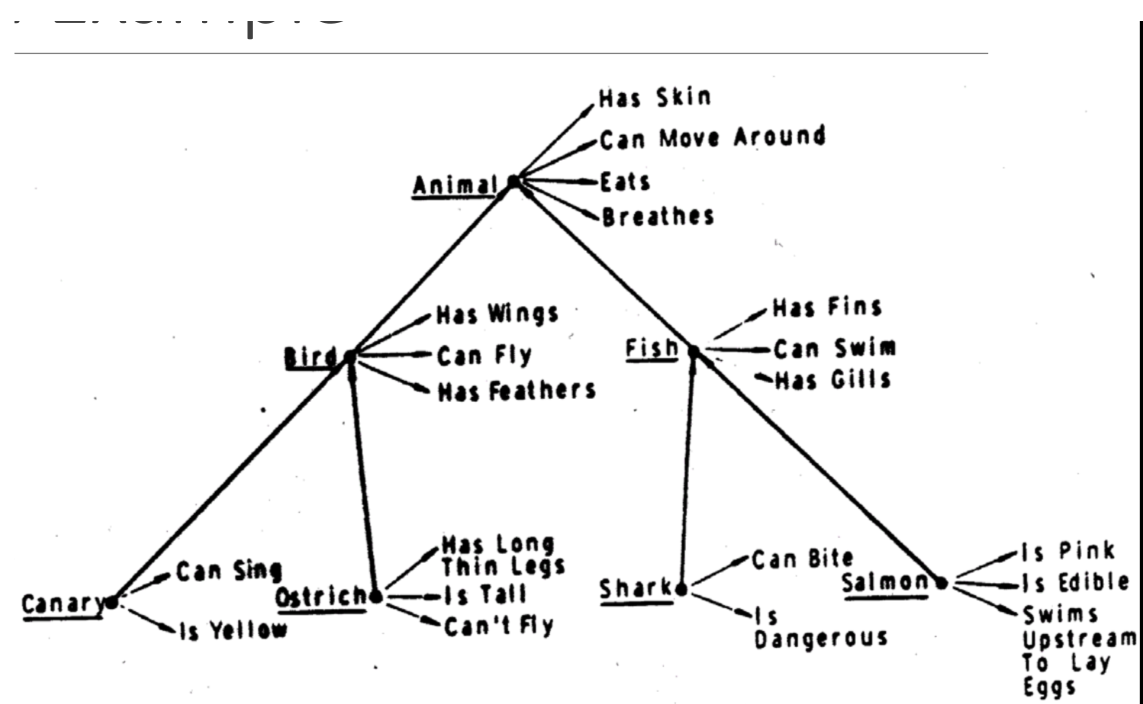
"Algorithm = Logic + Control"

- Model the problem at hand with "logic"
- Feed into automated reasoning system
- Push button and get solution

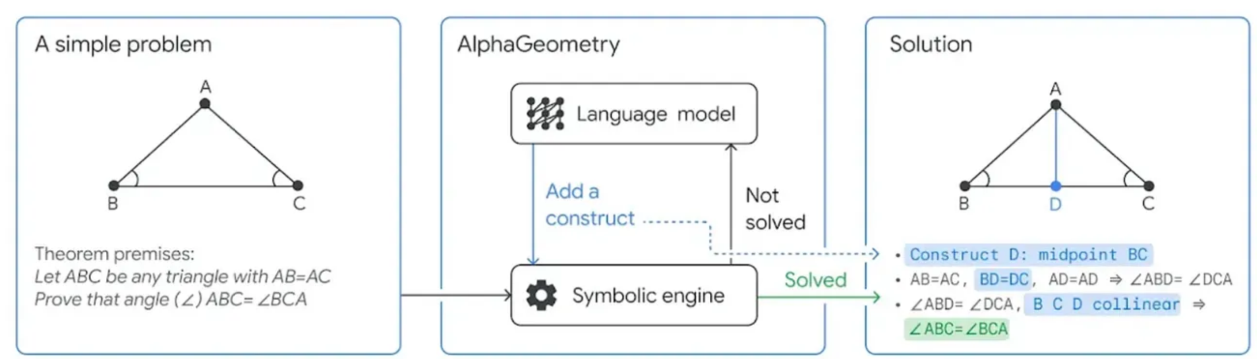
Logic	**Reasoning Tasks**
Classical	Proving
Non-monotonic	Disproving
Modal	Query answering
Probabilistic	Model computation
Temporal	Knowledge Completion
Graphs (Ontologies)	Diagnosis
Relational (Tables)	
Built-in Theories	

flight(toronto, london).
flight(london, rome).
flight(chicago, london).
flight(X, Y) :– flight(X, Z) , flight(Z, Y).

Relational



Ontology



AlphaGeometry, AlphaProof, LLM-modulo, ...

Neuro-Symbolic

Logic

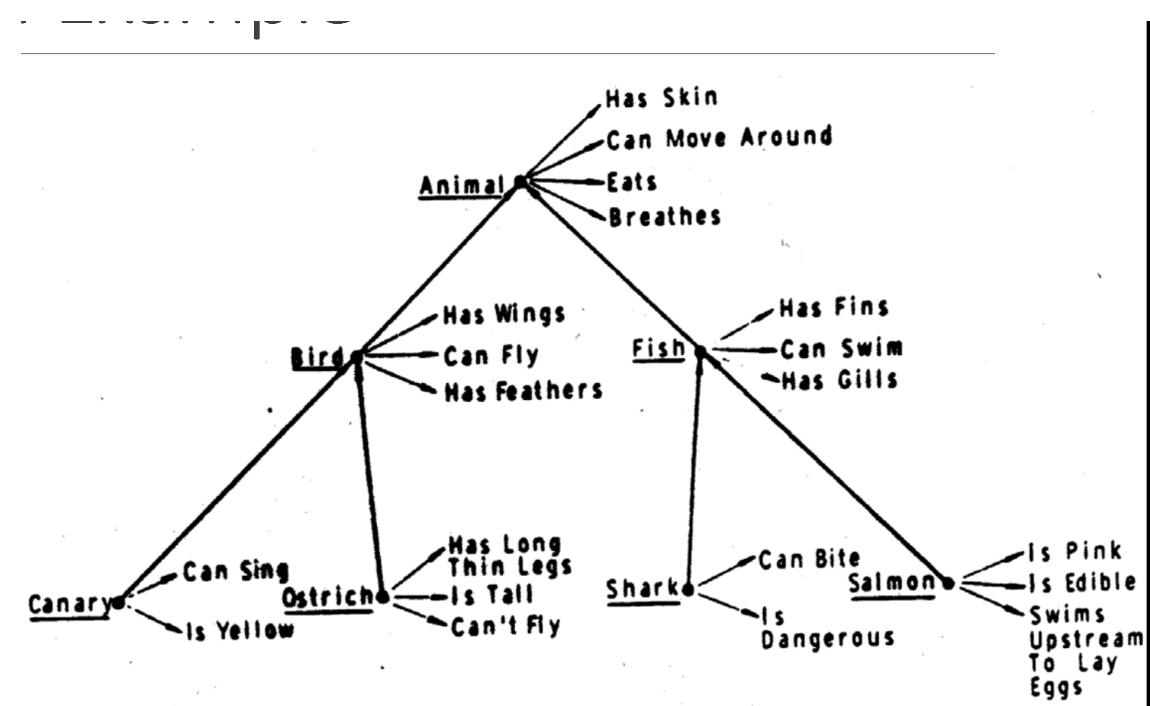
“Algorithm = Logic + Control”

- Model the problem at hand with “logic”
- Feed into automated reasoning system
- Push button and get solution

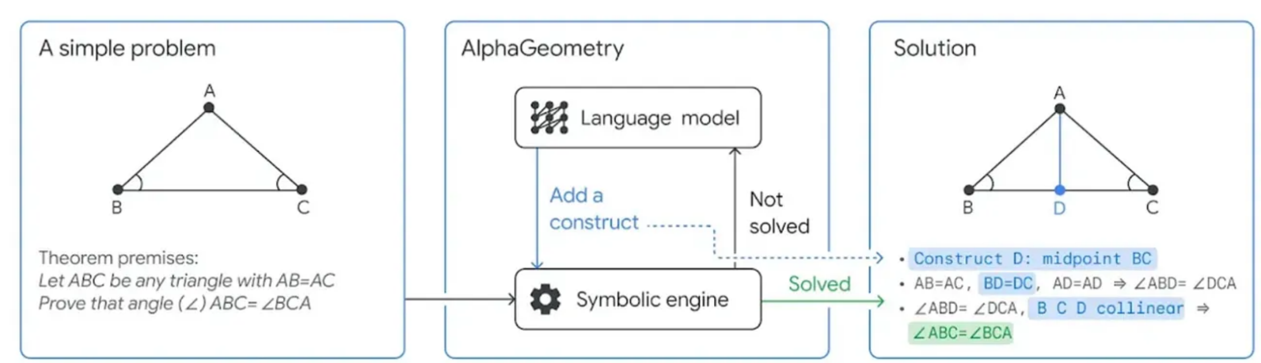
Logic	Reasoning Tasks
Classical	Proving
Non-monotonic	Disproving
Modal	Query answering
Probabilistic	Model computation
Temporal	Knowledge Completion
Graphs (Ontologies)	Diagnosis
Relational (Tables)	
Built-in Theories	

flight(toronto, london).
flight(london, rome).
flight(chicago, london).
flight(X, Y) :– flight(X, Z) , flight(Z, Y).

Relational



Ontology



AlphaGeometry, AlphaProof, LLM-modulo, ...

Neuro-Symbolic

Logic

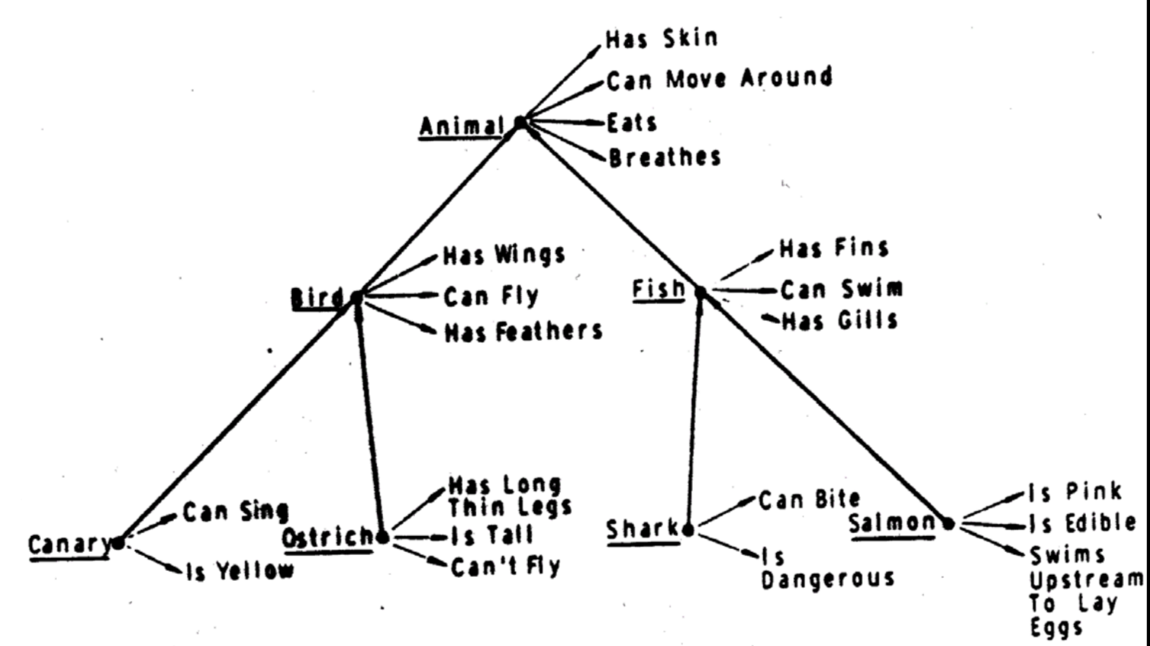
"Algorithm = Logic + Control"

- Model the problem at hand with "logic"
- Feed into automated reasoning system
- Push button and get solution

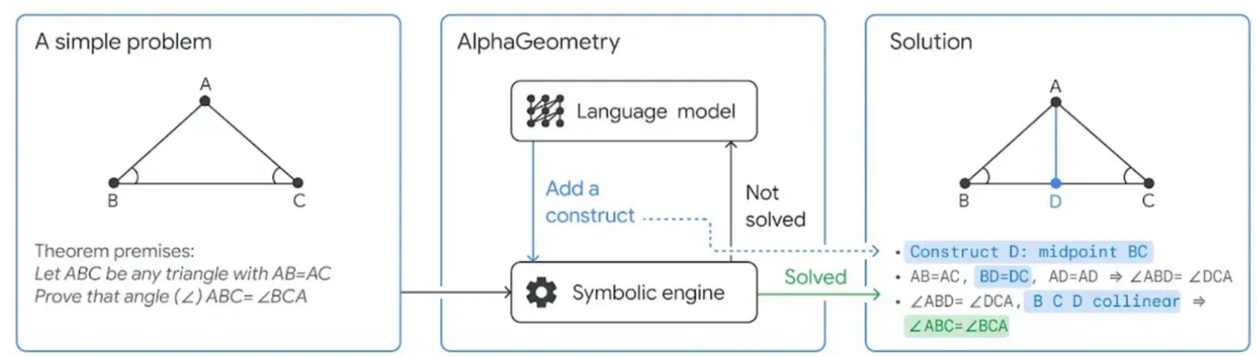
Logic	Reasoning Tasks
Classical	Proving
Non-monotonic	Disproving
Modal	Query answering
Probabilistic	Model computation
Temporal	Knowledge Completion
Graphs (Ontologies)	Diagnosis
Relational (Tables)	
Built-in Theories	

flight(toronto, london).
flight(london, rome).
flight(chicago, london).
flight(X, Y) :– flight(X, Z) , flight(Z, Y).

Relational



Ontology



AlphaGeometry, AlphaProof, LLM-modulo, ...

Neuro-Symbolic

Logic

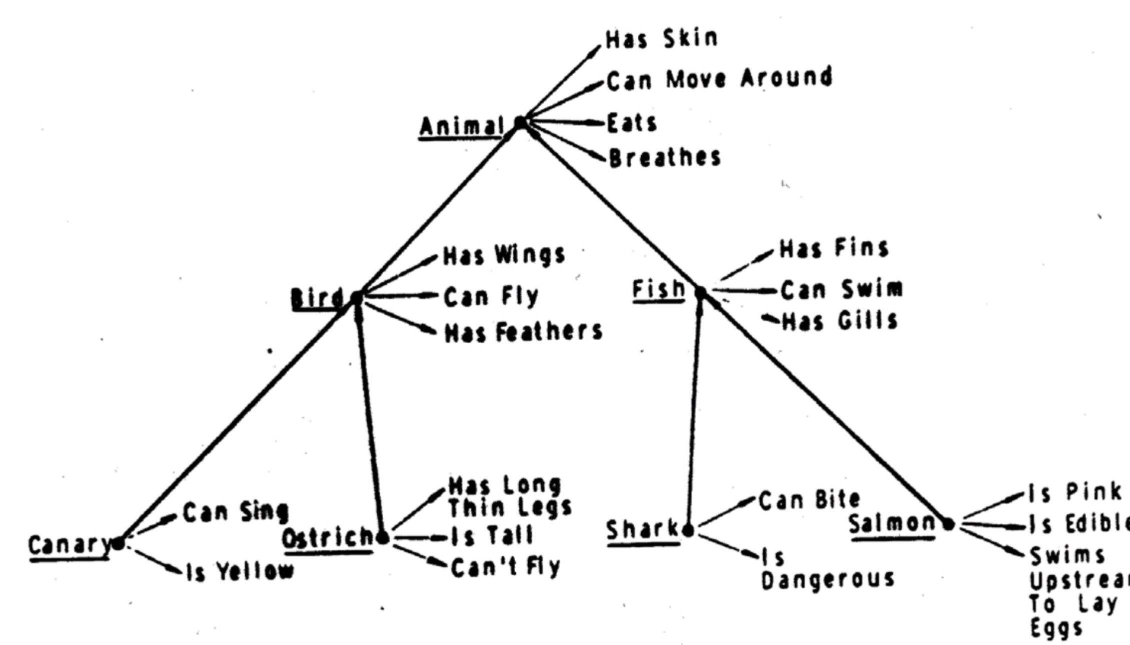
"Algorithm = Logic + Control"

- Model the problem at hand with "logic"
- Feed into automated reasoning system
- Push button and get solution 😂

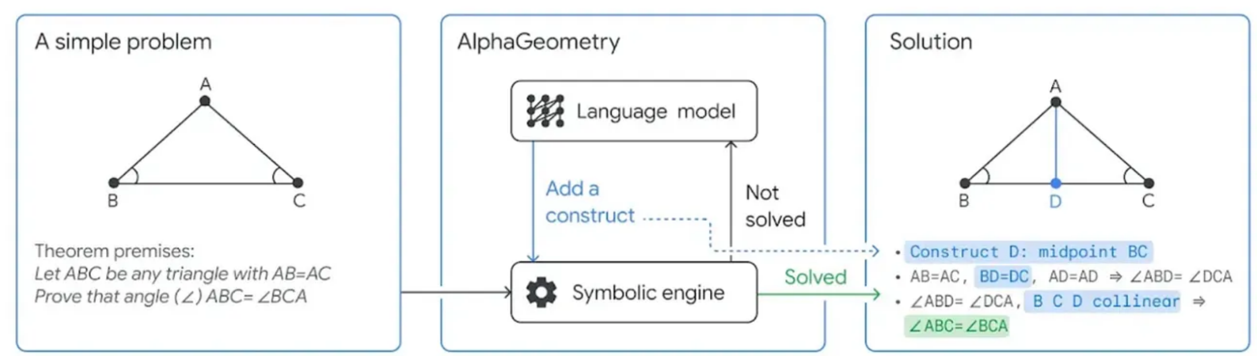
Logic	Reasoning Tasks
Classical	Proving
Non-monotonic	Disproving
Modal	Query answering
Probabilistic	Model computation
Temporal	Knowledge Completion
Graphs (Ontologies)	Diagnosis
Relational (Tables)	
Built-in Theories	

```
flight(toronto, london).
flight(london, rome).
flight(chicago, london).
flight(X, Y) :– flight(X, Z) , flight(Z, Y).
```

Relational



Ontology



AlphaGeometry, AlphaProof, LLM-modulo, ...

Neuro-Symbolic

Logic

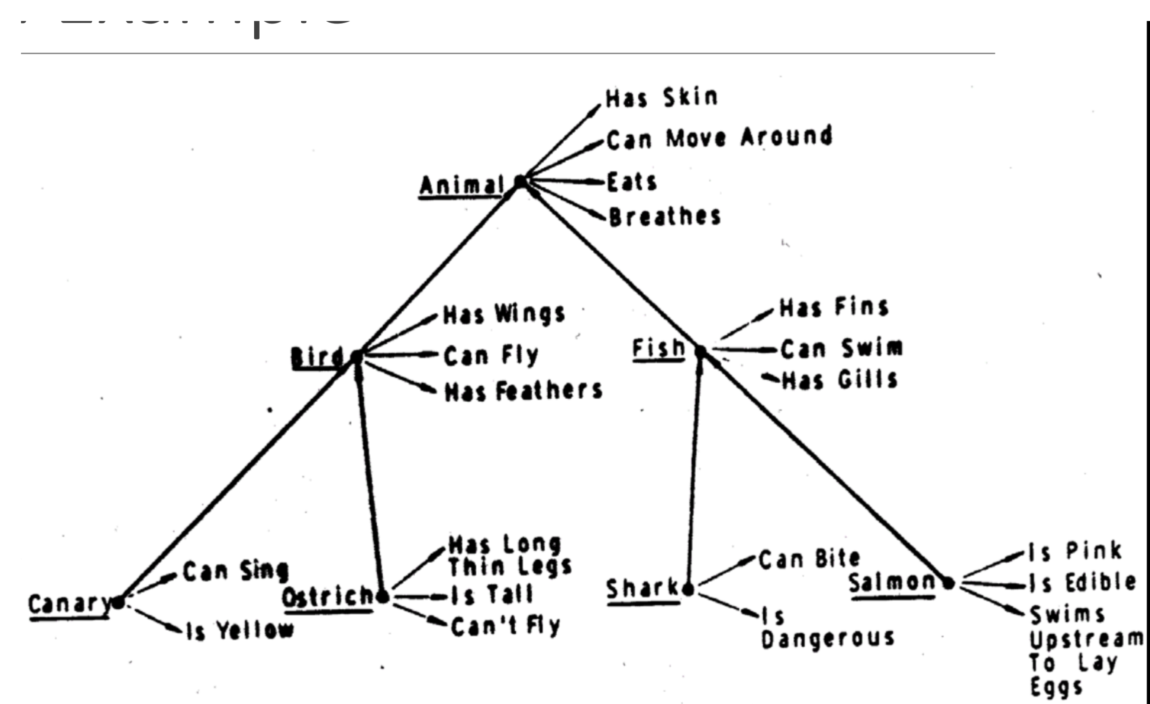
"Algorithm = Logic + Control"

- Model the problem at hand with "logic"
- Feed into automated reasoning system
- Push button and get solution

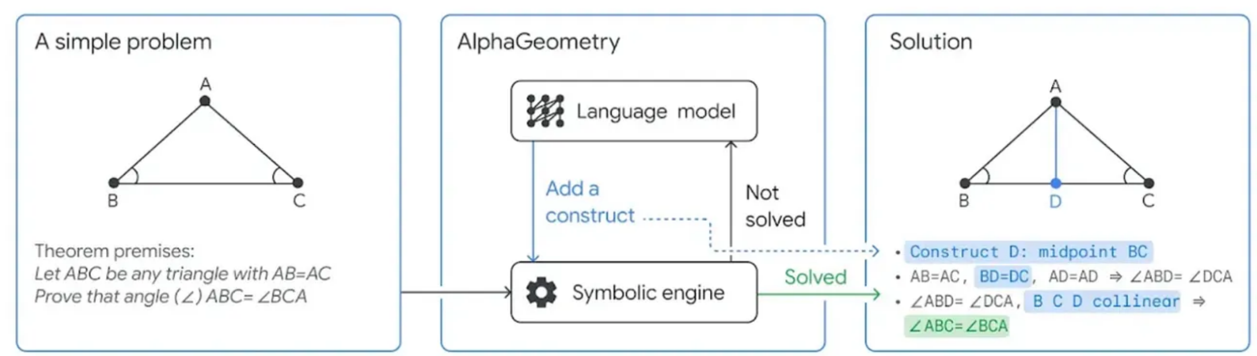
Logic	**Reasoning Tasks**
Classical	Proving
Non-monotonic	Disproving
Modal	Query answering
Probabilistic	Model computation
Temporal	Knowledge Completion
Graphs (Ontologies)	Diagnosis
Relational (Tables)	
Built-in Theories	

```
flight(toronto, london).
flight(london, rome).
flight(chicago, london).
flight(X, Y) :– flight(X, Z) , flight(Z, Y).
```

Relational



Ontology



AlphaGeometry, AlphaProof, LLM-modulo, ...

Neuro-Symbolic

Logic

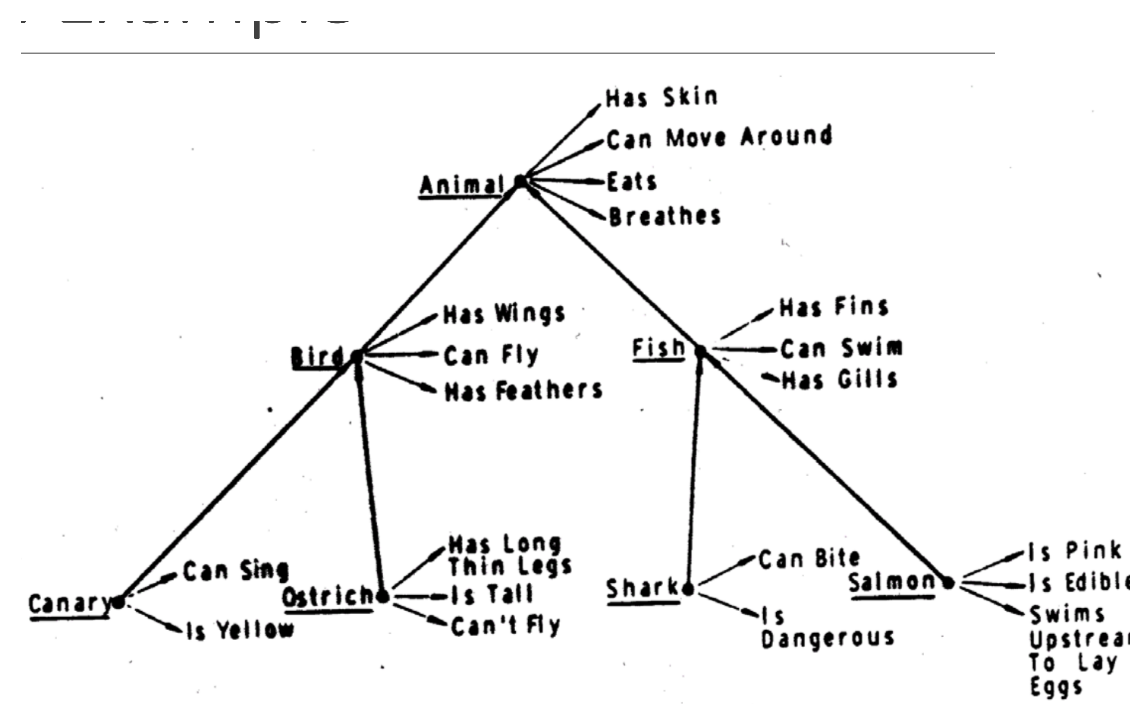
“Algorithm = Logic + Control”

- Model the problem at hand with “logic”
- Feed into automated reasoning system
- Push button and get solution

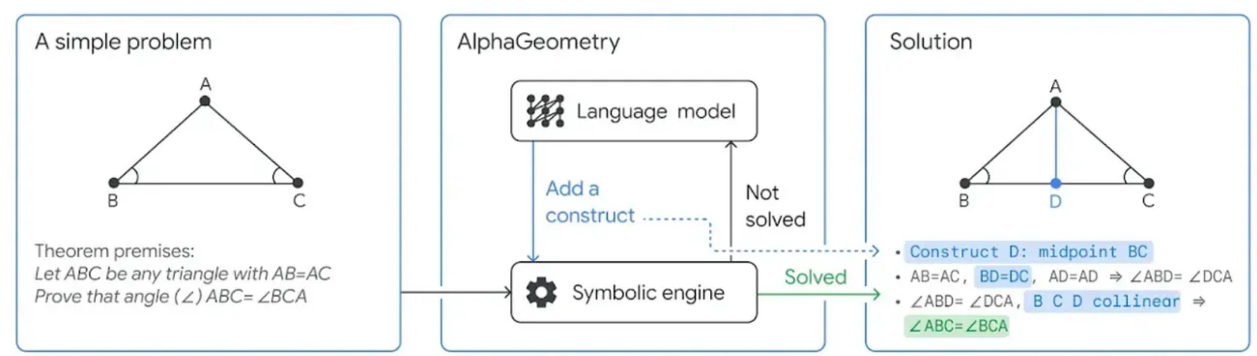
Logic	Reasoning Tasks
Classical	Proving
Non-monotonic	Disproving
Modal	Query answering
Probabilistic	Model computation
Temporal	Knowledge Completion
Graphs (Ontologies)	Diagnosis
Relational (Tables)	
Built-in Theories	

flight(toronto, london).
flight(london, rome).
flight(chicago, london).
$\text{flight}(X, Y) \mathrel{:-} \text{flight}(X, Z)\ , \text{flight}(Z, Y).$

Relational



Ontology



AlphaGeometry, AlphaProof, LLM-modulo, ...

Neuro-Symbolic

Logic

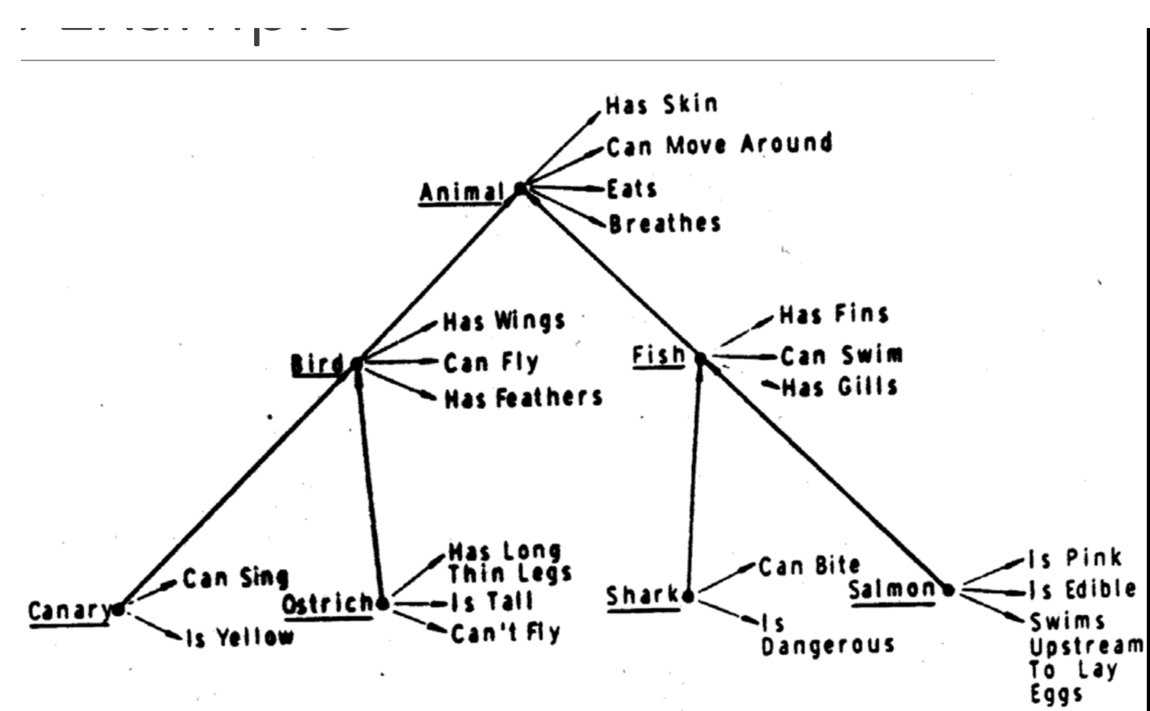
"Algorithm = Logic + Control"

- Model the problem at hand with "logic"
- Feed into automated reasoning system
- Push button and get solution 😂

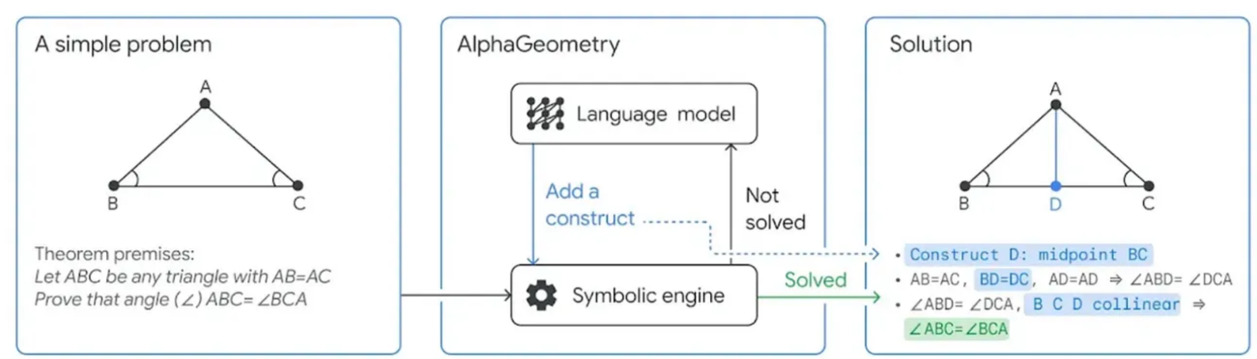
Logic	Reasoning Tasks
Classical	Proving
Non-monotonic	Disproving
Modal	Query answering
Probabilistic	Model computation
Temporal	Knowledge Completion
Graphs (Ontologies)	Diagnosis
Relational (Tables)	
Built-in Theories	

flight(toronto, london).
flight(london, rome).
flight(chicago, london).
$\mathrm{flight}(X, Y) :\!-\ \mathrm{flight}(X, Z)\ ,\ \mathrm{flight}(Z, Y).$

Relational



Ontology



AlphaGeometry, AlphaProof, LLM-modulo, ...

Neuro-Symbolic

Logic

"Algorithm = Logic + Control"

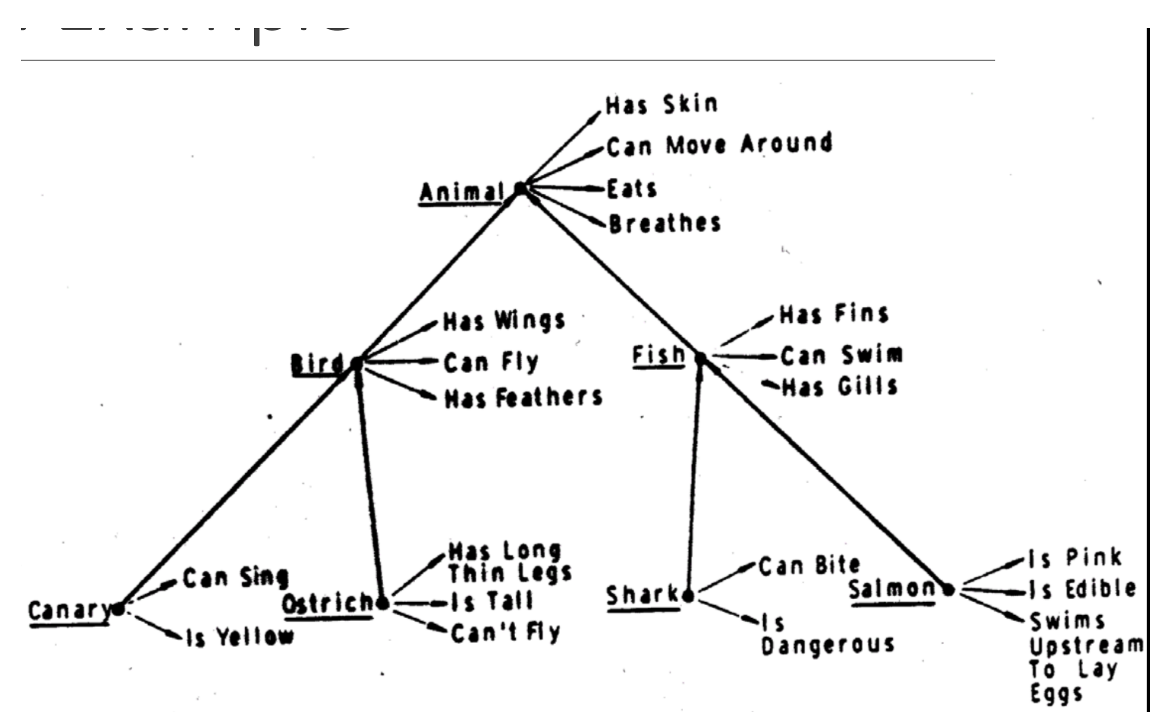
- Model the problem at hand with "logic"
- Feed into automated reasoning system
- Push button and get solution 😂

Logic	**Reasoning Tasks**
Classical	Proving
Non-monotonic	Disproving
Modal	Query answering
Probabilistic	Model computation
Temporal	Knowledge Completion
Graphs (Ontologies)	Diagnosis
Relational (Tables)	
Built-in Theories	

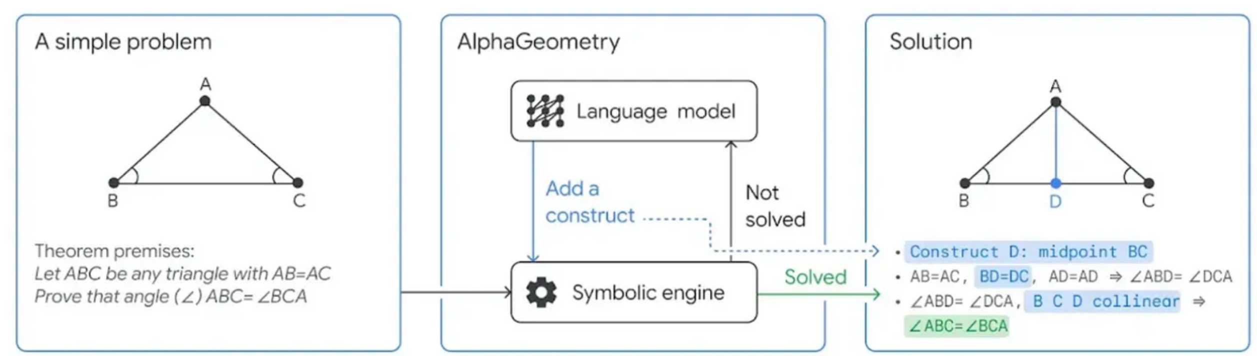
"Logic" vs "Logic Programming"?

```
flight(toronto, london).
flight(london, rome).
flight(chicago, london).
flight(X, Y) :- flight(X, Z) , flight(Z, Y).
```

Relational



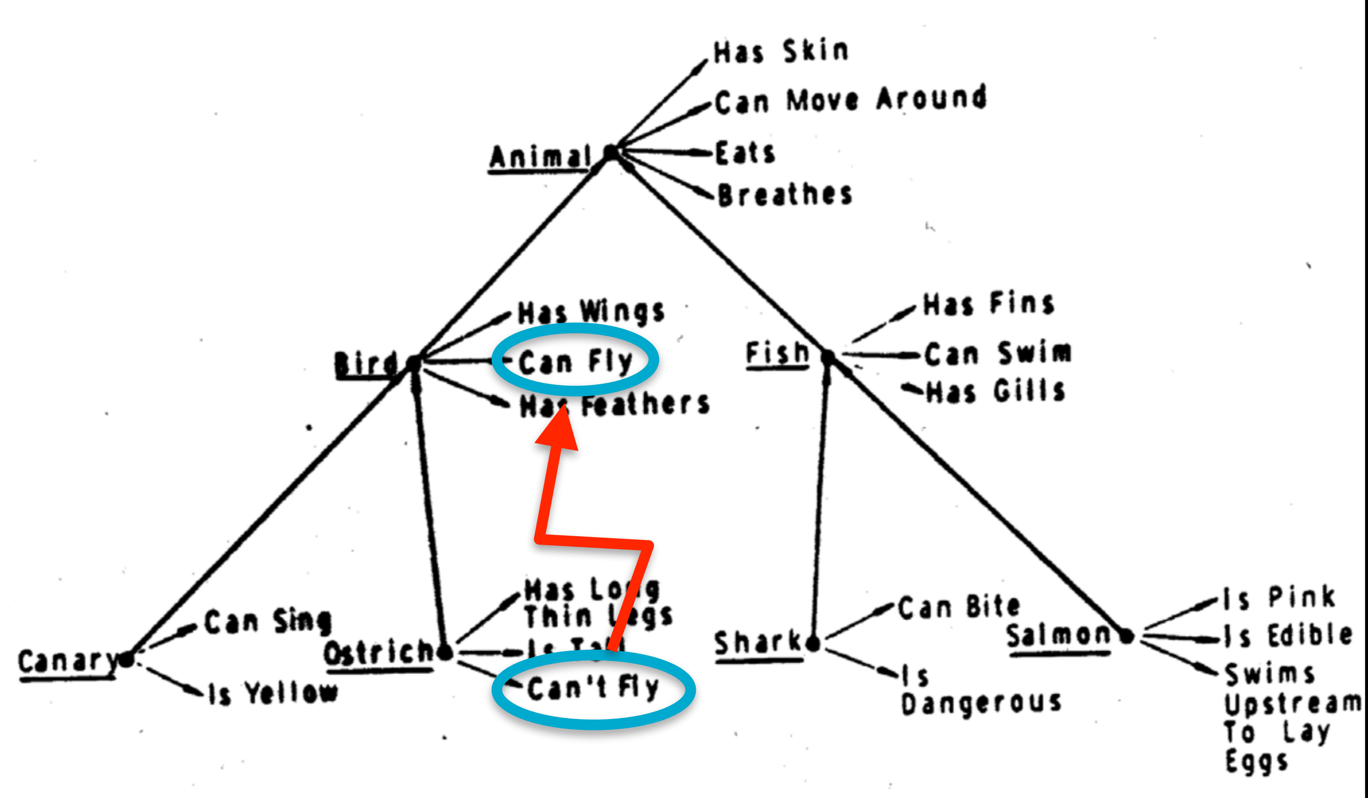
Ontology



AlphaGeometry, AlphaProof, LLM-modulo, ...

Neuro-Symbolic

Classical Logic and Logic Programming Semantics



If X is a bird

then X is an animal

If X is a bird and

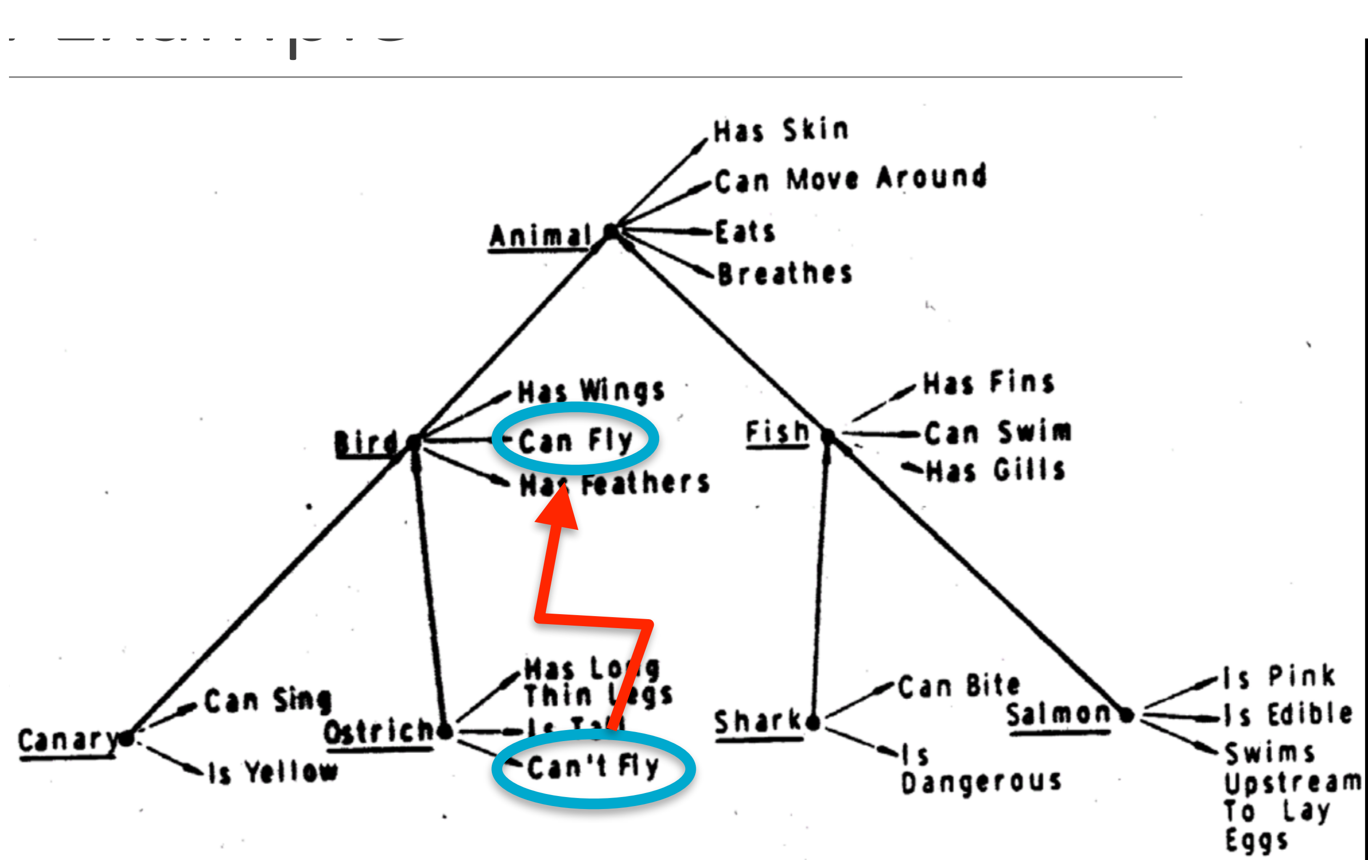
X is ***not*** *an ostrich*

then X can fly

Tweety is a bird

(Tweety is an ostrich)

Classical Logic and **Logic Programming** Semantics



If X is a bird

then X is an animal

If X is a bird and

*X is **not** an ostrich*

then X can fly

Tweety is a bird

(Tweety is an ostrich)

Classical (Open-World) Entailment

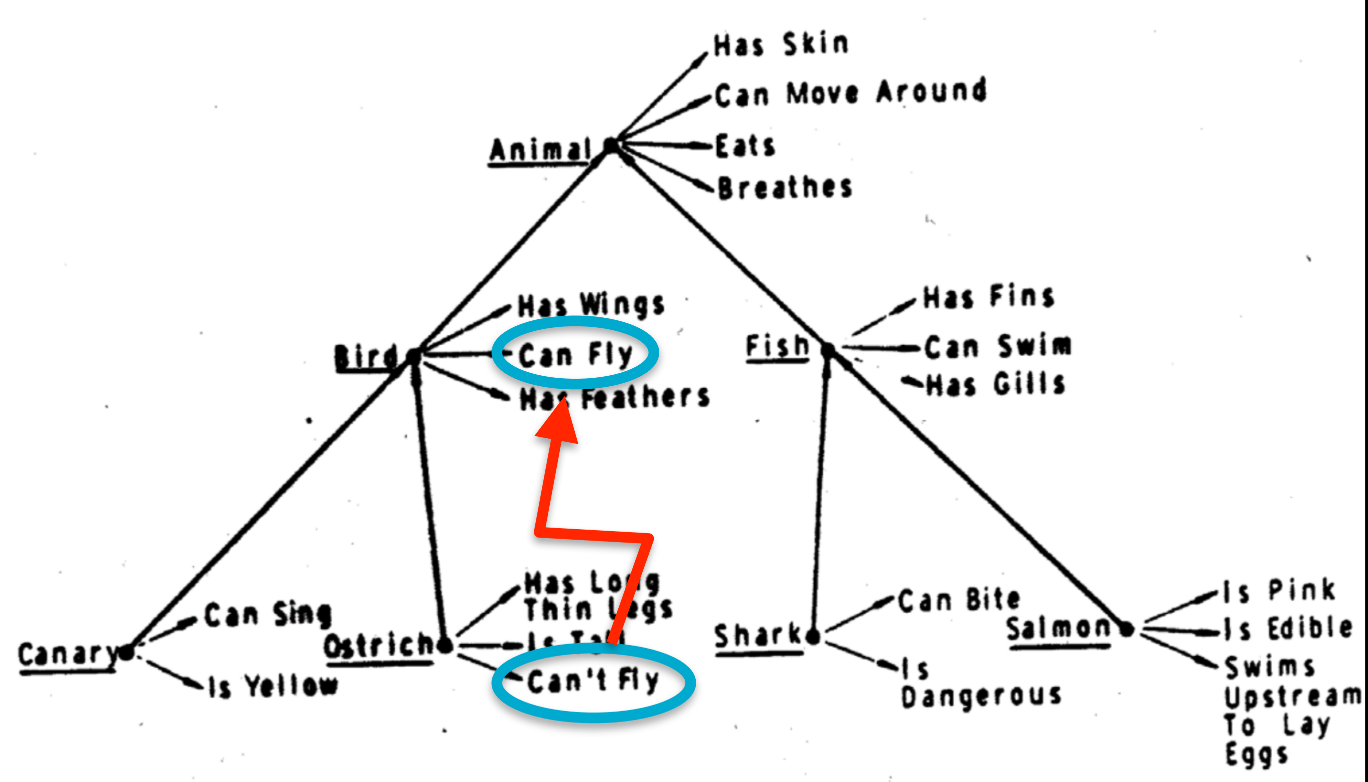
Non-Monotonic (Closed-World)

"Constraint" view

"Provability" view

Logic Programming

Classical Logic and **Logic Programming** Semantics



If X is a bird

then X is an animal

If X is a bird and

*X is **not** an ostrich*

then X can fly

Tweety is a bird

(Tweety is an ostrich)

Classical (Open-World) Entailment

✔ *Tweety is an animal*

"Constraint" view

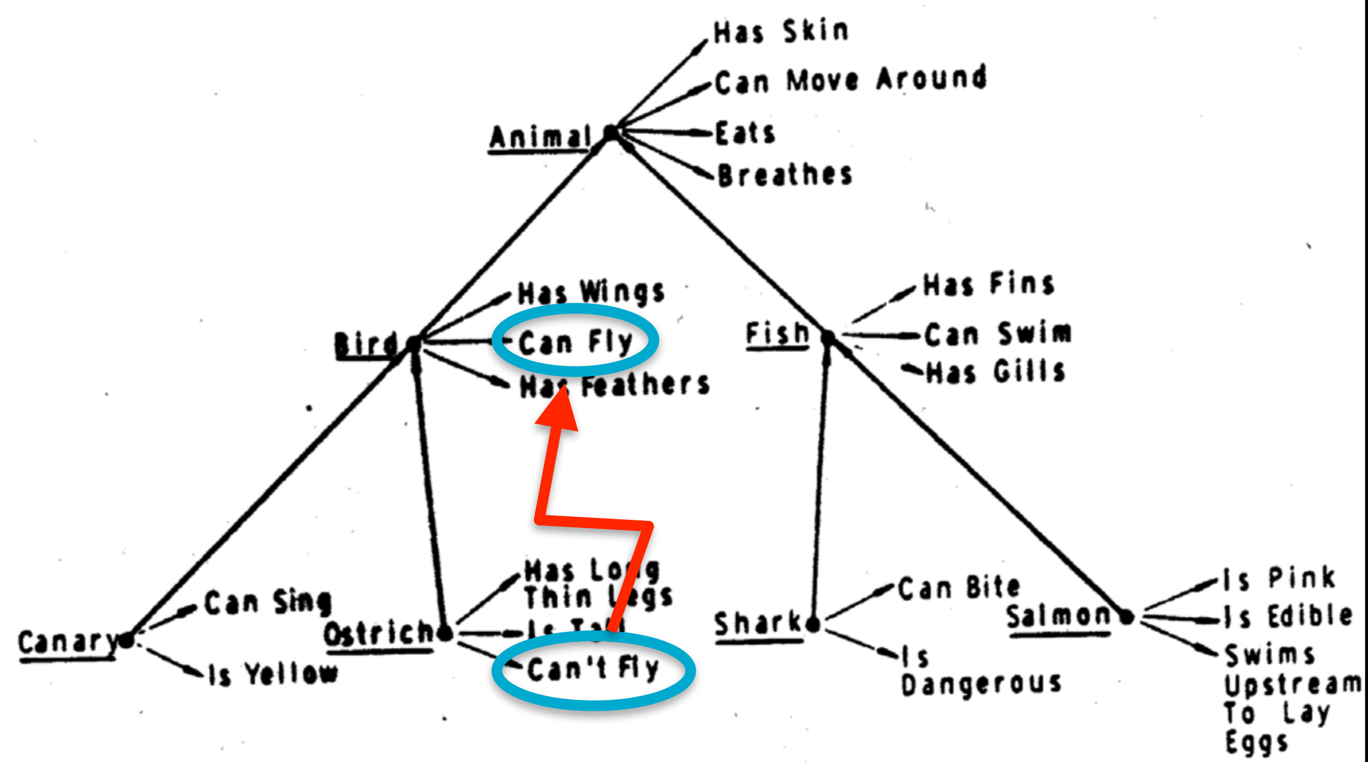
Non-Monotonic (Closed-World)

✔ *Tweety is an animal*

"Provability" view

Logic Programming

Classical Logic and **Logic Programming** Semantics



If X is a bird
then X is an animal

If X is a bird and
*X is **not** an ostrich*
then X can fly

Tweety is a bird
(Tweety is an ostrich)

Classical (Open-World) Entailment

- ✔ *Tweety is an animal*
- ✘ *Tweety can fly*

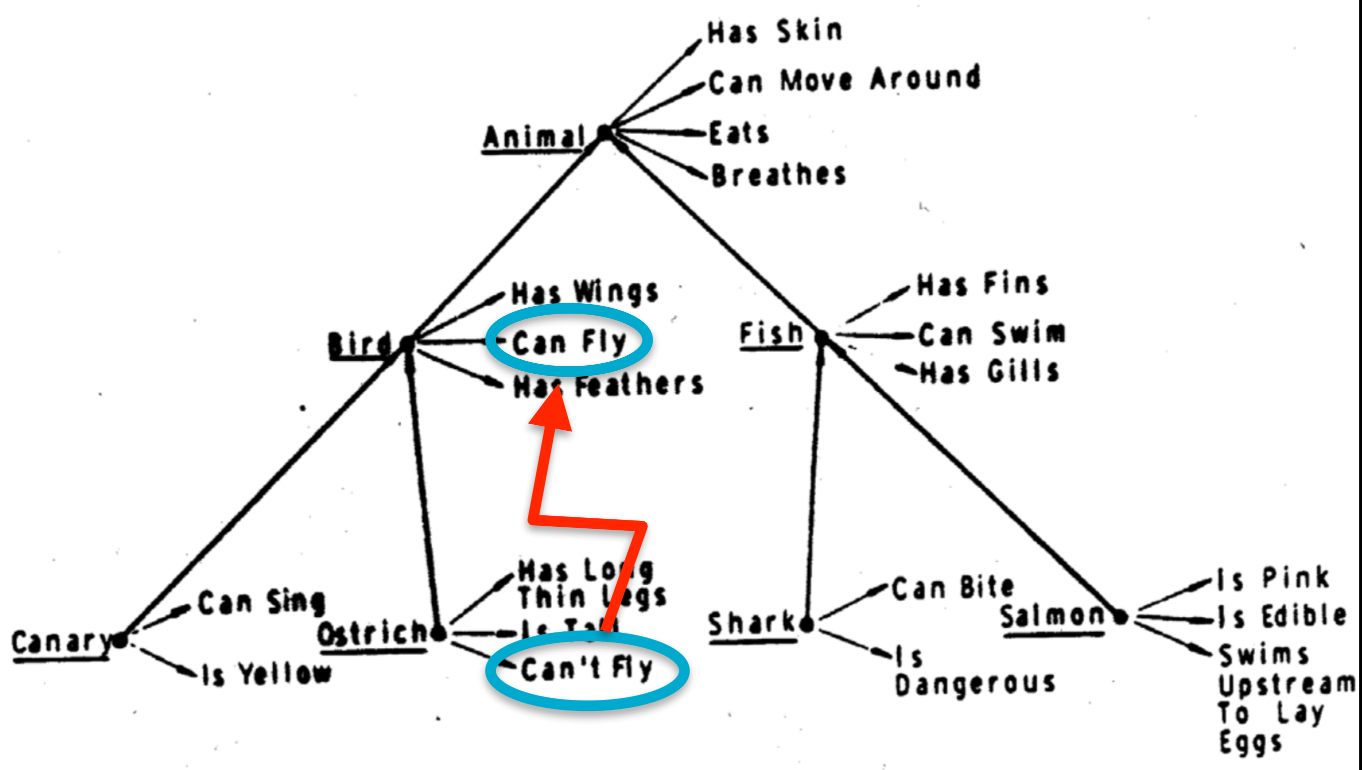
"Constraint" view

Non-Monotonic (Closed-World)

- ✔ *Tweety is an animal*
- ✔ *Tweety can fly*

"Provability" view
Logic Programming

Classical Logic and Logic Programming Semantics



If X is a bird
then X is an animal

If X is a bird and
*X is **not** an ostrich*
then X can fly

Tweety is a bird
(Tweety is an ostrich)

Classical (Open-World) Entailment

- ✔ *Tweety is an animal*
- ✘ *Tweety can fly*
- ✘ *Tweety cannot fly*

"Constraint" view

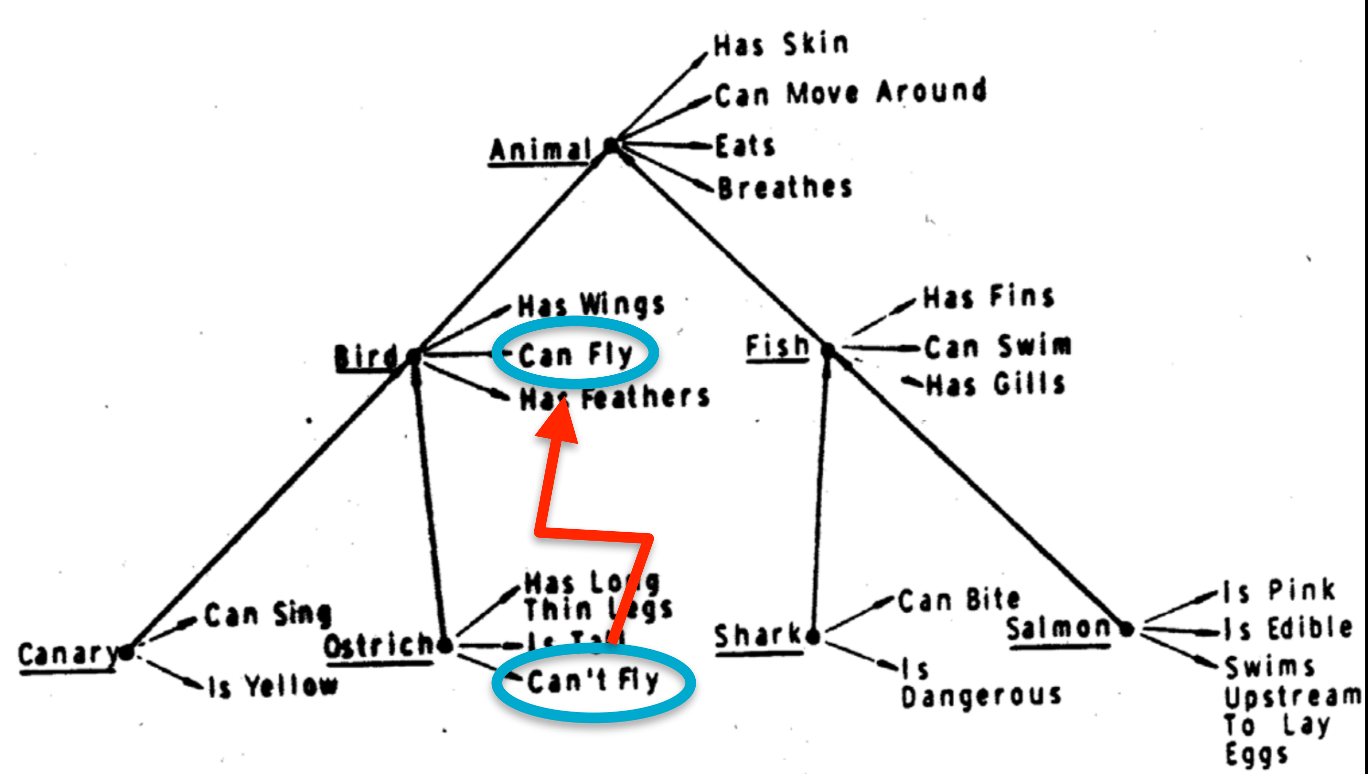
Non-Monotonic (Closed-World)

- ✔ *Tweety is an animal*
- ✔ *Tweety can fly*
- ✘ *Tweety cannot fly*

"Provability" view

Logic Programming

Classical Logic and Logic Programming Semantics



If X is a bird

then X is an animal

If X is a bird and

X is ***not*** *an ostrich*

then X can fly

Tweety is a bird

(Tweety is an ostrich)

Classical (Open-World) Entailment

- ✔ *Tweety is an animal*
- ✘ *Tweety can fly*
- ✘ *Tweety cannot fly*

"Constraint" view

Non-Monotonic (Closed-World)

- ✔ *Tweety is an animal*
- ✔ *Tweety can fly*
- ✘ *Tweety cannot fly*

"Provability" view

Logic Programming

Probabilistic Logic Programs

Facts	cat(tom).	*Tom is a cat*
Rules	drinks(X, milk) :– cat(X).	*If X is a cat then X drinks milk*
Default Negation	innocent(X) :– cat(X), **not** guilty(X). flies(X) :– bird(X), **not** abnormal(X).	*If X is a cat and X is not guilty then X is innocent*
... @ Time (Fusemate)	thirsty(X) @ T+1 :– thirsty(X) @ T, **not** drink(X, _) @ T.	*If X is thirsty at time T and X does not drink at time T then X is thirsty at time T+1*
Probabilities	0.8 :: cat(tom). 0.5 :: drinks(X, milk) :– cat(X).	
Distributions (Fusemate)	nr_siblings(X) ~ [[0, 0.05], [1, 0.10], … [5, 0.10]] :– cat(X).	
Queries	?– thirsty(tom) @ T | thirsty(tom) @ 2, drink(tom, milk) @ 5.	

Operational

Top-Down Inference

Bottom-Up Inference

Exact inference/sampling

Parameter Learning

Structure Learning

Probabilistic Logic Programs

Facts	cat(tom).	*Tom is a cat*
Rules	drinks(X, milk) :– cat(X).	*If X is a cat then X drinks milk*
Default Negation	innocent(X) :– cat(X), **not** guilty(X). flies(X) :– bird(X), **not** abnormal(X).	*If X is a cat and X is not guilty then X is innocent*
... @ Time (Fusemate)	thirsty(X) @ T+1 :– thirsty(X) @ T, **not** drink(X, _) @ T.	*If X is thirsty at time T and X does not drink at time T then X is thirsty at time T+1*
Probabilities	0.8 :: cat(tom). 0.5 :: drinks(X, milk) :– cat(X).	
Distributions (Fusemate)	nr_siblings(X) ~ [[0, 0.05], [1, 0.10], … [5, 0.10]] :– cat(X).	
Queries	?– thirsty(tom) @ T | thirsty(tom) @ 2, drink(tom, milk) @ 5.	

Operational
Top-Down Inference
Bottom-Up Inference
Exact inference/sampling
Parameter Learning
Structure Learning

Probabilistic Logic Programs

Facts	cat(tom).	*Tom is a cat*
Rules	drinks(X, milk) :– cat(X).	*If X is a cat then X drinks milk*
Default Negation	innocent(X) :– cat(X), **not** guilty(X). flies(X) :– bird(X), **not** abnormal(X).	*If X is a cat and X is not guilty then X is innocent*
... @ Time (Fusemate)	thirsty(X) @ T+1 :– thirsty(X) @ T, **not** drink(X, _) @ T.	*If X is thirsty at time T and X does not drink at time T then X is thirsty at time T+1*
Probabilities	0.8 :: cat(tom). 0.5 :: drinks(X, milk) :– cat(X).	
Distributions (Fusemate)	nr_siblings(X) ~ [[0, 0.05], [1, 0.10], … [5, 0.10]] :– cat(X).	
Queries	?– thirsty(tom) @ T | thirsty(tom) @ 2, drink(tom, milk) @ 5.	

Operational

Top-Down Inference

Bottom-Up Inference

Exact inference/sampling

Parameter Learning

Structure Learning

Probabilistic Logic Programs

Facts	cat(tom).	*Tom is a cat*
Rules	drinks(X, milk) :– cat(X).	*If X is a cat then X drinks milk*
Default Negation	innocent(X) :– cat(X), **not** guilty(X). flies(X) :– bird(X), **not** abnormal(X).	*If X is a cat and X is not guilty then X is innocent*
... @ Time (Fusemate)	thirsty(X) @ T+1 :– thirsty(X) @ T, **not** drink(X, _) @ T.	*If X is thirsty at time T and X does not drink at time T then X is thirsty at time T+1*
Probabilities	0.8 :: cat(tom). 0.5 :: drinks(X, milk) :– cat(X).	
Distributions (Fusemate)	nr_siblings(X) ~ [[0, 0.05], [1, 0.10], ... [5, 0.10]] :– cat(X).	
Queries	?– thirsty(tom) @ T | thirsty(tom) @ 2, drink(tom, milk) @ 5.	

Operational

Top-Down Inference

Bottom-Up Inference

Exact inference/sampling

Parameter Learning

Structure Learning

Probabilistic Logic Programs

Facts	cat(tom).	*Tom is a cat*
Rules	drinks(X, milk) :– cat(X).	*If X is a cat then X drinks milk*
Default Negation	innocent(X) :– cat(X), **not** guilty(X). flies(X) :– bird(X), **not** abnormal(X).	*If X is a cat and X is not guilty then X is innocent*
... @ Time (Fusemate)	thirsty(X) @ T+1 :– thirsty(X) @ T, **not** drink(X, _) @ T.	*If X is thirsty at time T and X does not drink at time T then X is thirsty at time T+1*
Probabilities	0.8 :: cat(tom). 0.5 :: drinks(X, milk) :– cat(X).	
Distributions (Fusemate)	nr_siblings(X) ~ [[0, 0.05], [1, 0.10], … [5, 0.10]] :– cat(X).	
Queries	?– thirsty(tom) @ T | thirsty(tom) @ 2, drink(tom, milk) @ 5.	

Operational

Top-Down Inference

Bottom-Up Inference

Exact inference/sampling

Parameter Learning

Structure Learning

Probabilistic Logic Programs

Facts cat(tom). *Tom is a cat*

Rules drinks(X, milk) :– cat(X). *If X is a cat then X drinks milk*

Default Negation

innocent(X) :– cat(X), **not** guilty(X).
flies(X) :– bird(X), **not** abnormal(X).

If X is a cat and X is not guilty then X is innocent

... @ Time (Fusemate)

thirsty(X) @ T+1 :–
    thirsty(X) @ T,
    not drink(X, _) @ T.

If X is thirsty at time T and X does not drink at time T then X is thirsty at time T+1

Probabilities

0.8 :: cat(tom).
0.5 :: drinks(X, milk) :– cat(X).

Distributions (Fusemate)

nr_siblings(X) ~ [[0, 0.05], [1, 0.10], … [5, 0.10]]
    :– cat(X).

Queries

?– thirsty(tom) @ T |
    thirsty(tom) @ 2, drink(tom, milk) @ 5.

Operational
Top-Down Inference
Bottom-Up Inference
Exact inference/sampling
Parameter Learning
Structure Learning

Probabilistic Logic Programs

Facts	cat(tom).	*Tom is a cat*
Rules	drinks(X, milk) :– cat(X).	*If X is a cat then X drinks milk*
Default Negation	innocent(X) :– cat(X), **not** guilty(X). flies(X) :– bird(X), **not** abnormal(X).	*If X is a cat and X is not guilty then X is innocent*
... @ Time (Fusemate)	thirsty(X) @ T+1 :– thirsty(X) @ T, **not** drink(X, _) @ T.	*If X is thirsty at time T and X does not drink at time T then X is thirsty at time T+1*
Probabilities	0.8 :: cat(tom). 0.5 :: drinks(X, milk) :– cat(X).	
Distributions (Fusemate)	nr_siblings(X) ~ [[0, 0.05], [1, 0.10], … [5, 0.10]] :– cat(X).	
Queries	?– thirsty(tom) @ T | thirsty(tom) @ 2, drink(tom, milk) @ 5.	

Operational
Top-Down Inference
Bottom-Up Inference
Exact inference/sampling
Parameter Learning
Structure Learning

Dynamic Data Structures and Distributions

Drawing without replacement

```
urn([r(1), r(2), g(1)]) @ 0.                %% Initially two red and one green distinguishable balls
draw ~ Balls @ T :-                         %% Draw  a ball uniformly if urn is not empty
    urn(Balls) @ T,
    Balls \= [].
urn(Balls -- [B]) @ T+1 :-                  %% Drawing a ball removes it from urn
    urn(Balls) @ T,
    draw = B @ T.
some(red) @ T :- draw=r(_) @ T.             %% Abstract from ball id, color only
some(green) @ T :- draw=g(_) @ T.
```

Queries

```
?- some(green) @ 0.
% 0.333333

?- some(green) @ 1 | some(red) @ 0.
% 0.5 conditional query

?- some(C1) @ 1, some(C2) @ 2 | some(red) @ 0. % Non-ground conditional query, two solutions:
% 0.5 :: [C1 = red, C2 = green]
% 0.5 :: [C1 = green, C2 = red]
```

Dynamic Data Structures and Distributions

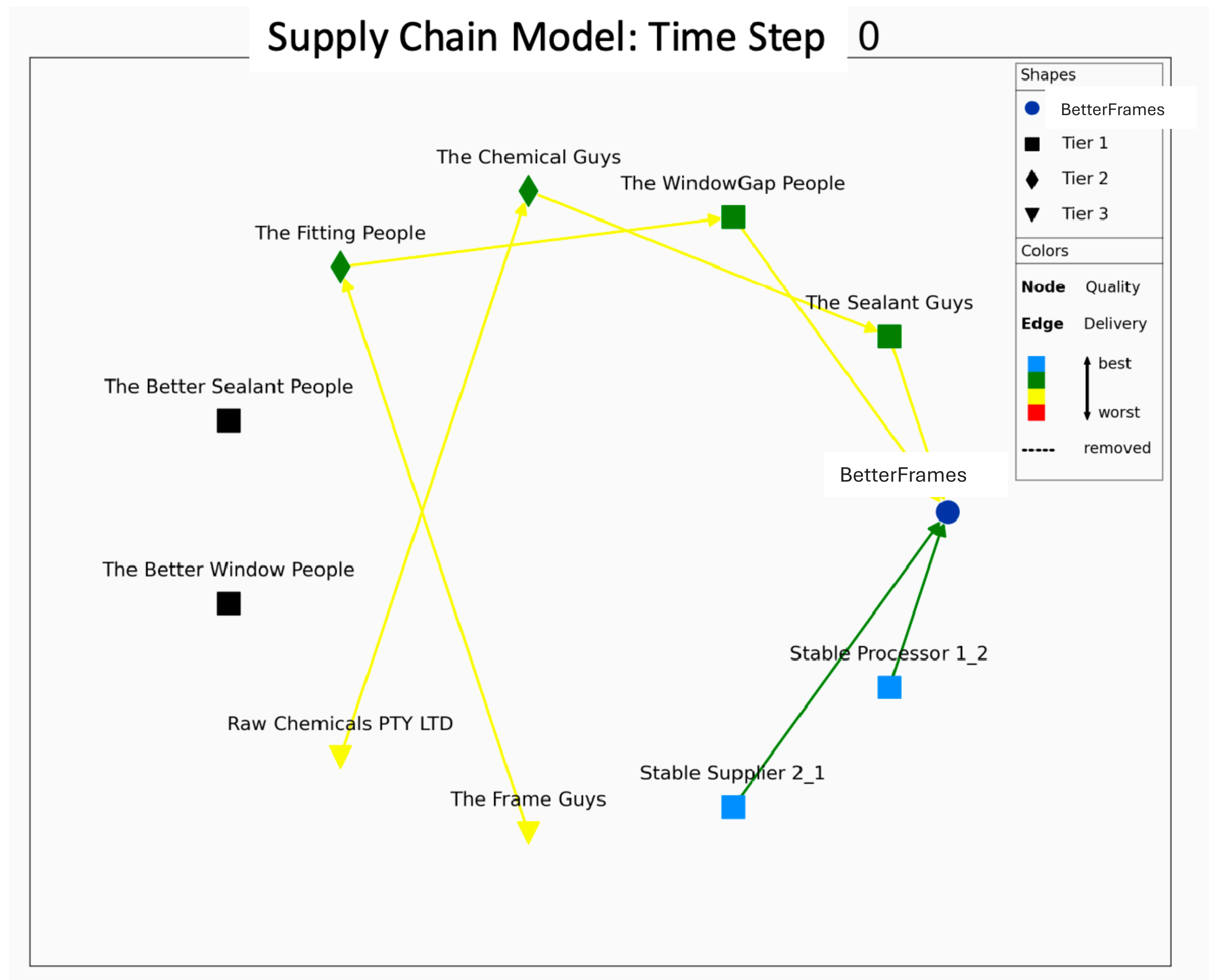
Drawing without replacement

```
urn([r(1), r(2), g(1)]) @ 0.
draw ~ Balls @ T :-
    urn(Balls) @ T,
    Balls \= [].
urn(Balls -- [B]) @ T+1 :-
    urn(Balls) @ T,
    draw = B @ T.
some(red) @ T :- draw=r(_) @ T.
some(green) @ T :- draw=g(_) @ T.
```

Queries

```
?- some(green) @ 0.
% 0.333333

?- some(green) @ 1 | some(red) @ 0.
% 0.5 conditional query

?- some(C1) @ 1, some(C2) @ 2 | some(red)
% 0.5 :: [C1 = red, C2 = green]
% 0.5 :: [C1 = green, C2 = red]
```

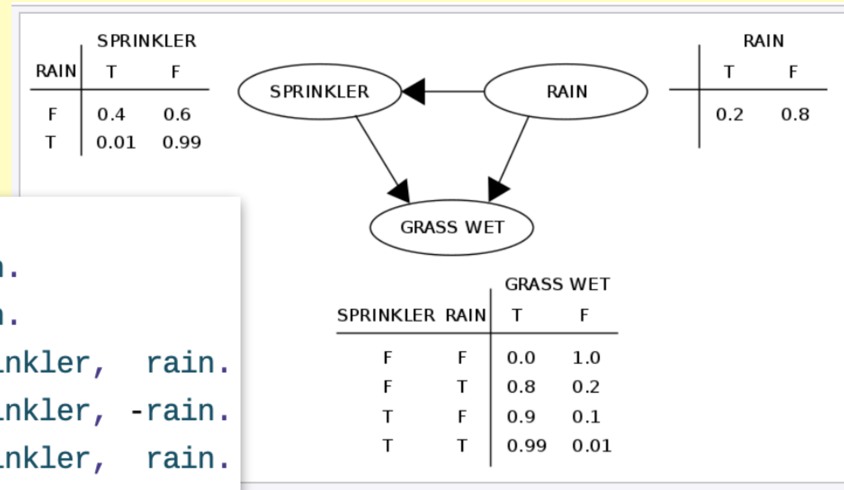
D61 Supply Chain Risk Assessment Application



Probabilistic Logic Programming (Fusemate)

Probabilistic Logic Programming (Fusemate)

Bayes Network

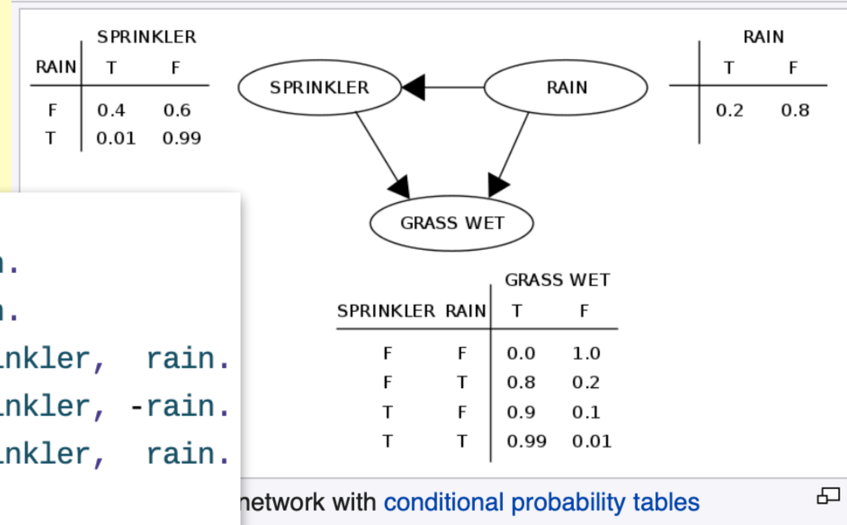


network with conditional probability tables

```
0.2  :: rain.
0.01 :: sprinkler :-  rain.
0.4  :: sprinkler :- -rain.
0.99 :: grasswet  :-  sprinkler,  rain.
0.9  :: grasswet  :-  sprinkler, -rain.
0.8  :: grasswet  :- -sprinkler,  rain.

?- rain | grasswet.
```

Probabilistic Logic Programming (Fusemate)

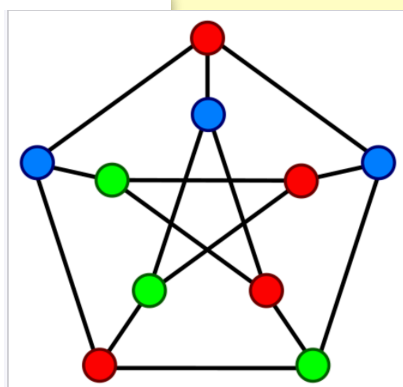
Bayes Network




```
0.2  :: rain.
0.01 :: sprinkler :-  rain.
0.4  :: sprinkler :- -rain.
0.99 :: grasswet  :-  sprinkler,  rain.
0.9  :: grasswet  :-  sprinkler, -rain.
0.8  :: grasswet  :- -sprinkler,  rain.

?- rain | grasswet.
```

NP-Complete Search Problems

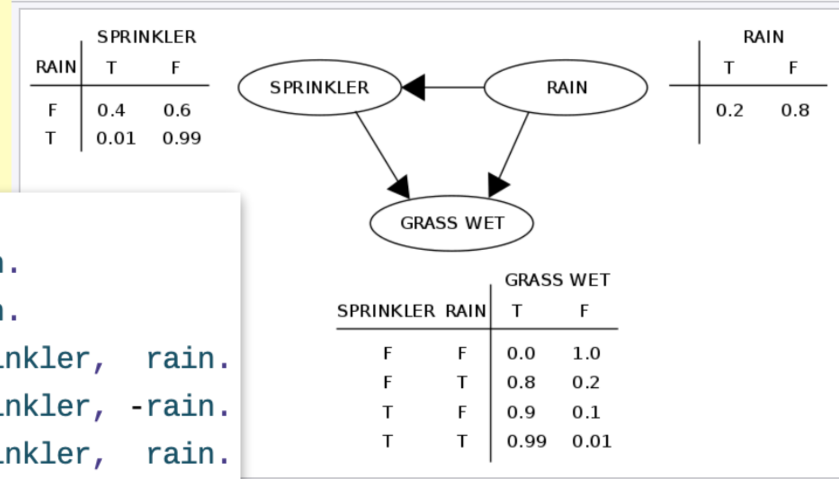
```
color(X) in [r, g, b] :- node(X).
node(1).  node(2). node(3). node(4).
noncol :- color(X)=r, color(Y)=r, edge(X,Y).
noncol :- color(X)=g, color(Y)=g, edge(X,Y).
noncol :- color(X)=b, color(Y)=b, edge(X,Y).
edge(1,2). edge(2,3). edge(3,1).

edge(4,1).
edge(4,2).
%edge(4,3).

%% \+ noncol is true iff there is a coloring
?- -noncol, color(1)=C1, color(2)=C2, color(3)=C3, color(4)=C4.
```



Logical variables X for domain objects

Probabilistic Logic Programming (Fusemate)

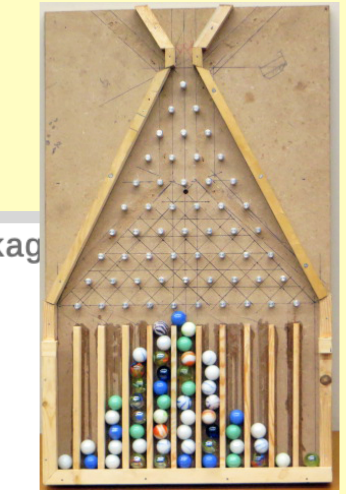
Bayes Network



network with conditional probability tables

```
0.2  :: rain.
0.01 :: sprinkler :-  rain.
0.4  :: sprinkler :- -rain.
0.99 :: grasswet  :-  sprinkler,  rain.
0.9  :: grasswet  :-  sprinkler, -rain.
0.8  :: grasswet  :- -sprinkler,  rain.

?- rain | grasswet.
```

Time

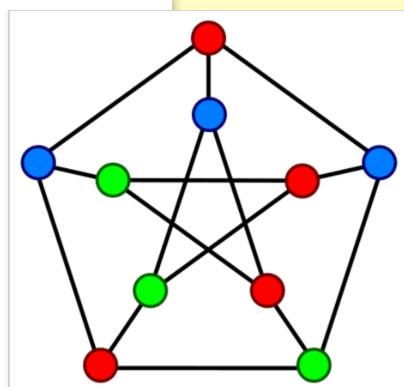


```
%% Some "random" blockag
block(1) @ 2.
block(2) @ 3.


prob(0).
(0.5 :: prob(K+1) + prob(K)) @ N+1 :-
    prob(K) @ N,
    \+ bl(K) @ N.
prob(K) @ N+1 :-
    prob(K) @ N,
    bl(K) @ N.
```

```
%% ?- prob(K) @ 4.
0.0625 :: prob(0) @ 4
0.3750 :: prob(1) @ 4
0.43750 :: prob(2) @ 4
0.0625 :: prob(3) @ 4
0.0625 :: prob(4) @ 4
```

NP-Complete Search Problems

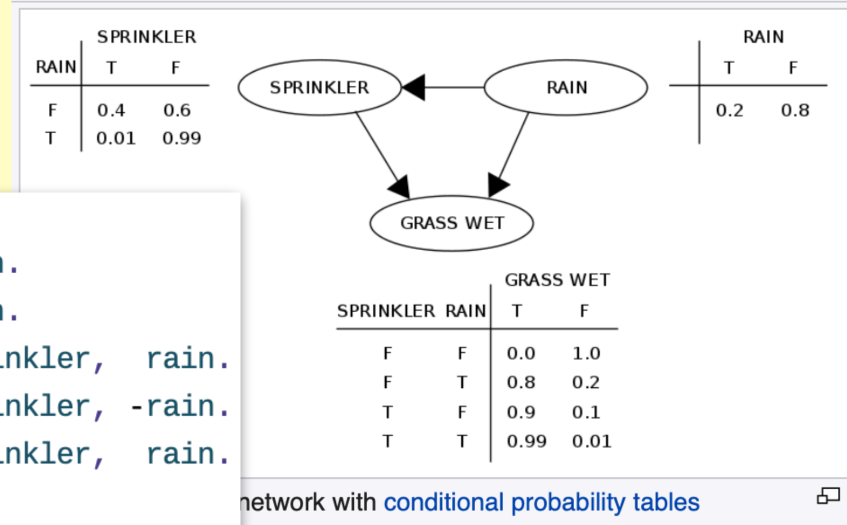
```
color(X) in [r, g, b] :- node(X).
node(1).  node(2). node(3). node(4).
noncol :- color(X)=r, color(Y)=r, edge(X,Y).
noncol :- color(X)=g, color(Y)=g, edge(X,Y).
noncol :- color(X)=b, color(Y)=b, edge(X,Y).
edge(1,2). edge(2,3). edge(3,1).

edge(4,1).
edge(4,2).
%edge(4,3).

%% \+ noncol is true iff there is a coloring
?- -noncol, color(1)=C1, color(2)=C2, color(3)=C3, color(4)=C4.
```



Logical variables X for domain objects

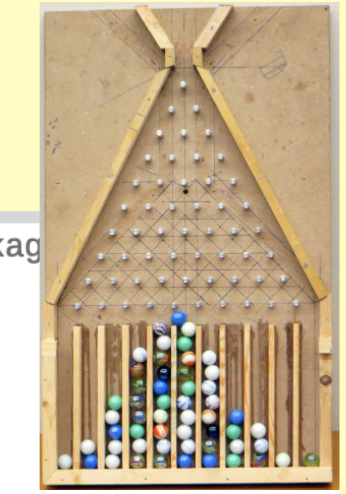
Probabilistic Logic Programming (Fusemate)

Bayes Network




```
0.2  :: rain.
0.01 :: sprinkler :-  rain.
0.4  :: sprinkler :- -rain.
0.99 :: grasswet  :-  sprinkler,  rain.
0.9  :: grasswet  :-  sprinkler, -rain.
0.8  :: grasswet  :- -sprinkler,  rain.

?- rain | grasswet.
```

Time

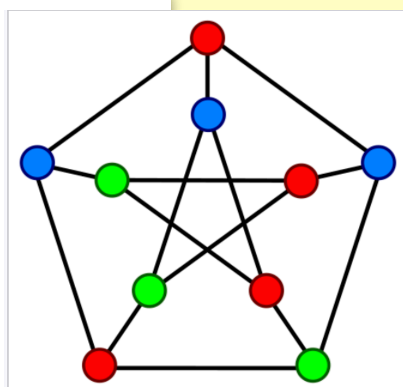


```
%% Some "random" blockag
block(1) @ 2.
block(2) @ 3.


prob(0).
(0.5 :: prob(K+1) + prob(K)) @ N+1 :-
    prob(K) @ N,
    \+ bl(K) @ N.
prob(K) @ N+1 :-
    prob(K) @ N,
    bl(K) @ N.
```

```
%% ?- prob(K) @ 4.
0.0625 :: prob(0) @ 4
0.3750 :: prob(1) @ 4
0.43750 :: prob(2) @ 4
0.0625 :: prob(3) @ 4
0.0625 :: prob(4) @ 4
```

NP-Complete Search Problems

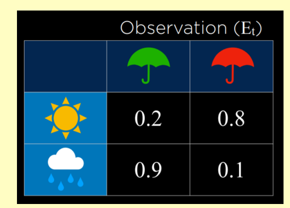
```
color(X) in [r, g, b] :- node(X).
node(1).  node(2). node(3). node(4).
noncol :- color(X)=r, color(Y)=r, edge(X,Y).
noncol :- color(X)=g, color(Y)=g, edge(X,Y).
noncol :- color(X)=b, color(Y)=b, edge(X,Y).
edge(1,2). edge(2,3). edge(3,1).

edge(4,1).
edge(4,2).
%edge(4,3).

%% \+ noncol is true iff there is a coloring
?- -noncol, color(1)=C1, color(2)=C2, color(3)=C3, color(4)=C4.
```

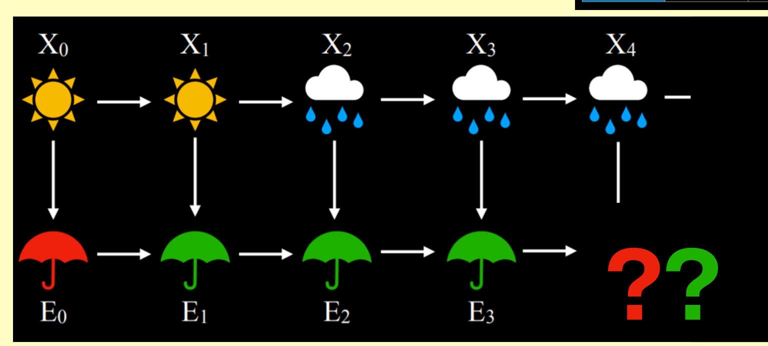


Logical variables X for domain objects

Hidden

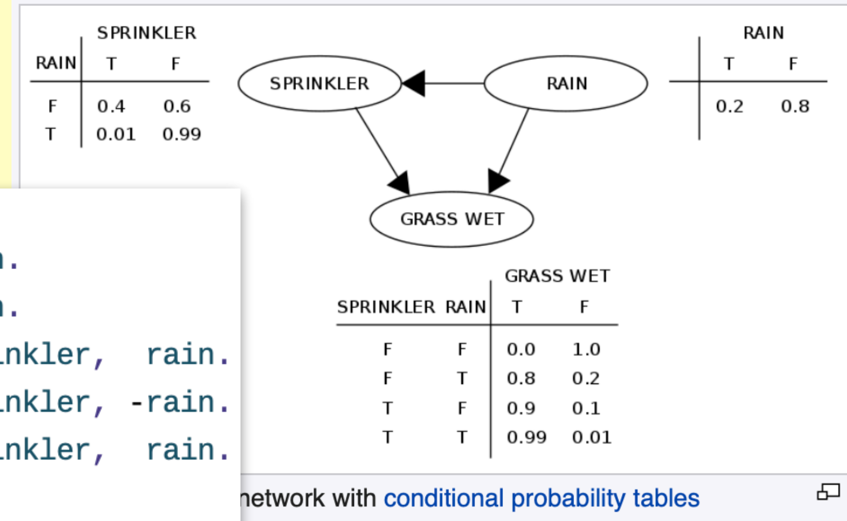
Markov Models





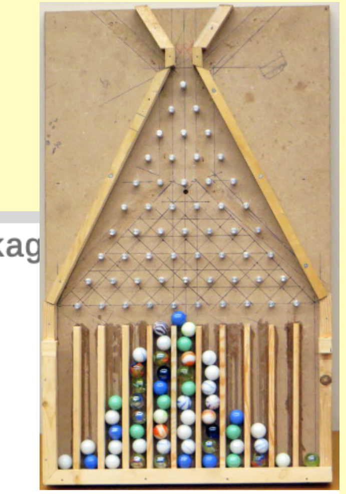
Probabilistic Logic Programming (Fusemate)

Bayes Network




```
0.2  :: rain.
0.01 :: sprinkler :-  rain.
0.4  :: sprinkler :- -rain.
0.99 :: grasswet  :-  sprinkler,  rain.
0.9  :: grasswet  :-  sprinkler, -rain.
0.8  :: grasswet  :- -sprinkler,  rain.

?- rain | grasswet.
```

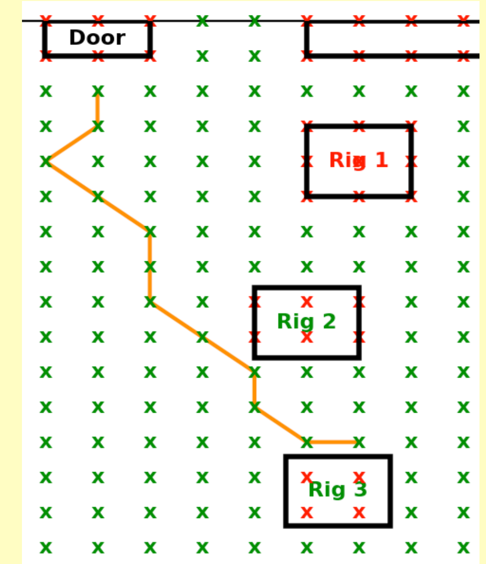
Time



```
%% Some "random" blockag
block(1) @ 2.
block(2) @ 3.

prob(0).
(0.5 :: prob(K+1) + prob(K)) @ N+1 :-
    prob(K) @ N,
    \+ bl(K) @ N.
prob(K) @ N+1 :-
    prob(K) @ N,
    bl(K) @ N.
```

```
%% ?- prob(K) @ 4.
0.0625 :: prob(0) @ 4
0.3750 :: prob(1) @ 4
0.43750 :: prob(2) @ 4
0.0625 :: prob(3) @ 4
0.0625 :: prob(4) @ 4
```

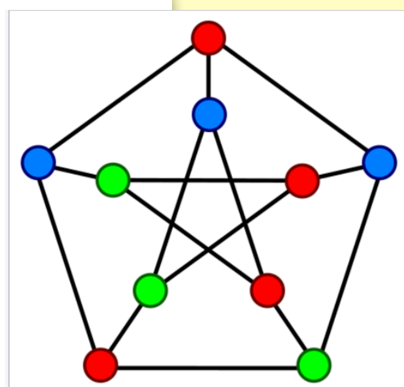
Algorithms



Probabilistic A*

NP-Complete Search Problems

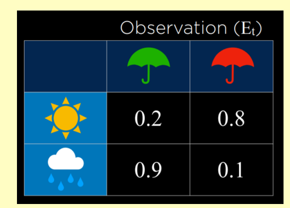
```
color(X) in [r, g, b] :- node(X).
node(1).  node(2). node(3). node(4).
noncol :- color(X)=r, color(Y)=r, edge(X,Y).
noncol :- color(X)=g, color(Y)=g, edge(X,Y).
noncol :- color(X)=b, color(Y)=b, edge(X,Y).
edge(1,2). edge(2,3). edge(3,1).

edge(4,1).
edge(4,2).
%edge(4,3).

%% \+ noncol is true iff there is a coloring
?- -noncol, color(1)=C1, color(2)=C2, color(3)=C3, color(4)=C4.
```

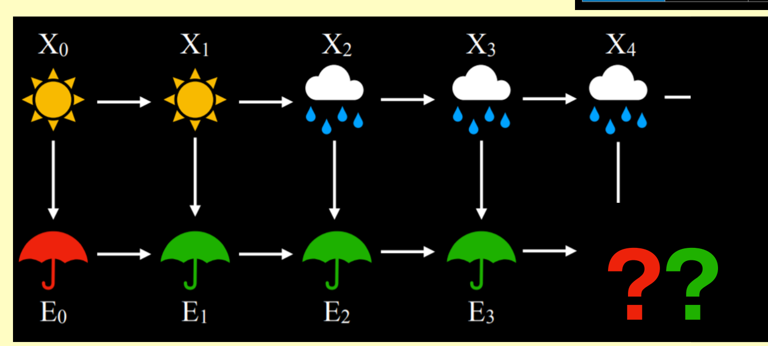


Logical variables X for domain objects

Hidden

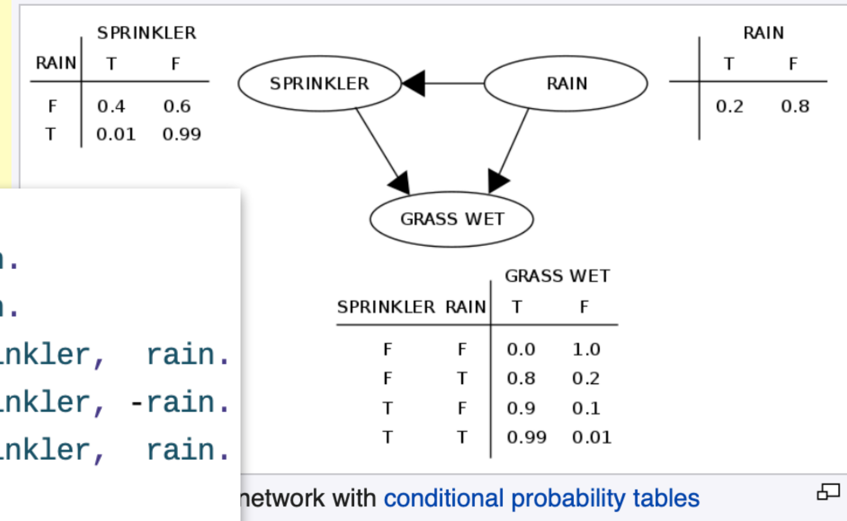
Markov Models





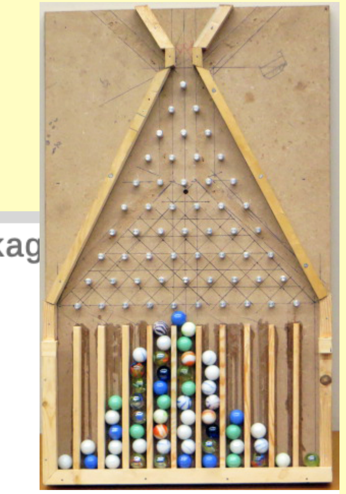
Probabilistic Logic Programming (Fusemate)

Bayes Network




```
0.2  :: rain.
0.01 :: sprinkler :-  rain.
0.4  :: sprinkler :- -rain.
0.99 :: grasswet  :-  sprinkler,  rain.
0.9  :: grasswet  :-  sprinkler, -rain.
0.8  :: grasswet  :- -sprinkler,  rain.

?- rain | grasswet.
```

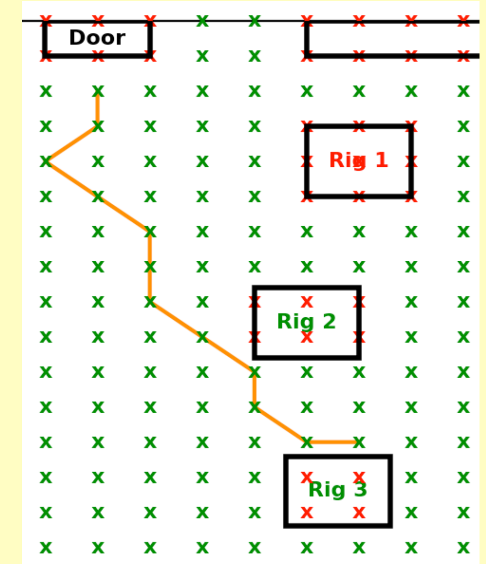
Time



```
%% Some "random" blockag
block(1) @ 2.
block(2) @ 3.


prob(0).
(0.5 :: prob(K+1) + prob(K)) @ N+1 :-
    prob(K) @ N,
    \+ bl(K) @ N.
prob(K) @ N+1 :-
    prob(K) @ N,
    bl(K) @ N.
```

```
%% ?- prob(K) @ 4.
0.0625 :: prob(0) @ 4
0.3750 :: prob(1) @ 4
0.43750 :: prob(2) @ 4
0.0625 :: prob(3) @ 4
0.0625 :: prob(4) @ 4
```

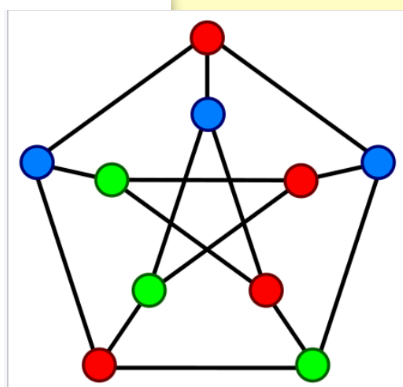
Algorithms



Probabilistic A*

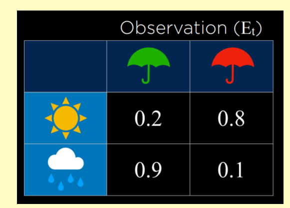
NP-Complete Search Problems

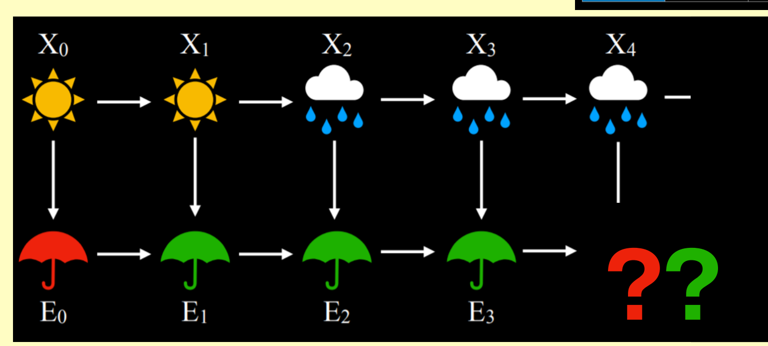
```
color(X) in [r, g, b] :- node(X).
node(1).  node(2). node(3). node(4).
noncol :- color(X)=r, color(Y)=r, edge(X,Y).
noncol :- color(X)=g, color(Y)=g, edge(X,Y).
noncol :- color(X)=b, color(Y)=b, edge(X,Y).
edge(1,2). edge(2,3). edge(3,1).

edge(4,1).
edge(4,2).
%edge(4,3).

%% \+ noncol is true iff there is a coloring
?- -noncol, color(1)=C1, color(2)=C2, color(3)=C3, color(4)=C4.
```



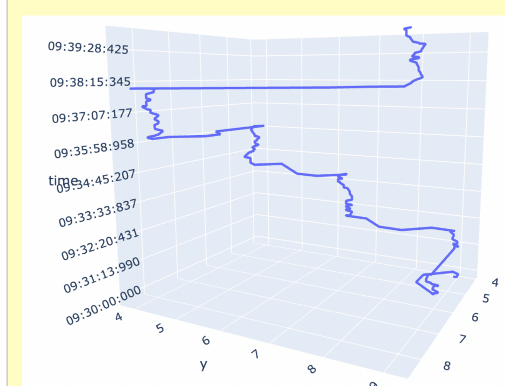
Logical variables X for domain objects

Hidden Markov Models



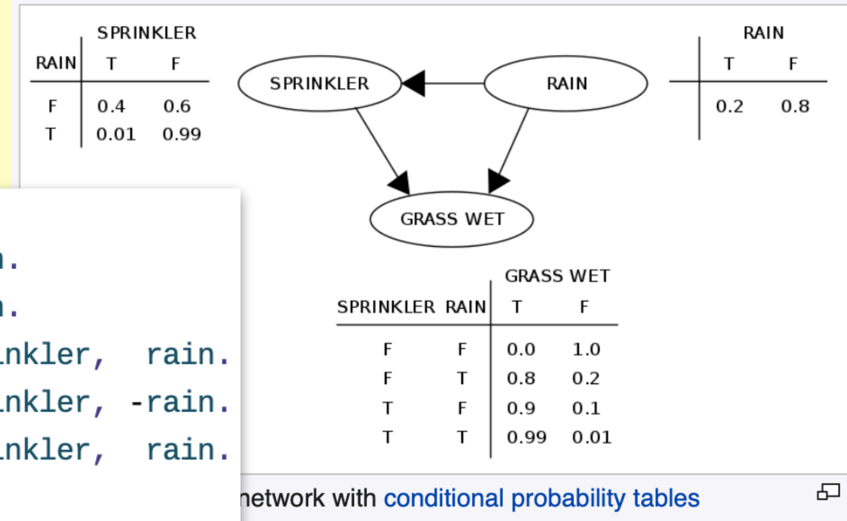


Simulation by Sampling



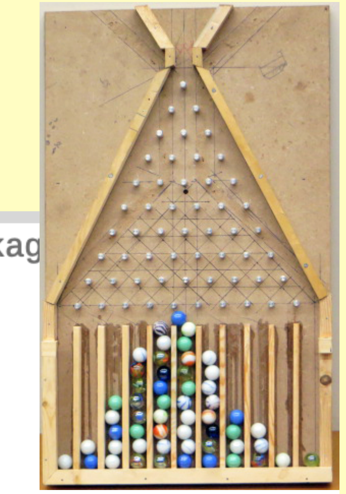
Probabilistic Logic Programming (Fusemate)

Bayes Network




```
0.2  :: rain.
0.01 :: sprinkler :-  rain.
0.4  :: sprinkler :- -rain.
0.99 :: grasswet  :-  sprinkler,  rain.
0.9  :: grasswet  :-  sprinkler, -rain.
0.8  :: grasswet  :- -sprinkler,  rain.

?- rain | grasswet.
```

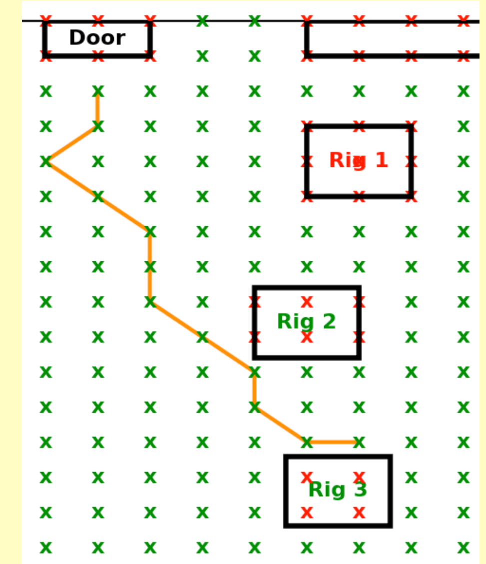
Time



```
%% Some "random" blockag
block(1) @ 2.
block(2) @ 3.


prob(0).
(0.5 :: prob(K+1) + prob(K)) @ N+1 :-
    prob(K) @ N,
    \+ bl(K) @ N.
prob(K) @ N+1 :-
    prob(K) @ N,
    bl(K) @ N.
```

```
%% ?- prob(K) @ 4.
0.0625 :: prob(0) @ 4
0.3750 :: prob(1) @ 4
0.43750 :: prob(2) @ 4
0.0625 :: prob(3) @ 4
0.0625 :: prob(4) @ 4
```

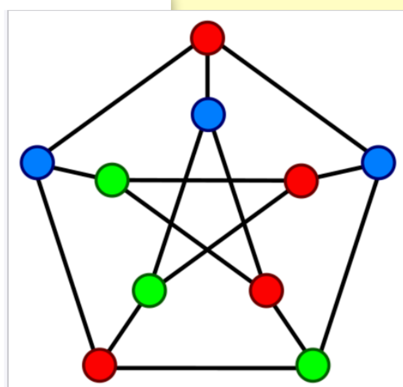
Algorithms



Probabilistic A*

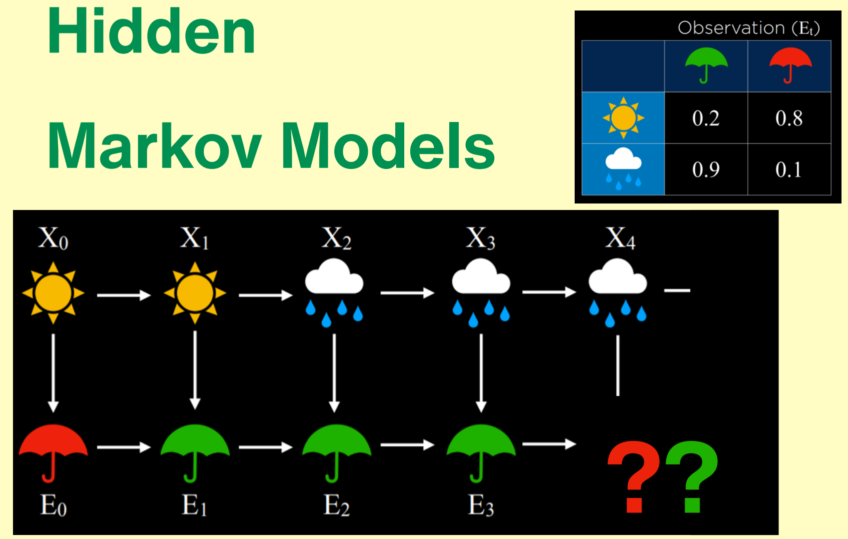
NP-Complete Search Problems

```
color(X) in [r, g, b] :- node(X).
node(1).  node(2). node(3). node(4).
noncol :- color(X)=r, color(Y)=r, edge(X,Y).
noncol :- color(X)=g, color(Y)=g, edge(X,Y).
noncol :- color(X)=b, color(Y)=b, edge(X,Y).
edge(1,2). edge(2,3). edge(3,1).

edge(4,1).
edge(4,2).
%edge(4,3).

%% \+ noncol is true iff there is a coloring
?- -noncol, color(1)=C1, color(2)=C2, color(3)=C3, color(4)=C4.
```

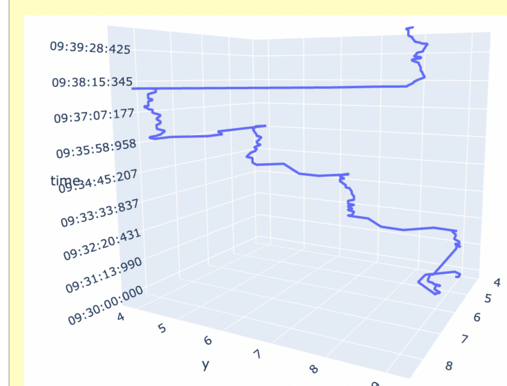


Logical variables X for domain objects

Hidden Markov Models



Simulation by Sampling



Expressivity: full history (non-Markovian); random variables are first-class citizens

Fusemate Probabilistic Logic Programming System

Implementation in Python

(From earlier versions in Scala)

Two-way interface Python <-> Fusemate

Python data structures available in Fusemate

Logic program can be written as Python functions

Efficient probablistic inference

Default negation via well-founded model

Rules cannot change past states

Two-phase inference algorithm

- Phase 1 “grounding”

  Removal of first-order variables

  -> Bayes-net like program (may contain cycles)

- Pase 2 inference/sampling

  Top-down variable elimination with caching of results

Strong Python integration

```
def weather_0():
    return {'rainy': 0.5, 'sunny': 0.5}

def weather_Tp1(weather_T):
    return {'rainy':  {'rainy': 0.8, 'sunny': 0.2},
            'sunny': {'rainy': 0.2, 'sunny': 0.8}}\
            [weather_T]

def obs_T(weather_T):
    return {'rainy':  {'shop': 0.8, 'clean': 0.2},
             'sunny': {'shop': 0.2, 'walk': 0.8}}\
             [weather_T]
```

Fusemate Probabilistic Logic Programming System

Implementation in Python

(From earlier versions in Scala)

Two-way interface Python <-> Fusemate

Python data structures available in Fusemate

Logic program can be written as Python functions

Efficient probablistic inference

Default negation via well-founded model

Rules cannot change past states

Two-phase inference algorithm

- Phase 1 “grounding”

  Removal of first-order variables

  -> Bayes-net like program (may contain cycles)

- Pase 2 inference/sampling

  Top-down variable elimination with caching of results

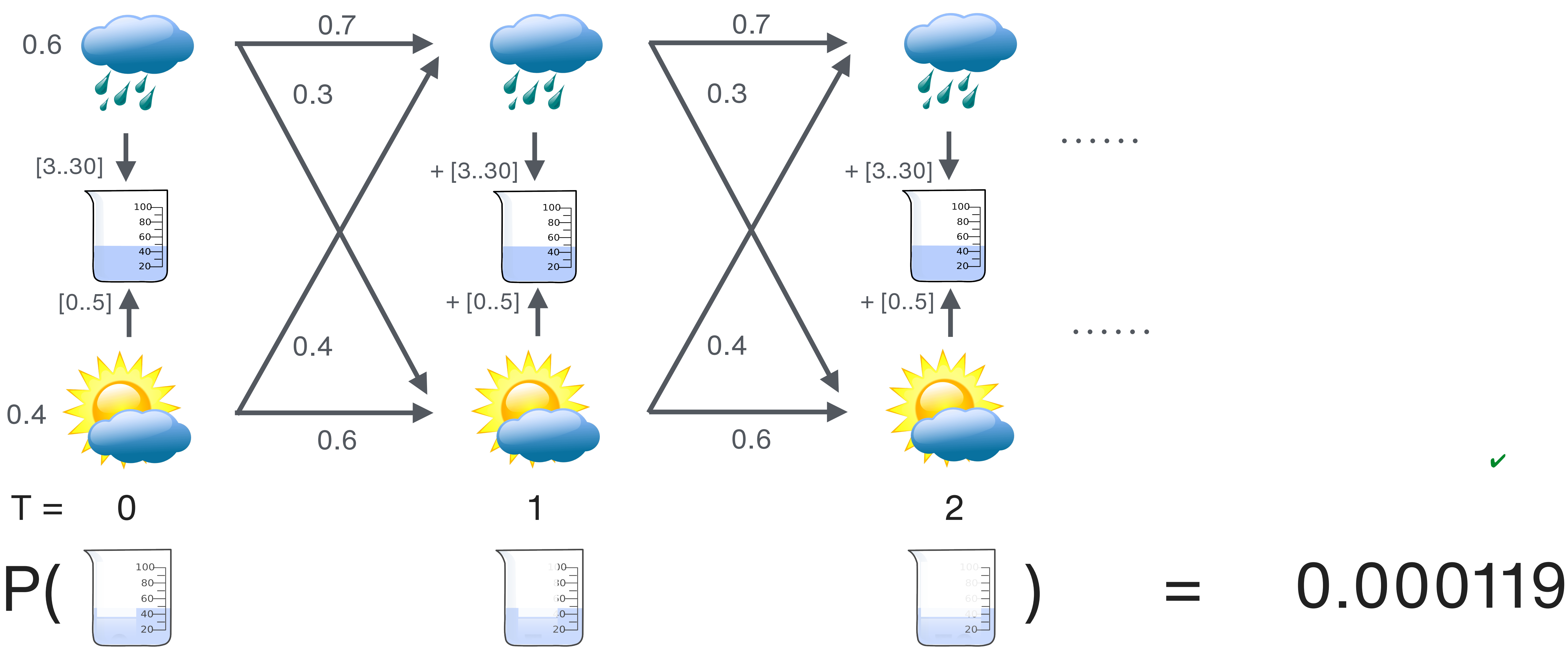
Strong Python integration

```
def weather_0():
    return {'rainy': 0.5, 'sunny': 0.5}

def weather_Tp1(weather_T):
    return {'rainy':  {'rainy': 0.8, 'sunny': 0.2},
              'sunny': {'rainy': 0.2, 'sunny': 0.8}}\
             [weather_T]

def obs_T(weather_T):
    return {'rainy':  {'shop': 0.8, 'clean': 0.2},
              'sunny': {'shop': 0.2, 'walk': 0.8}}\
              [weather_T]
```

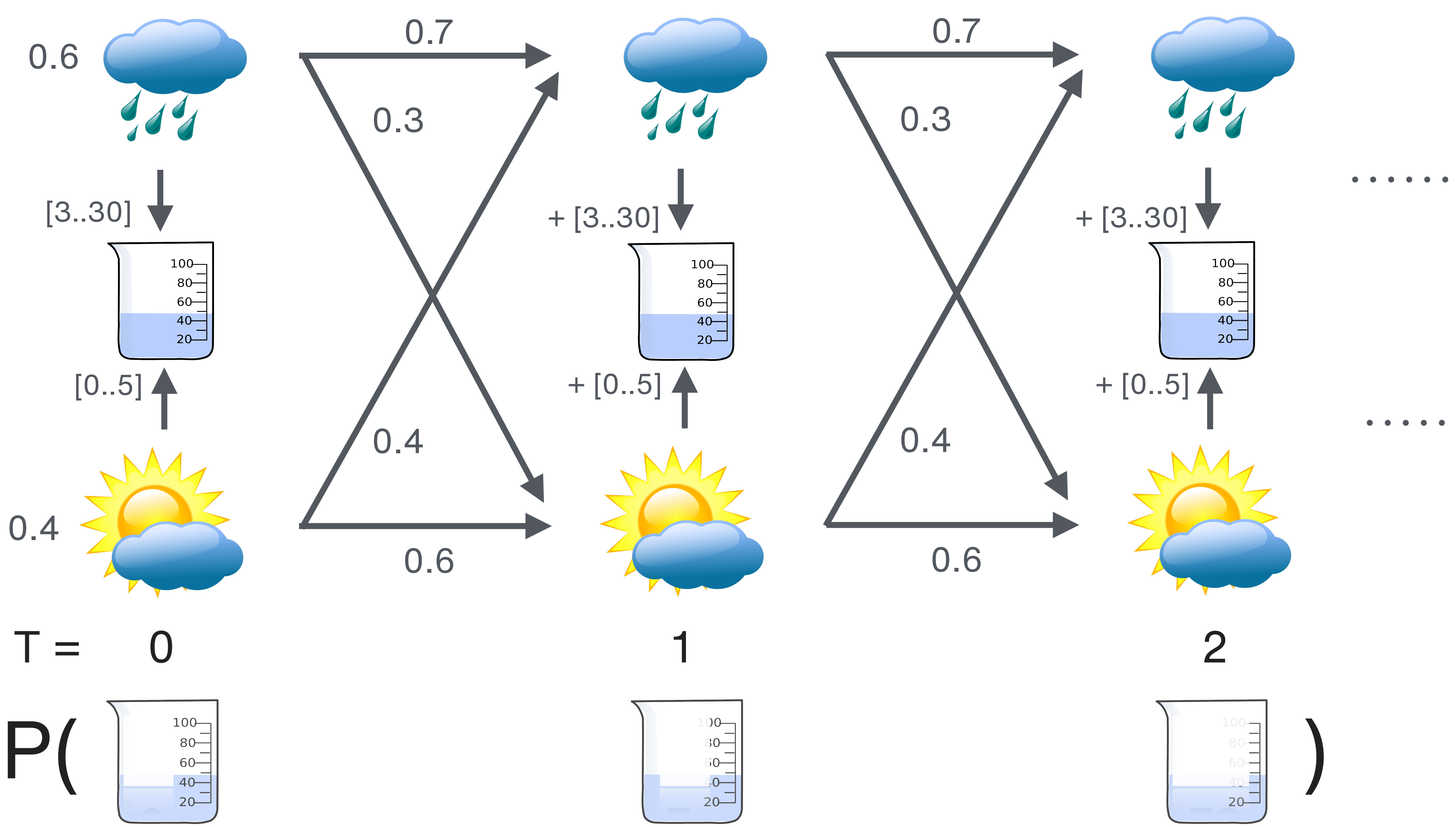
Contribution:
“Inconsistency Pruning”
for better efficiency

Peter Baumgartner and Elena Tartaglia. *Bottom-Up Stratified Probabilistic Logic Programming with Fusemate*. ICLP 2023

Fusemate Inconsistency Pruning



Fusemate Inconsistency Pruning

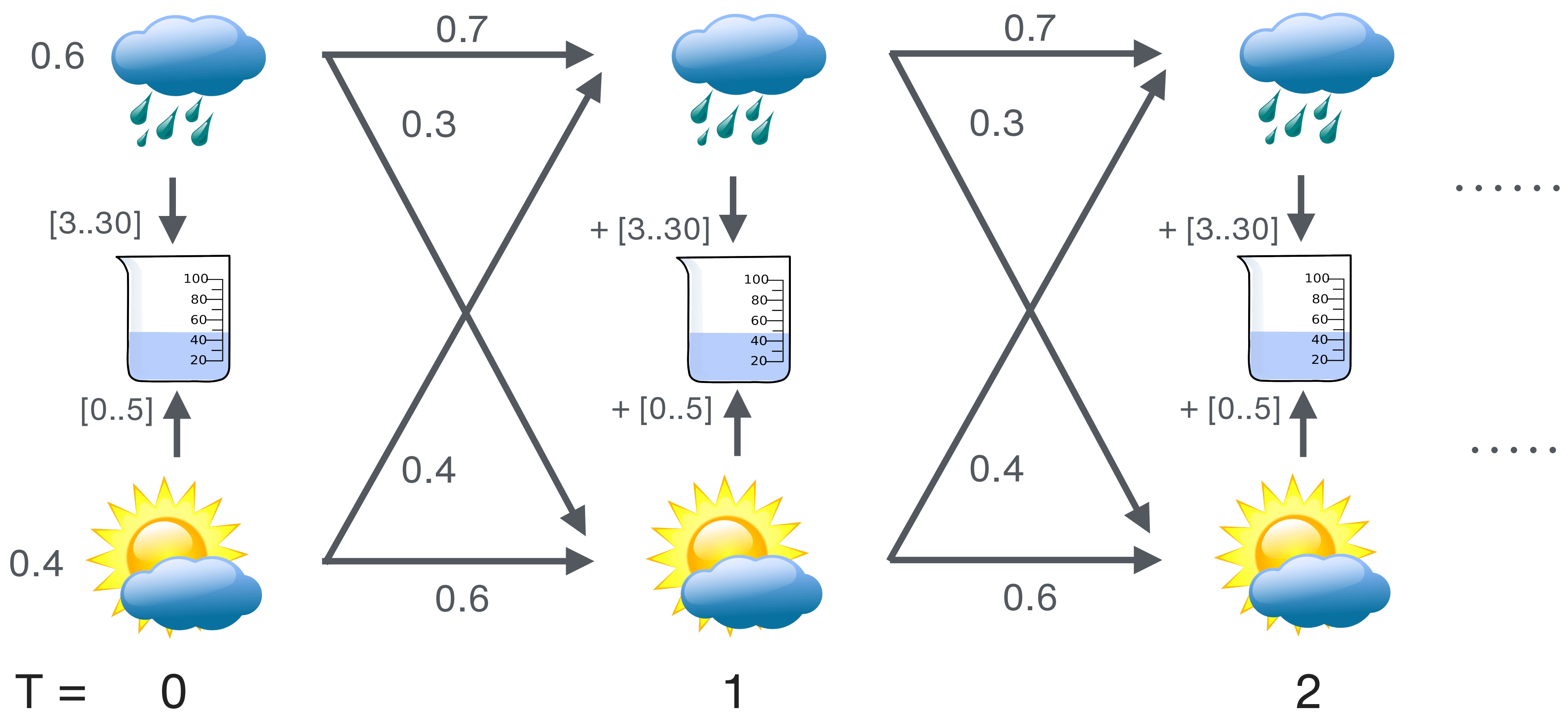


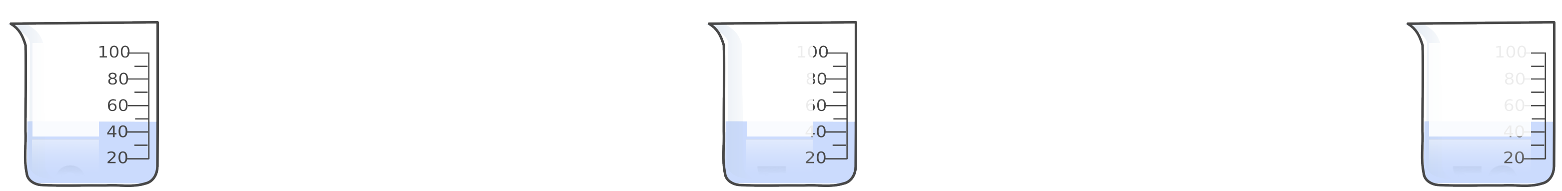
= 0.000119

```
state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.

state ~ [[rainy, 0.7], [sunny, 0.3]] @ T+1 :-
    state=rainy @ T.

obs ~ [R+3..R+30] @ T :-
    state=rainy @ T, T > 0, obs=R @ T-1.

?- obs=0 @ 0, obs=4 @ 1, obs=20 @ 2.
```

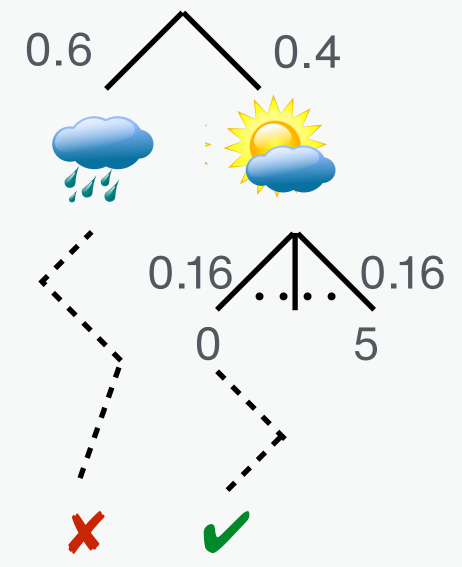
✔

Fusemate Inconsistency Pruning



P() = 0.000119

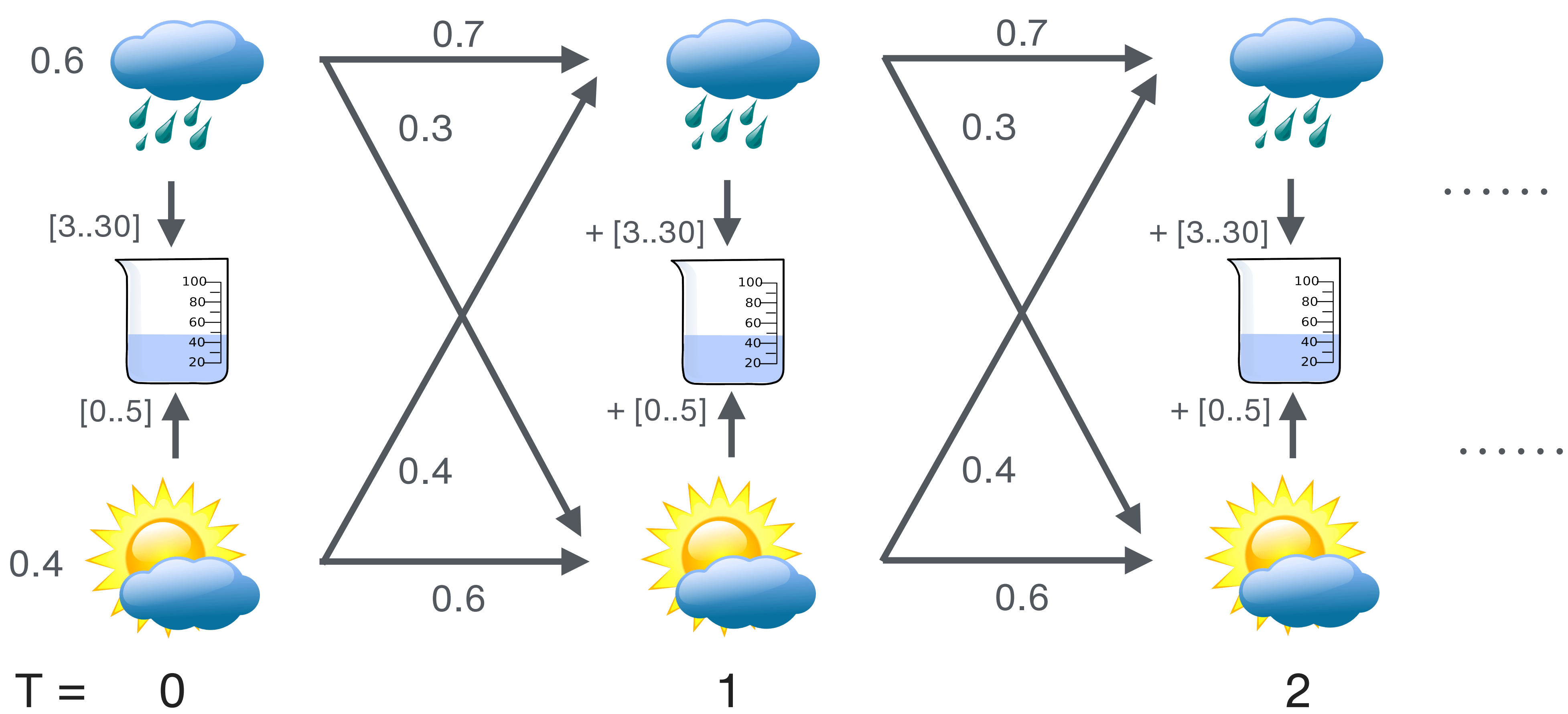
Distribution Semantics

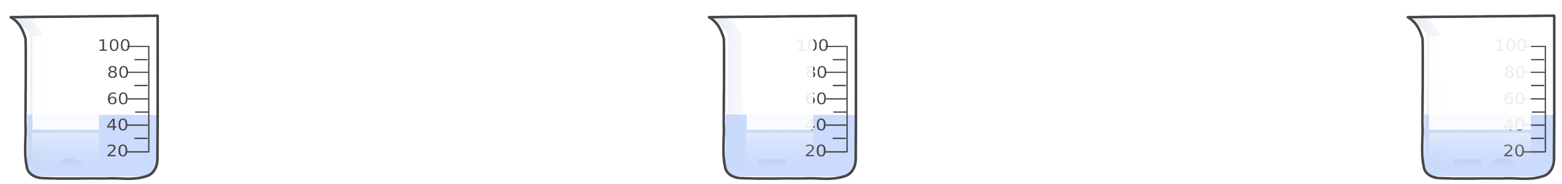


P(query) = $\sum_{✔}$ P(✔)

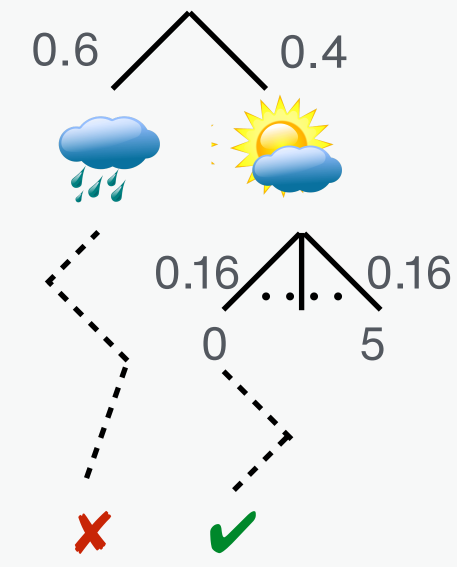
```
state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.

state ~ [[rainy, 0.7], [sunny, 0.3]] @ T+1 :-
    state=rainy @ T.

obs ~ [R+3..R+30] @ T :-
    state=rainy @ T, T > 0, obs=R @ T-1.

?- obs=0 @ 0, obs=4 @ 1, obs=20 @ 2.
```

Fusemate Inconsistency Pruning



P() = 0.000119

Distribution Semantics



P(query) = ∑✔ P(✔)

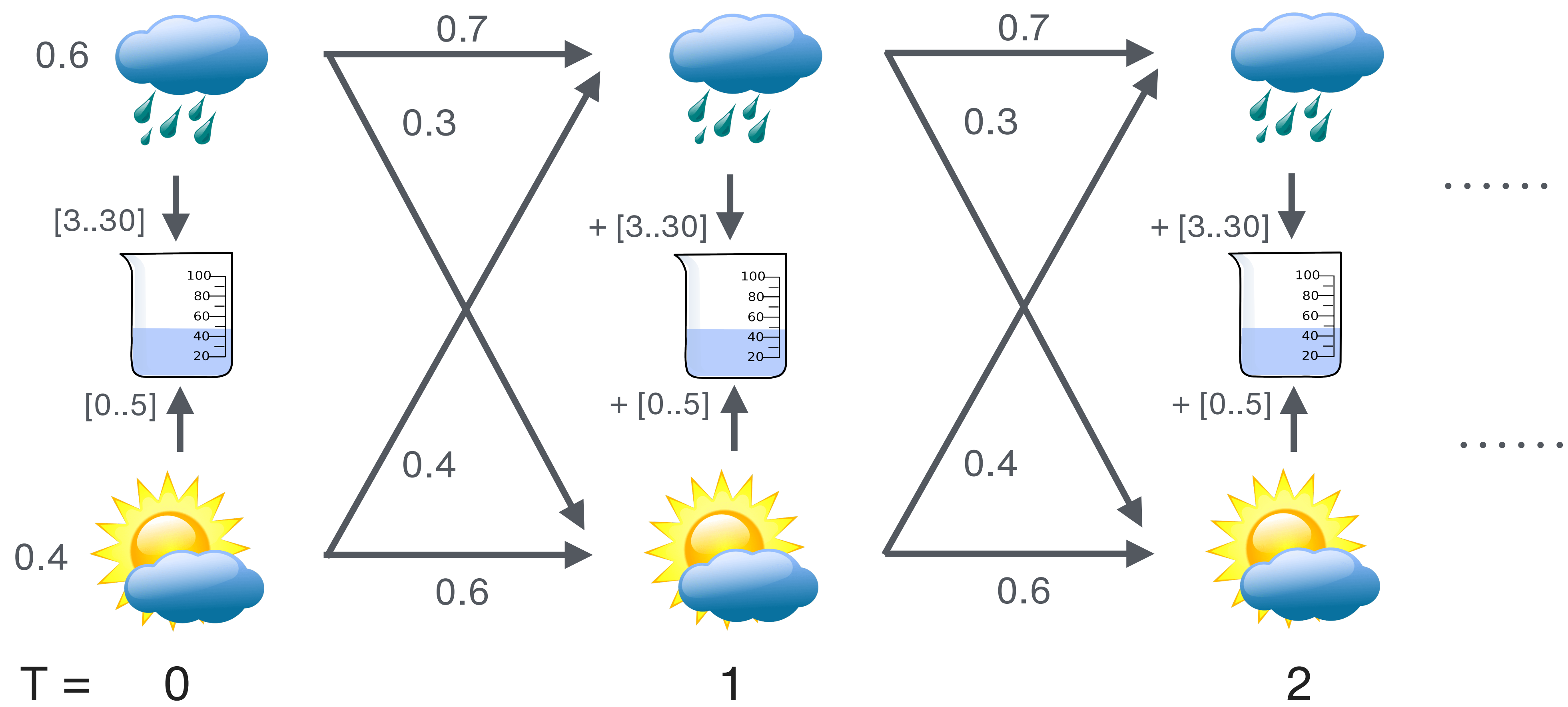
```
state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.

state ~ [[rainy, 0.7], [sunny, 0.3]] @ T+1 :-
    state=rainy @ T.

obs ~ [R+3..R+30] @ T :-
    state=rainy @ T, T > 0, obs=R @ T-1.

?- obs=0 @ 0, obs=4 @ 1, obs=20 @ 2.
```

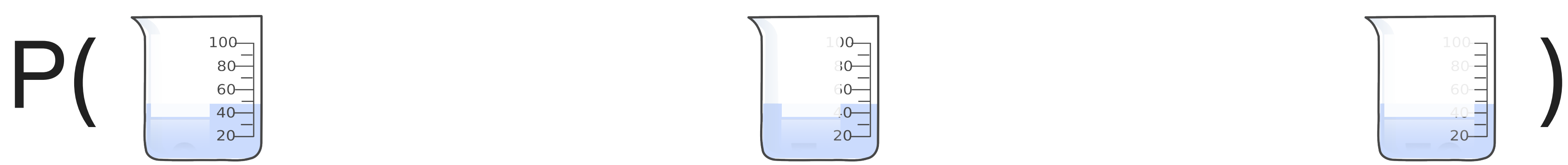
Computing query success probabilities

(1) Program grounding (≈ Bayes net)

(2) Query probability (marginal probability by var. elim.)

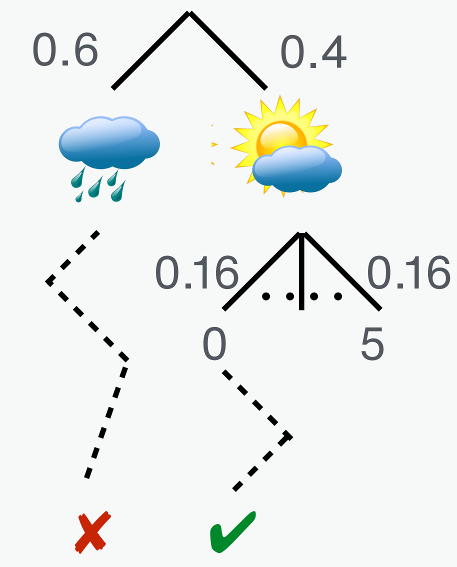
Fusemate Inconsistency Pruning





= 0.000119

Distribution Semantics



P(query) = $\sum_{✔}$ P(✔)

```
state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.

state ~ [[rainy, 0.7], [sunny, 0.3]] @ T+1 :-
    state=rainy @ T.

obs ~ [R+3..R+30] @ T :-
    state=rainy @ T, T > 0, obs=R @ T-1.

?- obs=0 @ 0, obs=4 @ 1, obs=20 @ 2.
```

Computing query success probabilities

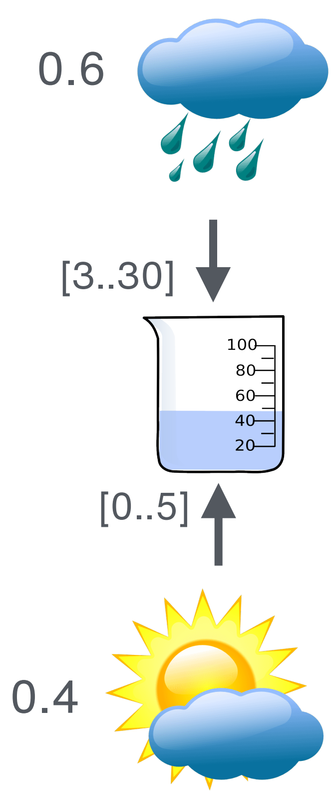
(1) Program grounding (≈ Bayes net)

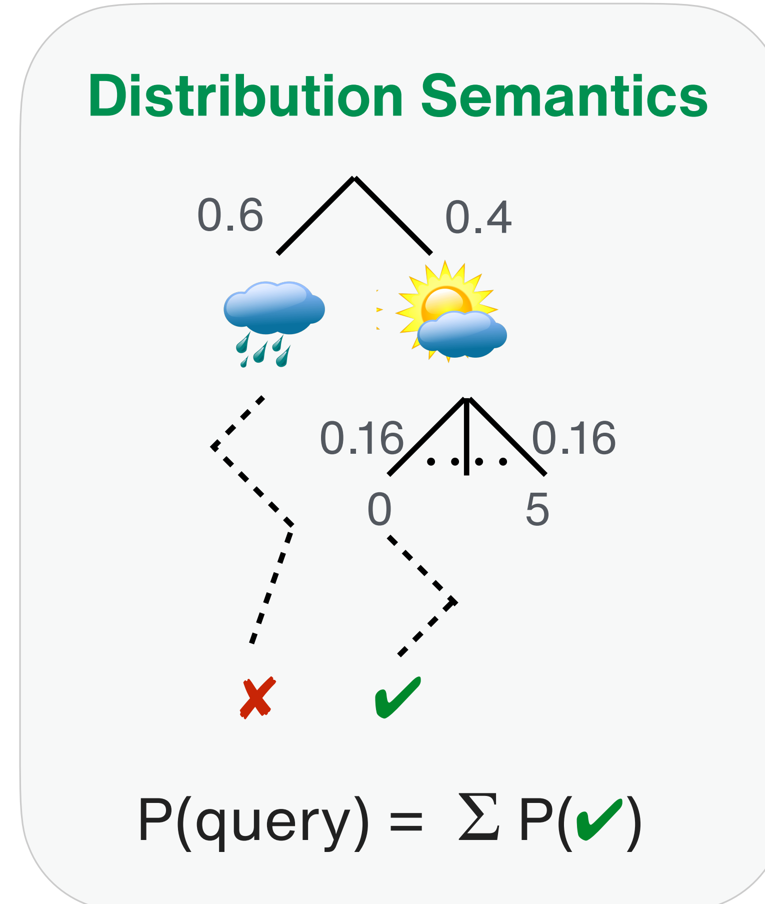
(2) Query probability (marginal probability by var. elim.)

Naive (1): too many rules (quadratic in this case)

Solution: “Inconsistency Pruning”

Efficiency by Inconsistency Pruning



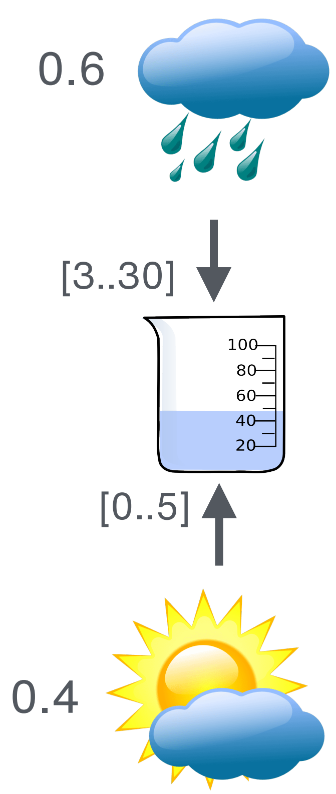


(Already grounded) program rules T = 0

```
state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.
obs ~ [3..30] @ 0 :– state=rainy @ 0.
obs ~ [0..5]  @ 0 :– state=sunny @ 0.
```

```
?– obs=0 @ 0, obs=2 @ 1, obs=20 @ 2.
```

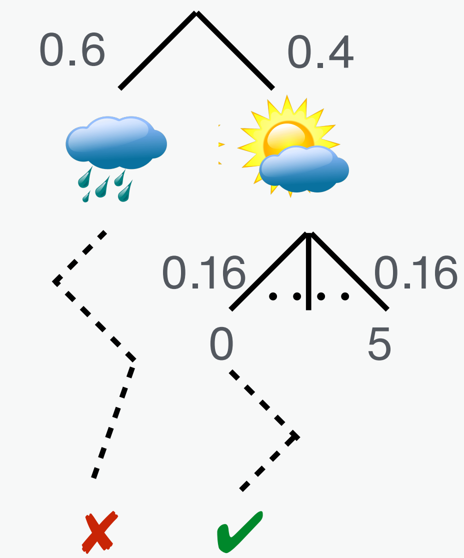
Efficiency by Inconsistency Pruning



In increasing time order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with $\bigcup$ heads

Distribution Semantics



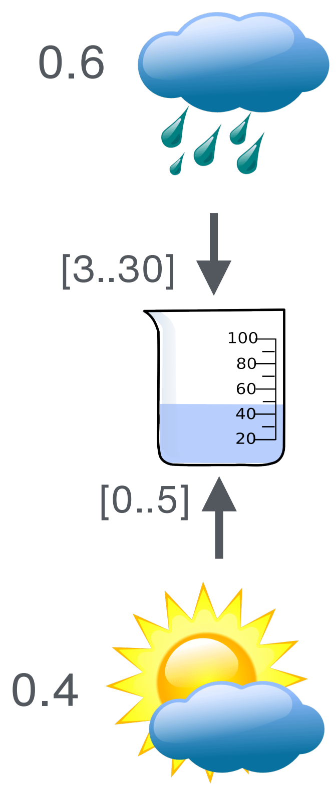
P(query) = Σ P(✔)

(Already grounded) program rules T = 0

```
state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.
obs ~ [3..30] @ 0 :- state=rainy @ 0.
obs ~ [0..5]  @ 0 :- state=sunny @ 0.
```

```
?- obs=0 @ 0, obs=2 @ 1, obs=20 @ 2.
```

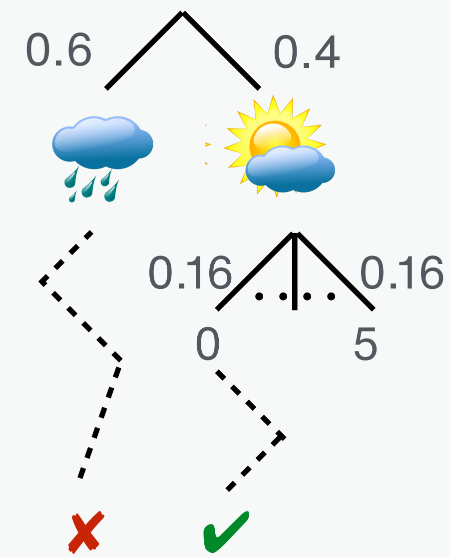
Efficiency by Inconsistency Pruning



In increasing time order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with $\bigcup$ heads

Distribution Semantics



P(query) = Σ P(✔)

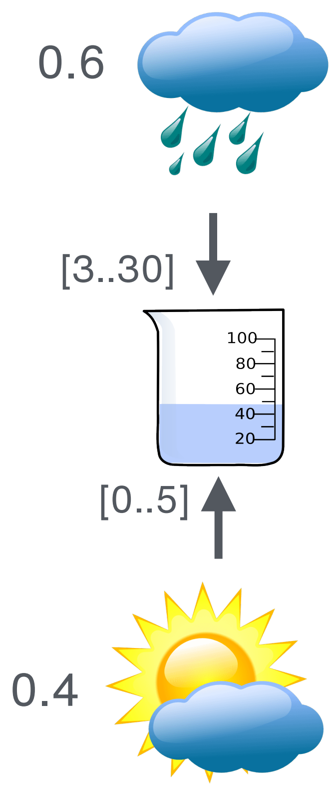
(Already grounded) program rules T = 0

```
state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.
obs ~ [3..30] @ 0 :– state=rainy @ 0.
obs ~ [0..5]  @ 0 :– state=sunny @ 0.
```

Strengthen query by regression

```
?– obs=0 @ 0, obs=2 @ 1, obs=20 @ 2, state=sunny @ 0.
```

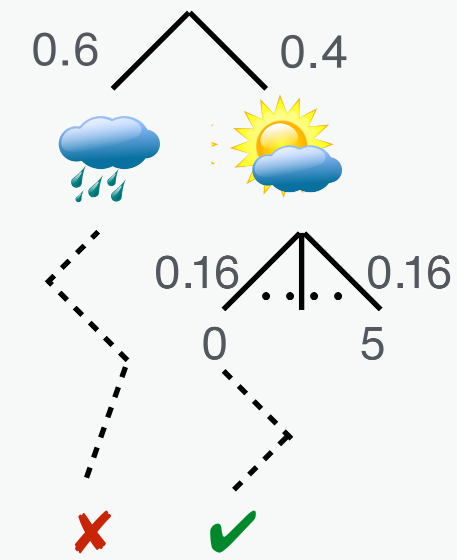
Efficiency by Inconsistency Pruning



In increasing time order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with $\bigcup$ heads

Distribution Semantics



P(query) = Σ P(✔)

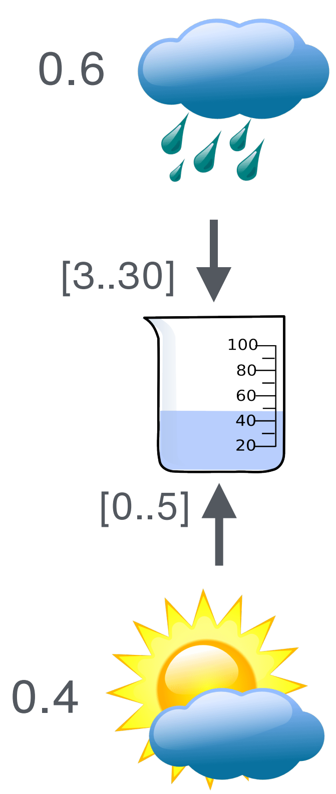
(Already grounded) program rules T = 0

```
state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.
obs ~ [3..30] @ 0 :- state=rainy @ 0.
obs ~ [0..5]  @ 0 :- state=sunny @ 0.
```

Strengthen query by regression

```
?- obs=0 @ 0, obs=2 @ 1, obs=20 @ 2, state=sunny @ 0.
```

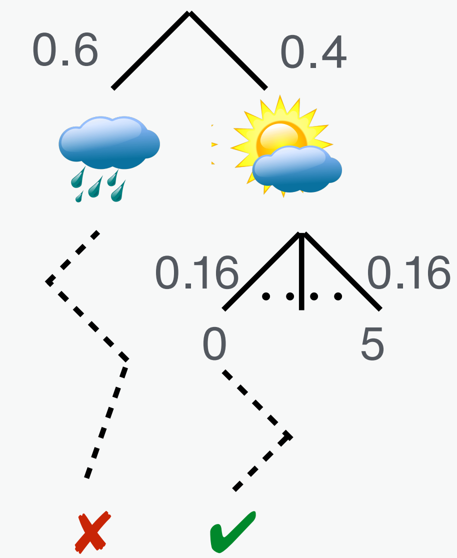
Efficiency by Inconsistency Pruning



In increasing time order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with $\bigcup$ heads

Distribution Semantics



P(query) = Σ P(✔)

(Already grounded) program rules T = 0

```
state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.
```

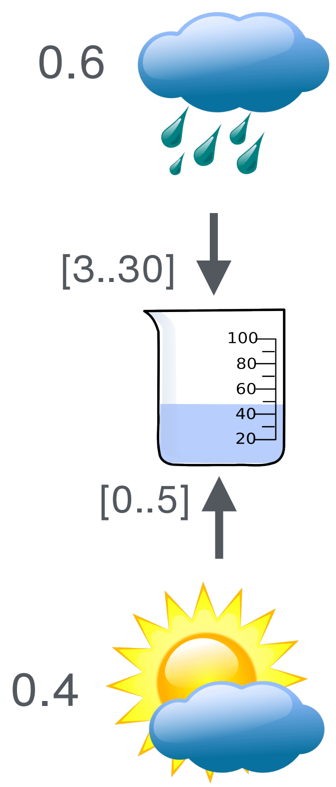
~~obs ~ [3..30] @ 0 :- state=rainy @ 0.~~ IP pruning

```
obs ~ [0..5]  @ 0 :- state=sunny @ 0.
```

Strengthen query by regression

```
?- obs=0 @ 0, obs=2 @ 1, obs=20 @ 2, state=sunny @ 0.
```

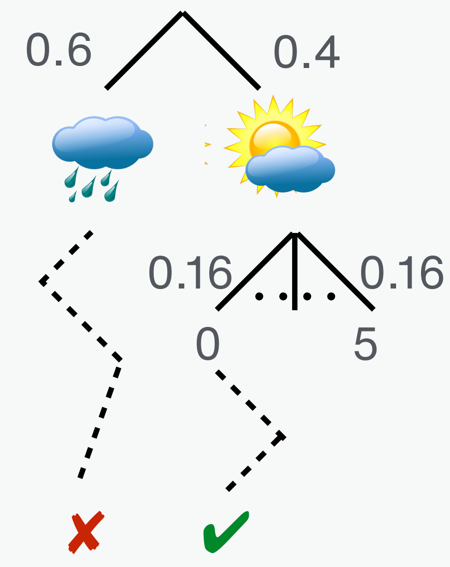
Efficiency by Inconsistency Pruning



In increasing time order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with ⋃ heads

Distribution Semantics



P(query) = Σ P(✔)

(Already grounded) program rules T = 0 ⟶ **Domain after T = 0**

state ~ [[rainy, 0.6], [sunny, 0.4]] @ 0.

~~obs ~ [3..30] @ 0 :- state=rainy @ 0.~~ IP pruning

obs ~ [0..5] @ 0 :- state=sunny @ 0.

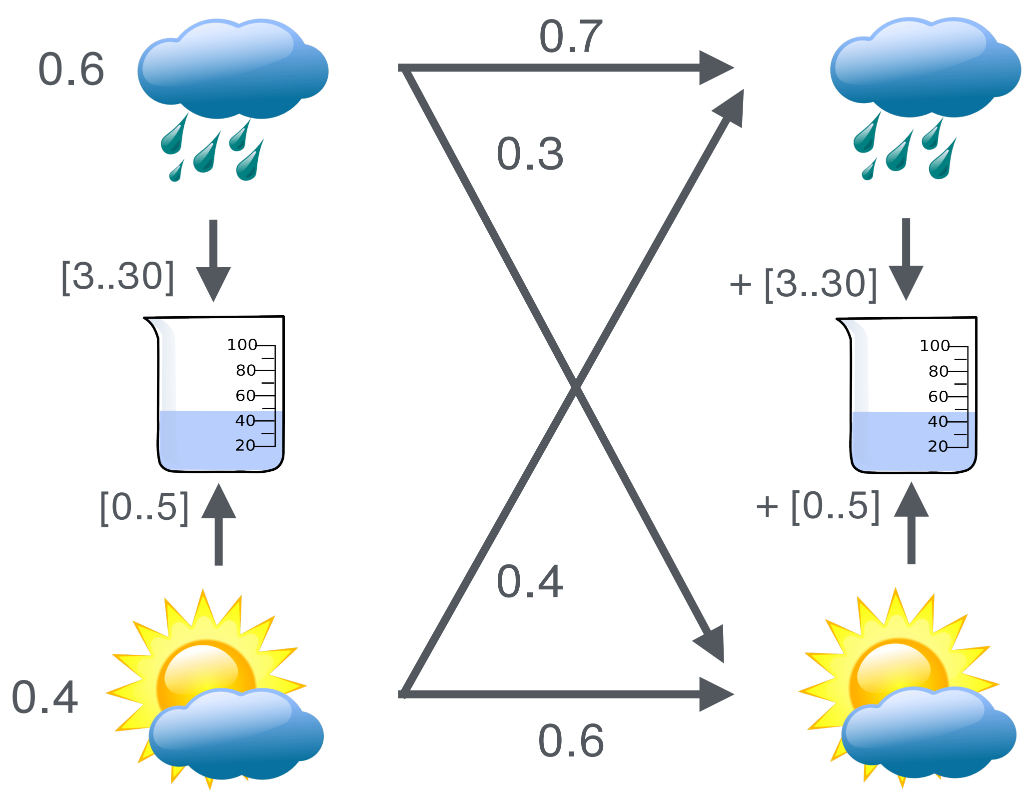
Domain after T = 0:

state = rainy @ 0.
state = sunny @ 0.
obs = 0 @ 0.
obs = 1 @ 0.
:
obs = 5 @ 0.

Strengthen query by regression

?- obs=0 @ 0, obs=2 @ 1, obs=20 @ 2, state=sunny @ 0.

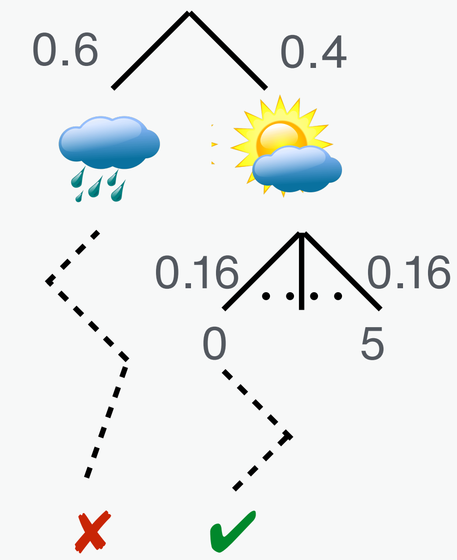
Efficiency by Inconsistency Pruning



In increasing stratification order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with $\bigcup$ heads

Distribution Semantics



P(query) = Σ P(✔)

Domain T = 1

obs = 0 @ 0.

obs = 1 @ 0.

:

obs = 5 @ 0.

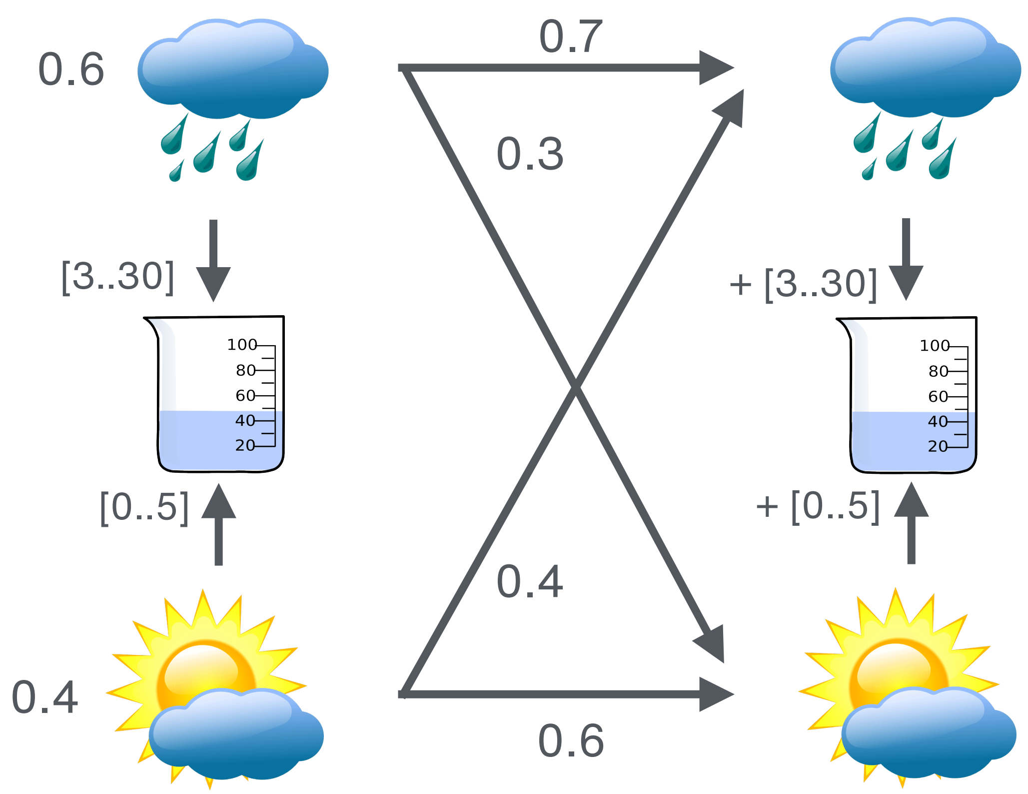
state = rainy @ 1.

state = sunny @ 1.

```
obs ~ [R+3..R+30] @ T :–
    state=rainy @ T,
    T > 0,
    obs=R @ T–1.
```

?– obs=0 @ 0, obs=4 @ 1, obs=20 @ 2, state=sunny @ 0.

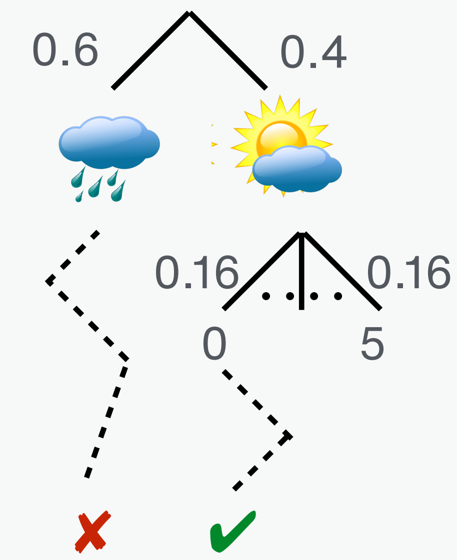
Efficiency by Inconsistency Pruning



In increasing stratification order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with $\bigcup$ heads

Distribution Semantics



P(query) = Σ P(✔)

Domain T = 1 ⟶ Grounded program rules T = 1

obs = 0 @ 0.

obs = 1 @ 0.

:

obs = 5 @ 0.

state = rainy @ 1.

state = sunny @ 1.

obs ~ [3..30] @ 1 :– state=rainy @ 1, obs=0 @ 0.

obs ~ [4..31] @ 1 :– state=rainy @ 1, obs=1 @ 0.

:

obs ~ [0..5] @ 1 :– state=sunny @ 1, obs=0 @ 0.

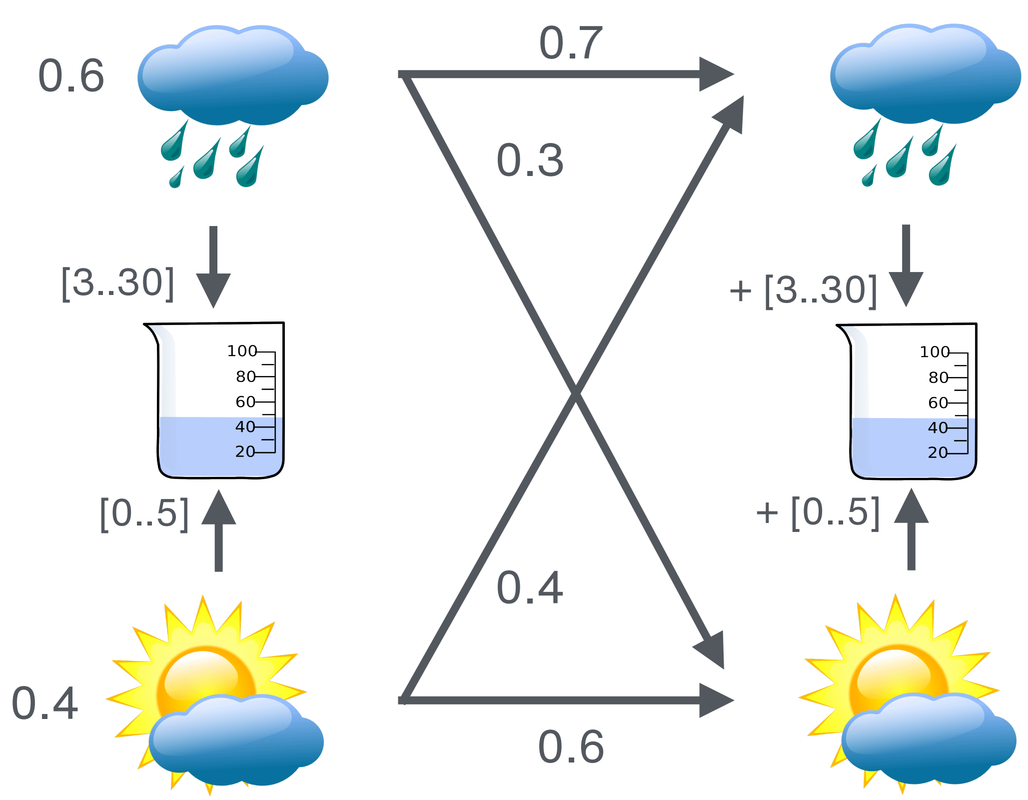
obs ~ [1..6] @ 1 :– state=sunny @ 1, obs=1 @ 0.

:

```
obs ~ [R+3..R+30] @ T :–
    state=rainy @ T,
    T > 0,
    obs=R @ T–1.
```

?– obs=0 @ 0, obs=4 @ 1, obs=20 @ 2, state=sunny @ 0.

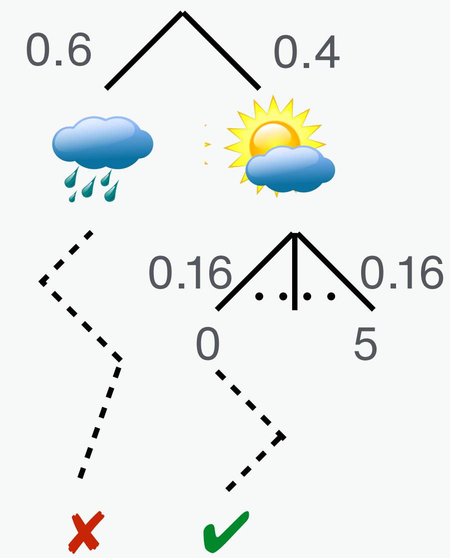
Efficiency by Inconsistency Pruning



In increasing stratification order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with $\bigcup$ heads

Distribution Semantics



P(query) = Σ P(✓)

Domain T = 1 ⟶ Grounded program rules T = 1

obs = 0 @ 0.

obs = 1 @ 0.

:

obs = 5 @ 0.

state = rainy @ 1.

state = sunny @ 1.

obs ~ [3..30] @ 1 :– state=rainy @ 1, obs=0 @ 0.

obs ~ [4..31] @ 1 :– state=rainy @ 1, obs=1 @ 0.

:

obs ~ [0..5] @ 1 :– state=sunny @ 1, obs=0 @ 0.

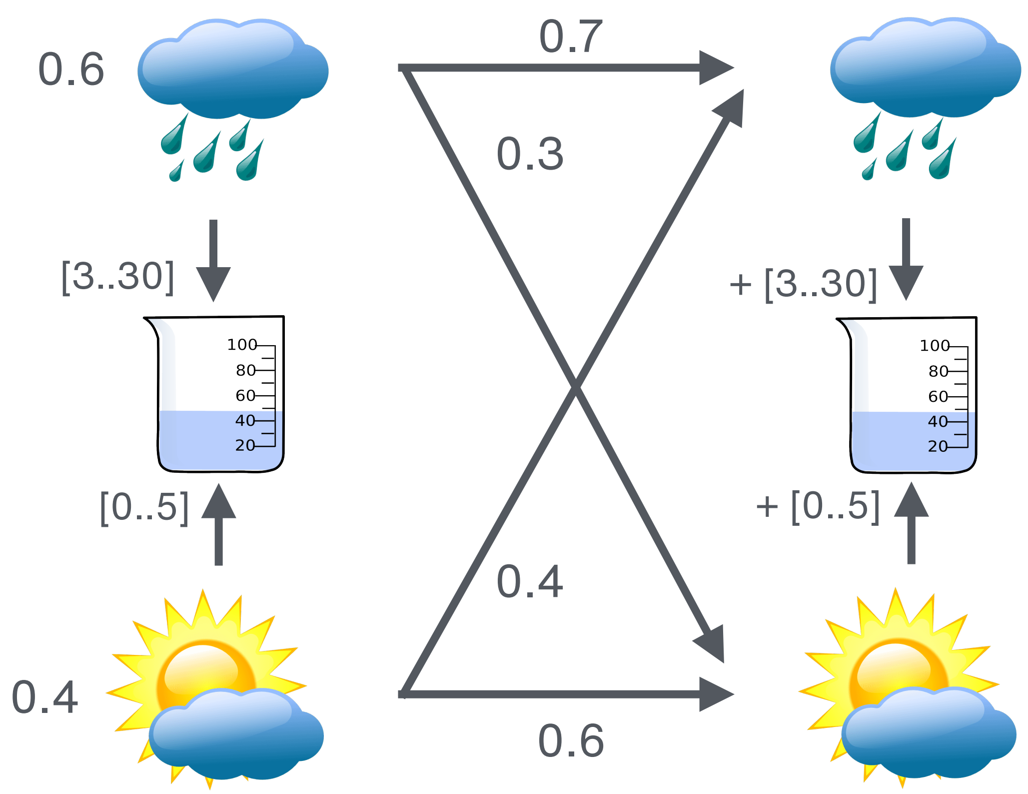
obs ~ [1..6] @ 1 :– state=sunny @ 1, obs=1 @ 0.

:

obs ~ [R+3..R+30] @ T :–
  state=rainy @ T,
  T > 0,
  obs=R @ T–1.

?– obs=0 @ 0, obs=4 @ 1, obs=20 @ 2, state=sunny @ 0.

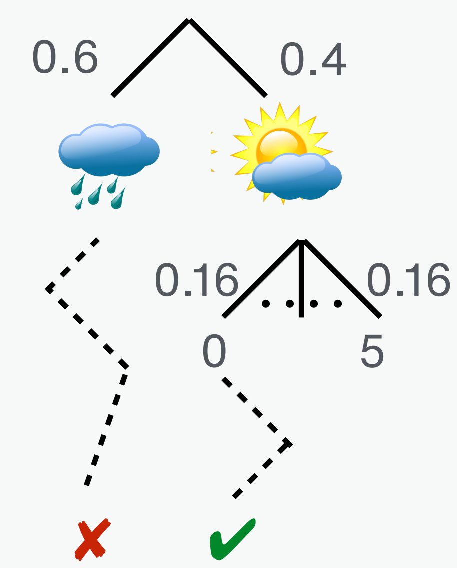
Efficiency by Inconsistency Pruning



In increasing stratification order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with $\bigcup$ heads

Distribution Semantics



P(query) = Σ P(✔)

Domain T = 1 → Grounded program rules T = 1

obs = 0 @ 0.

obs = 1 @ 0.

:

obs = 5 @ 0.

state = rainy @ 1.

state = sunny @ 1.

obs ~ [3..30] @ 1 :– state=rainy @ 1, obs=0 @ 0.

obs ~ [4..31] @ 1 :– state=rainy @ 1, obs=1 @ 0.

:

obs ~ [0..5] @ 1 :– state=sunny @ 1, obs=0 @ 0.

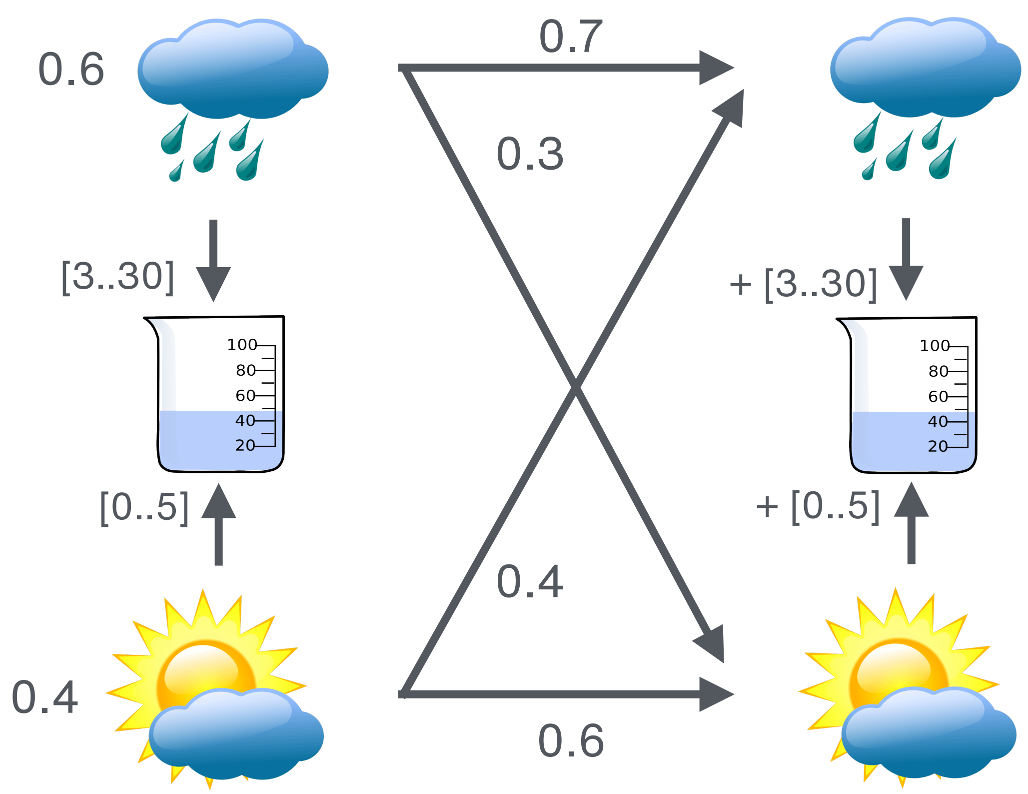
obs ~ [1..6] @ 1 :– state=sunny @ 1, obs=1 @ 0.

:

obs ~ [R+3..R+30] @ T :–
state=rainy @ T,
T > 0,
obs=R @ T–1.

?– obs=0 @ 0, obs=4 @ 1, obs=20 @ 2, state=sunny @ 0.

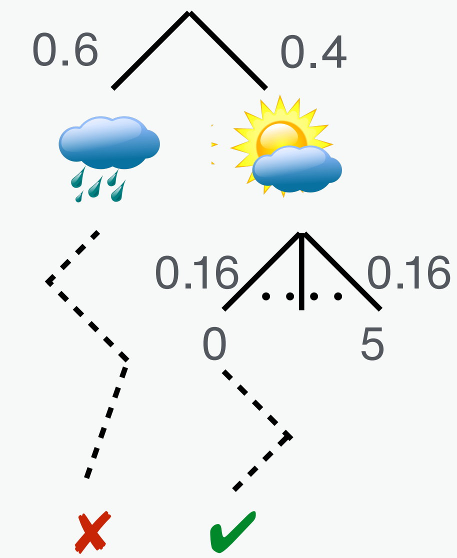
Efficiency by Inconsistency Pruning



In increasing stratification order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with ⋃ heads

Distribution Semantics



P(query) = Σ P(✔)

Domain T = 1 ⟶ **Grounded program rules T = 1**

obs = 0 @ 0.

obs = 1 @ 0.

:

obs = 5 @ 0.

state = rainy @ 1.

state = sunny @ 1.

obs ~ [3..30] @ 1 :– state=rainy @ 1, obs=0 @ 0.

IP ~~obs ~ [4..31] @ 1 :– state=rainy @ 1, obs=1 @ 0.~~

:

obs ~ [0..5] @ 1 :– state=sunny @ 1, obs=0 @ 0.

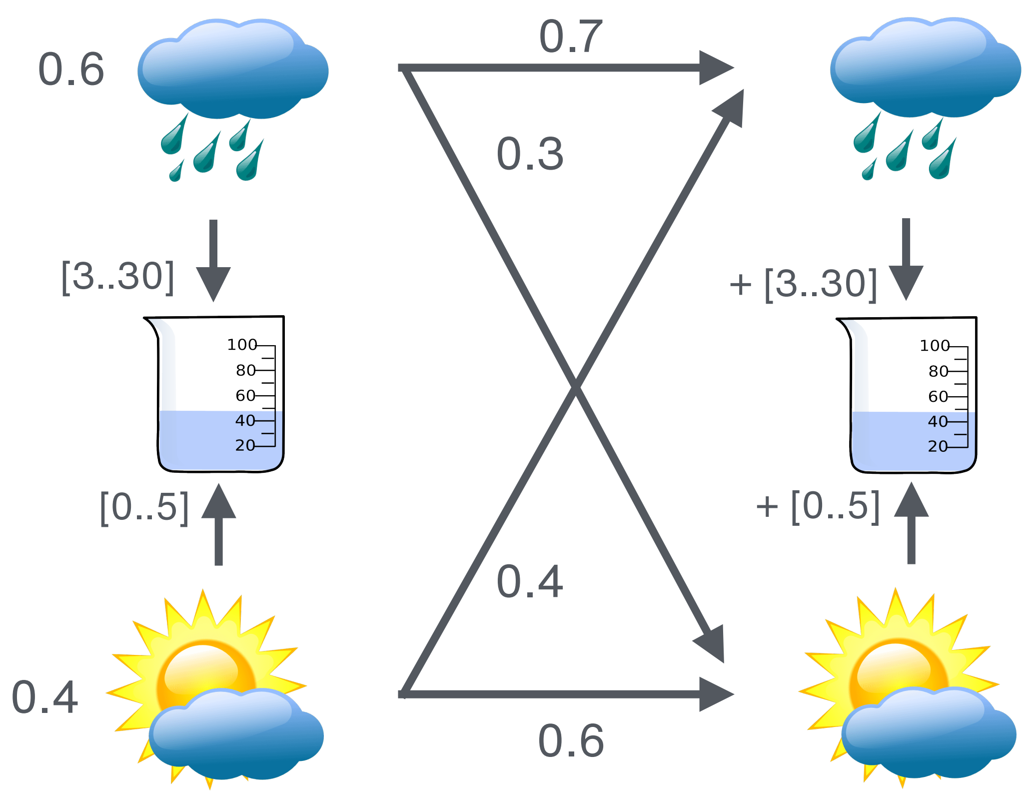
IP ~~obs ~ [1..6] @ 1 :– state=sunny @ 1, obs=1 @ 0.~~

:

obs ~ [R+3..R+30] @ T :–
state=rainy @ T,
T > 0,
obs=R @ T–1.

?– obs=0 @ 0, obs=4 @ 1, obs=20 @ 2, state=sunny @ 0.

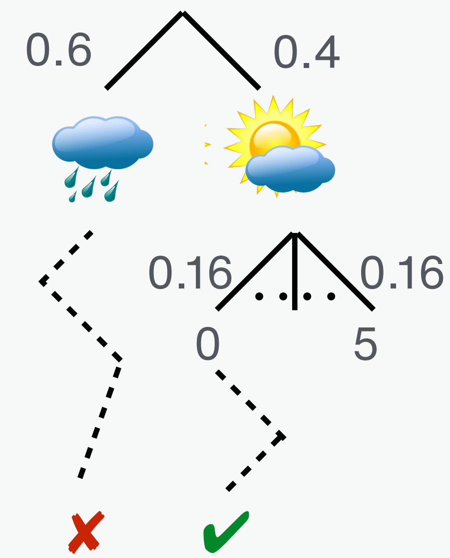
Efficiency by Inconsistency Pruning



In increasing stratification order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with ⋃ heads

Distribution Semantics



P(query) = Σ P(✔)

Domain T = 1 ⟶ **Grounded program rules T = 1**

obs = 0 @ 0.

obs = 1 @ 0.

:

obs = 5 @ 0.

state = rainy @ 1.

state = sunny @ 1.

obs ~ [3..30] @ 1 :– state=rainy @ 1, obs=0 @ 0.

IP ~~obs ~ [4..31] @ 1 :– state=rainy @ 1, obs=1 @ 0.~~

:

obs ~ [0..5] @ 1 :– state=sunny @ 1, obs=0 @ 0.

IP ~~obs ~ [1..6] @ 1 :– state=sunny @ 1, obs=1 @ 0.~~

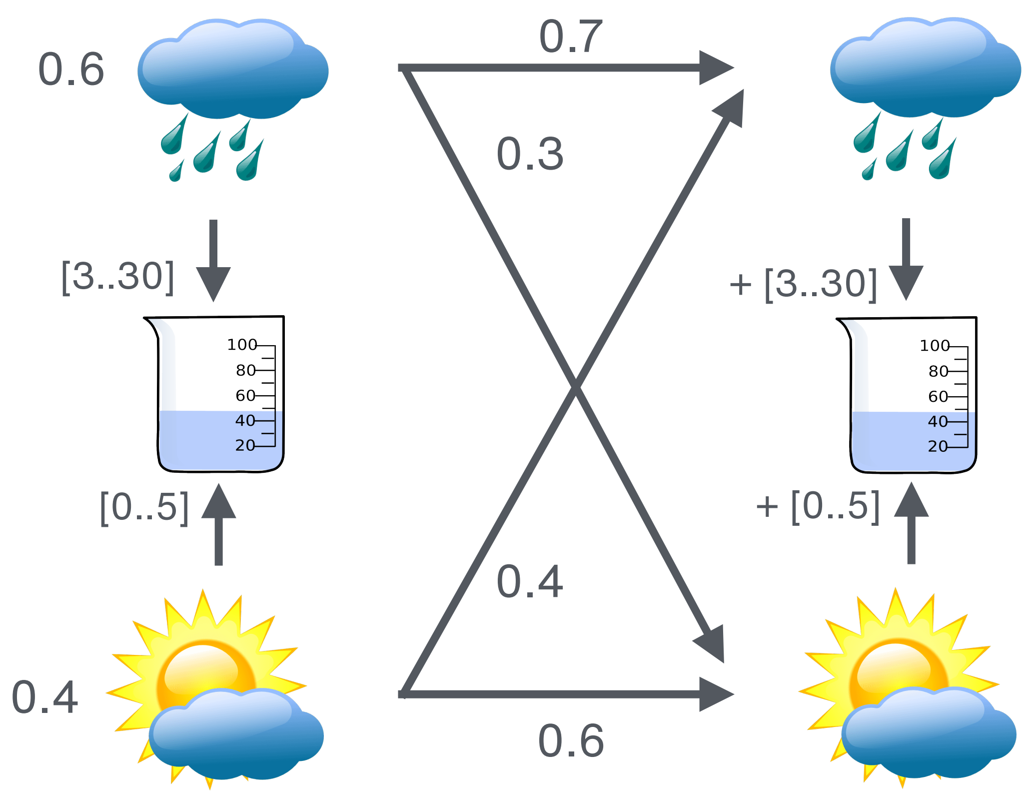
:

obs ~ [R+3..R+30] @ T :–
state=rainy @ T,
T > 0,
obs=R @ T–1.

Inconsistency pruning: 62 -> 2 rules

?– obs=0 @ 0, obs=4 @ 1, obs=20 @ 2, state=sunny @ 0.

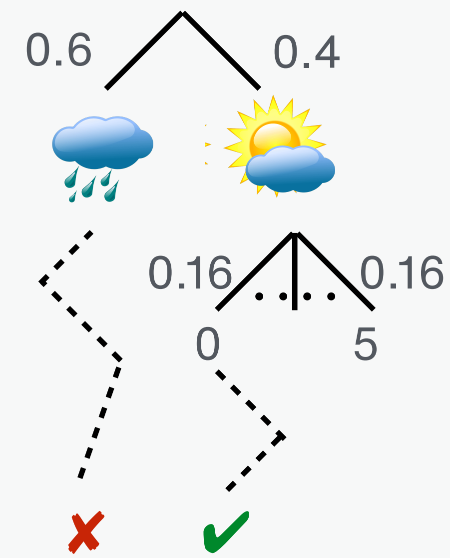
Efficiency by Inconsistency Pruning



In increasing stratification order:

- Ground out program over current domain
- Query regression, inconsistency pruning
- Extend current domain with ⋃ heads

Distribution Semantics



P(query) = Σ P(✔)

Domain T = 1 ⟶ **Grounded program rules T = 1**

obs = 0 @ 0.

obs = 1 @ 0.

:

obs = 5 @ 0.

state = rainy @ 1.

state = sunny @ 1.

obs ~ [3..30] @ 1 :– state=rainy @ 1, obs=0 @ 0.

IP ~~obs ~ [4..31] @ 1 :– state=rainy @ 1, obs=1 @ 0.~~

:

obs ~ [0..5] @ 1 :– state=sunny @ 1, obs=0 @ 0.

IP ~~obs ~ [1..6] @ 1 :– state=sunny @ 1, obs=1 @ 0.~~

:

obs ~ [R+3..R+30] @ T :–
state=rainy @ T,
T > 0,
obs=R @ T–1.

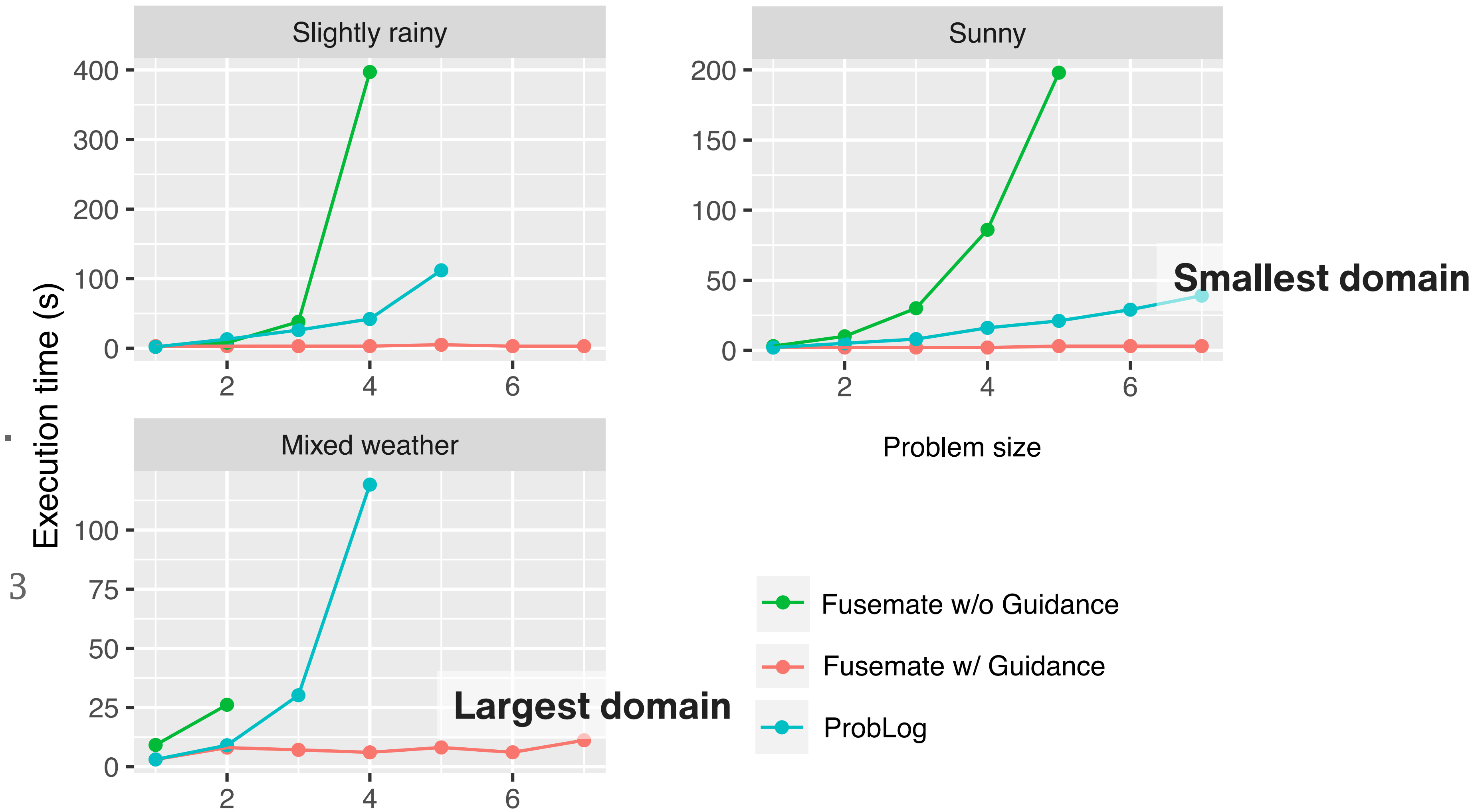
Inconsistency pruning: 62 -> 2 rules

?– obs=0 @ 0, obs=4 @ 1, obs=20 @ 2, state=sunny @ 0.

Experimental Evaluation 1 - Hidden Markov Model

Runtime Results Fusemate vs ProbLog

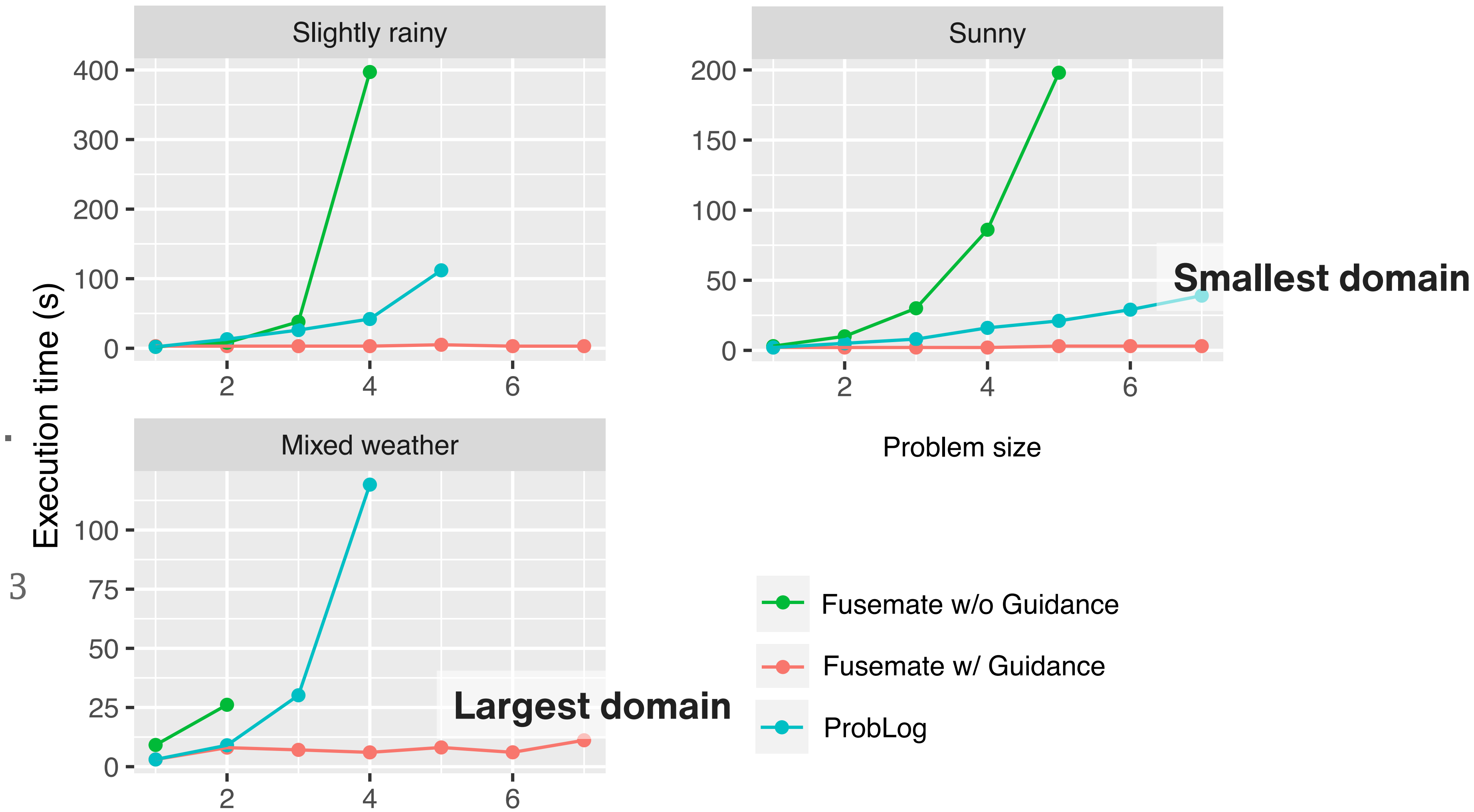
```
Rainy/sunny example from above

%% Queries for N=3
%% Sunny
?-state=X @ 3 | obs=0 @ 1, obs=0 @ 2, obs=0 @ 3.

%% Rainy
?-state=X @ 3 | obs=4 @ 1, obs=8 @ 2, obs=12 @ 3

%% Mixed
state=X @ 3 | obs=0 @ 1, obs=4 @ 2, obs=24 @ 3.
```



Experimental Evaluation 1 - Hidden Markov Model

Runtime Results Fusemate vs ProbLog

```
Rainy/sunny example from above

%% Queries for N=3
%% Sunny
?-state=X @ 3 | obs=0 @ 1, obs=0 @ 2, obs=0 @ 3.

%% Rainy
?-state=X @ 3 | obs=4 @ 1, obs=8 @ 2, obs=12 @ 3

%% Mixed
state=X @ 3 | obs=0 @ 1, obs=4 @ 2, obs=24 @ 3.
```



Grounding vs Inference - Mixed Weather

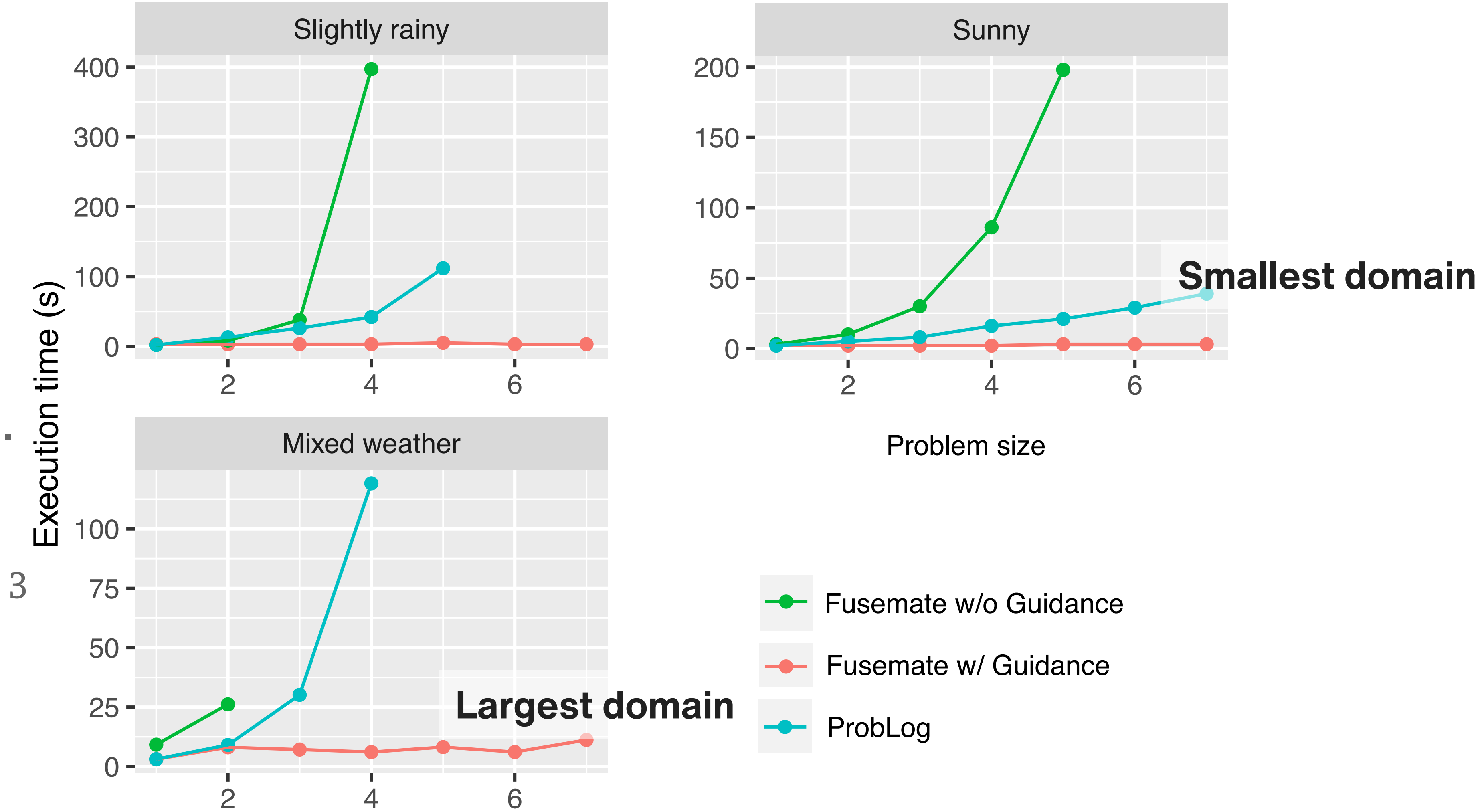
	Fusemate #ground rules		ProbLog		
N	query-guided	unguided	total time	grounding time	#ground rules
2	2200	6500	9.0	8.3	53
3	2270	12900	30	19	276
4	2300	21400	119	33	499
5	2400	32000		50	682
6	2470	45000		65	839
7	2500	60000		95	1068

Experimental Evaluation 1 - Hidden Markov Model

Runtime Results Fusemate vs ProbLog

Rainy/sunny example from above

```
%% Queries for N=3
%% Sunny
?-state=X @ 3 | obs=0 @ 1, obs=0 @ 2, obs=0 @ 3.

%% Rainy
?-state=X @ 3 | obs=4 @ 1, obs=8 @ 2, obs=12 @ 3

%% Mixed
state=X @ 3 | obs=0 @ 1, obs=4 @ 2, obs=24 @ 3.
```



Grounding vs Inference - Mixed Weather

	Fusemate #ground rules		ProbLog		
N	query-guided	unguided	total time	grounding time	#ground rules
2	2200	6500	9.0	8.3	53
3	2270	12900	30	19	276
4	2300	21400	119	33	499
5	2400	32000		50	682
6	2470	45000		65	839
7	2500	60000		95	1068

Fusemate:

Improved grounding pays off

Inference engine implements UNA

ProbLog:

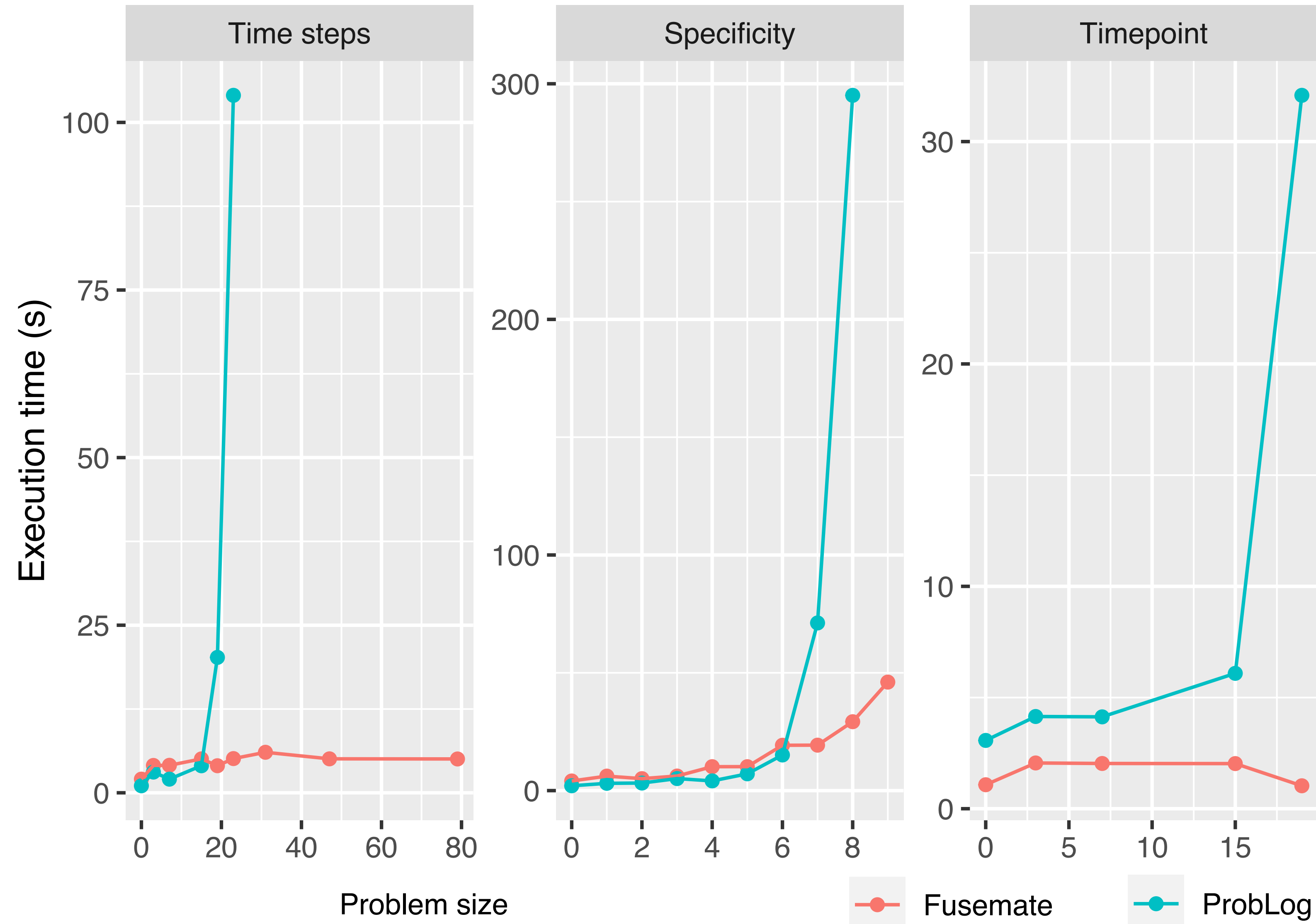
Grounding OK?

Bottleneck inference component?

Experimental Evaluation 2 - Markov Model

Runtime Results Fusemate vs ProbLog

```
%% Markov Model
in ~ [a, b, c] @ 0.
in ~ [[a, 0.9],[b, 0.05],
     [c, 0.05]] @ T+1 :- in=a @ T.
in ~ [[a, 0.7],[c, 0.3]] @ T+1 :- in=b @ T.
in ~ [[a, 0.8],[c, 0.2]] @ T+1 :- in=c @ T.

%% Time steps N = 20
?- in=a@0, in=a@1,.., in=a@20.

%% Specificity, N = 7
?- in=a@0, in=a@1, in=L2@2,..,in=L8 @ 8.

%% Timepoint, N = 20
?- in=a@23.
```



(ProbLog code from ProbLog tutorial web page)

Learning (Largely TBD in Fusemate)

Probability parameters learning

MLE, EM

Learning the structure of logic programs

Inductive Logic Programming (1970s)

Probabilistic Version [Riguzzi 2015]

Logic programs from tabular data

Probabilistic version of CART

Probabilistic decision lists [2017]

FOLD-RM [Gupta et al, ICLP 2023]

CON-FOLD [McGinness and B, ICLP 2024]

= FOLD-RM with confidence values

Very short explanations

	PassengerId	Survived	Pclass	Title	Sex	Age	SibSp	Parch	
0	1	False	3	Mr	male	22	1	0	
1	2	True	1	Mrs	female	38	1	0	
2	3	True	3	Miss	female	26	0	0	STON/C
3	4	True	1	Mrs	female	35	1	0	
4	5	False	3	Mr	male	NaN	0	0	

```
survived(X) :- not perished(X).
perished(X) :- not sex(X, female).
perished(X) :-
    sex(X, female), pclass(X, 3),
    fare(X, N), not N <= 23.25.
```

Learning (Largely TBD in Fusemate)

Probability parameters learning

MLE, EM

Learning the structure of logic programs

Inductive Logic Programming (1970s)

Probabilistic Version [Riguzzi 2015]

Logic programs from tabular data

Probabilistic version of CART

Probabilistic decision lists [2017]

FOLD-RM [Gupta et al, ICLP 2023]

CON-FOLD [McGinness and B, ICLP 2024]

= FOLD-RM with confidence values

Very short explanations

Conditions for Survival?

	PassengerId	Survived	Pclass	Title	Sex	Age	SibSp	Parch	
0	1	False	3	Mr	male	22	1	0	
1	2	True	1	Mrs	female	38	1	0	
2	3	True	3	Miss	female	26	0	0	STON/C
3	4	True	1	Mrs	female	35	1	0	
4	5	False	3	Mr	male	NaN	0	0	

```
survived(X) :- not perished(X).
perished(X) :- not sex(X, female).
perished(X) :-
    sex(X, female), pclass(X, 3),
    fare(X, N), not N <= 23.25.
```

Learning (Largely TBD in Fusemate)

Probability parameters learning

MLE, EM

Learning the structure of logic programs

Inductive Logic Programming (1970s)

Probabilistic Version [Riguzzi 2015]

Logic programs from tabular data

Probabilistic version of CART

Probabilistic decision lists [2017]

FOLD-RM [Gupta et al, ICLP 2023]

CON-FOLD [McGinness and B, ICLP 2024]

= FOLD-RM with confidence values

Very short explanations

Conditions for Survival?

	PassengerId	Survived	Pclass	Title	Sex	Age	SibSp	Parch	
0	1	False	3	Mr	male	22	1	0	
1	2	True	1	Mrs	female	38	1	0	
2	3	True	3	Miss	female	26	0	0	STON/C
3	4	True	1	Mrs	female	35	1	0	
4	5	False	3	Mr	male	NaN	0	0	

survived(X) :– not perished(X).

perished(X) :– not sex(X, female).

perished(X) :–

sex(X, female), pclass(X, 3),

fare(X, N), not N <= 23.25.

Learning (Largely TBD in Fusemate)

Probability parameters learning

MLE, EM

Learning the structure of logic programs

Inductive Logic Programming (1970s)

Probabilistic Version [Riguzzi 2015]

Logic programs from tabular data

Probabilistic version of CART

Probabilistic decision lists [2017]

FOLD-RM [Gupta et al, ICLP 2023]

CON-FOLD [McGinness and B, ICLP 2024]

= FOLD-RM with confidence values

Very short explanations

Conditions for Survival?

	PassengerId	Survived	Pclass	Title	Sex	Age	SibSp	Parch	
0	1	False	3	Mr	male	22	1	0	
1	2	True	1	Mrs	female	38	1	0	
2	3	True	3	Miss	female	26	0	0	STON/C
3	4	True	1	Mrs	female	35	1	0	
4	5	False	3	Mr	male	NaN	0	0	

```
0.97 survived(X) :- not perished(X).
     perished(X) :- not sex(X, female).
     perished(X) :-
         sex(X, female), pclass(X, 3),
         fare(X, N), not N <= 23.25.
```

Learning (Largely TBD in Fusemate)

Probability parameters learning

MLE, EM

Learning the structure of logic programs

Inductive Logic Programming (1970s)

Probabilistic Version [Riguzzi 2015]

Logic programs from tabular data

Probabilistic version of CART

Probabilistic decision lists [2017]

FOLD-RM [Gupta et al, ICLP 2023]

CON-FOLD [McGinness and B, ICLP 2024]

= FOLD-RM with confidence values

Very short explanations

Conditions for Survival?

	PassengerId	Survived	Pclass	Title	Sex	Age	SibSp	Parch	
0	1	False	3	Mr	male	22	1	0	
1	2	True	1	Mrs	female	38	1	0	
2	3	True	3	Miss	female	26	0	0	STON/C
3	4	True	1	Mrs	female	35	1	0	
4	5	False	3	Mr	male	NaN	0	0	

~~0.97~~ survived(X) :– not perished(X).

0.9 perished(X) :– not sex(X, female).

perished(X) :–

sex(X, female), pclass(X, 3),

~~fare(X, N), not N <= 23.25.~~

Part 1

- Probabilistic
- Logic
- Programming
- Fusemate Implementation

Part 2

- LLMs + Logic (Programming)
- Neural Networks + Logic (Programming)

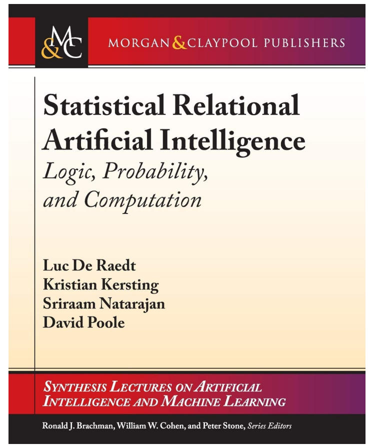
Statistics/NN/LLM+ Logic Combinations

StarAI =

RelationalAI/Logic +

Learning + Statistics (1980s)

Fusemate



NeSy =

Neural Networks + Symbolic Reasoning

Neural-Symbolic Learning and Reasoning:
A Survey and Interpretation

Tarek R. Besold et al TAREK-R.BESOLD@CITY.AC.UK
Department of Computer Science, City, University of London

NeSy + StarAI ?

From Statistical Relational to Neural Symbolic Artificial Intelligence: a Survey.

Giuseppe Marra[a], Sebastijan Dumančić[c], Robin Manhaeve[a], Luc De Raedt[a,b]

[a] *KU Leuven, Department of Computer Science and Leuven.AI*
[b] *Örebro University, Center for Applied Autonomous Sensor Systems*
[c] *Delft University of Technology, Department of Software Technology*

DeepProbLog - see below

LLMs + Logic

Augmented Language Models: a Survey

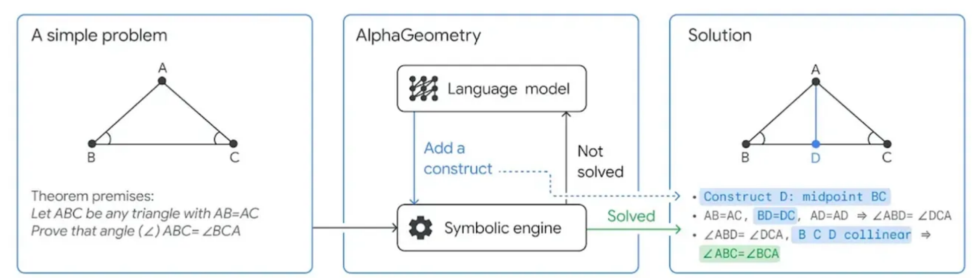
Grégoire Mialon* et al gmialon@meta.com

See below

Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks

Subbarao Kambhampati[1] Karthik Valmeekam[1] Lin Guan[1] Mudit Verma[1] Kaya Stechly[1] Siddhant Bhambri[1] Lucas Saldyt[1] Anil Murthy[1]

AlphaZero -> AlphaGeometry, AlphaProof



Lachlan's PhD - "AlphaPhysics"

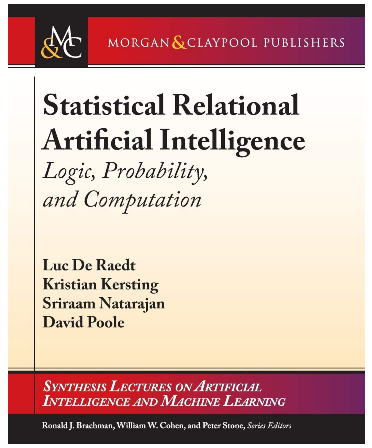
Statistics/NN/LLM+ Logic Combinations

StarAI =

RelationalAI/Logic +

Learning + Statistics (1980s)

Fusemate



NeSy =

Neural Networks + Symbolic Reasoning

Neural-Symbolic Learning and Reasoning:
A Survey and Interpretation

Tarek R. Besold et al TAREK-R.BESOLD@CITY.AC.UK
Department of Computer Science, City, University of London

NeSy + StarAI ?

From Statistical Relational to Neural Symbolic Artificial Intelligence: a Survey.

Giuseppe Marra[a], Sebastijan Dumančić[c], Robin Manhaeve[a], Luc De Raedt[a,b]

[a] *KU Leuven, Department of Computer Science and Leuven.AI*
[b] *Örebro University, Center for Applied Autonomous Sensor Systems*
[c] *Delft University of Technology, Department of Software Technology*

DeepProbLog - see below

LLMs + Logic

Augmented Language Models: a Survey

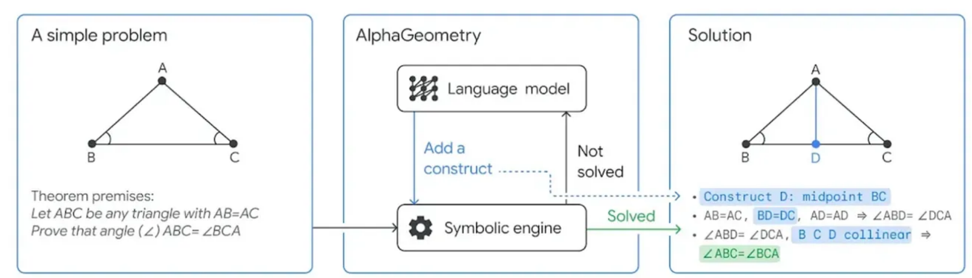
Grégoire Mialon* et al *gmialon@meta.com*

See below

Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks

Subbarao Kambhampati[1] Karthik Valmeekam[1] Lin Guan[1] Mudit Verma[1] Kaya Stechly[1]
Siddhant Bhambri[1] Lucas Saldyt[1] Anil Murthy[1]

AlphaZero -> AlphaGeometry, AlphaProof



Lachlan's PhD - "AlphaPhysics"

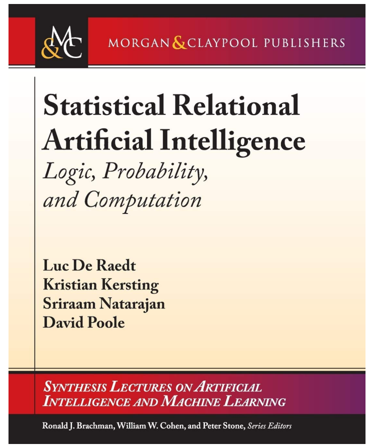
Statistics/NN/LLM+ Logic Combinations

StarAI =

RelationalAI/Logic +

Learning + Statistics (1980s)

Fusemate



NeSy =

Neural Networks + Symbolic Reasoning

Neural-Symbolic Learning and Reasoning:
A Survey and Interpretation

Tarek R. Besold et al TAREK-R.BESOLD@CITY.AC.UK
Department of Computer Science, City, University of London

NeSy + StarAI ?

From Statistical Relational to Neural Symbolic Artificial Intelligence: a Survey.

Giuseppe Marra[a], Sebastijan Dumančić[c], Robin Manhaeve[a], Luc De Raedt[a,b]

[a] *KU Leuven, Department of Computer Science and Leuven.AI*
[b] *Örebro University, Center for Applied Autonomous Sensor Systems*
[c] *Delft University of Technology, Department of Software Technology*

DeepProbLog - see below

LLMs + Logic

Augmented Language Models: a Survey

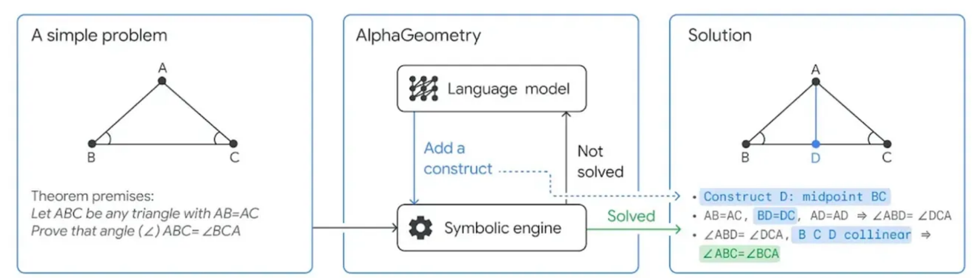
Grégoire Mialon* et al *gmialon@meta.com*

See below

Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks

Subbarao Kambhampati [1] Karthik Valmeekam [1] Lin Guan [1] Mudit Verma [1] Kaya Stechly [1] Siddhant Bhambri [1] Lucas Saldyt [1] Anil Murthy [1]

AlphaZero -> AlphaGeometry, AlphaProof



Lachlan's PhD - "AlphaPhysics"

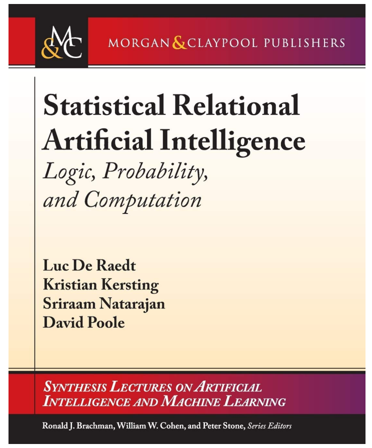
Statistics/NN/LLM+ Logic Combinations

StarAI =

RelationalAI/Logic +

Learning + Statistics (1980s)

Fusemate



NeSy =

Neural Networks + Symbolic Reasoning

Neural-Symbolic Learning and Reasoning:
A Survey and Interpretation

Tarek R. Besold et al TAREK-R.BESOLD@CITY.AC.UK
Department of Computer Science, City, University of London

NeSy + StarAI ?

From Statistical Relational to Neural Symbolic Artificial Intelligence: a Survey.

Giuseppe Marra[a], Sebastijan Dumančić[c], Robin Manhaeve[a], Luc De Raedt[a,b]

[a] *KU Leuven, Department of Computer Science and Leuven.AI*
[b] *Örebro University, Center for Applied Autonomous Sensor Systems*
[c] *Delft University of Technology, Department of Software Technology*

DeepProbLog - see below

LLMs + Logic

Augmented Language Models: a Survey

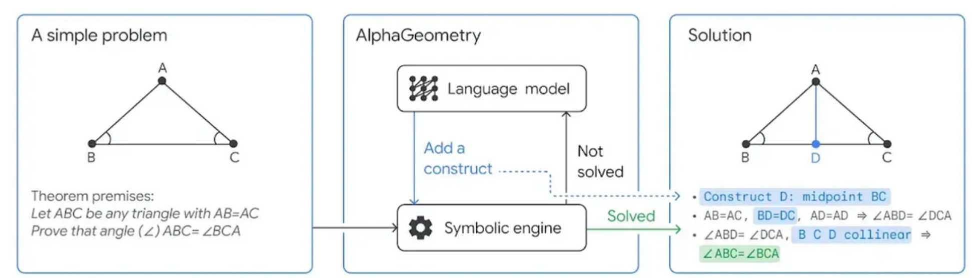
Grégoire Mialon* et al *gmialon@meta.com*

See below

Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks

Subbarao Kambhampati[1] Karthik Valmeekam[1] Lin Guan[1] Mudit Verma[1] Kaya Stechly[1] Siddhant Bhambri[1] Lucas Saldyt[1] Anil Murthy[1]

AlphaZero -> AlphaGeometry, AlphaProof



Lachlan's PhD - "AlphaPhysics"

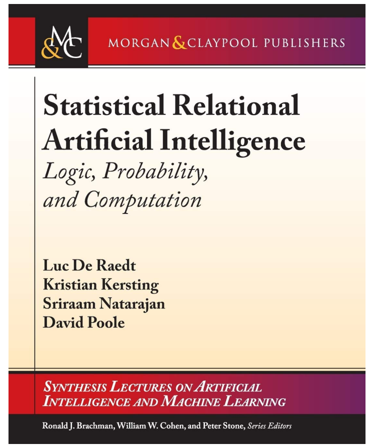
Statistics/NN/LLM+ Logic Combinations

StarAI =

RelationalAI/Logic +

Learning + Statistics (1980s)

Fusemate



NeSy =

Neural Networks + Symbolic Reasoning

Neural-Symbolic Learning and Reasoning:
A Survey and Interpretation

Tarek R. Besold et al TAREK-R.BESOLD@CITY.AC.UK
Department of Computer Science, City, University of London

NeSy + StarAI ?

From Statistical Relational to Neural Symbolic Artificial Intelligence: a Survey.

Giuseppe Marra[a], Sebastijan Dumančić[c], Robin Manhaeve[a], Luc De Raedt[a,b]

[a] *KU Leuven, Department of Computer Science and Leuven.AI*
[b] *Örebro University, Center for Applied Autonomous Sensor Systems*
[c] *Delft University of Technology, Department of Software Technology*

DeepProbLog - see below

LLMs + Logic

Augmented Language Models: a Survey

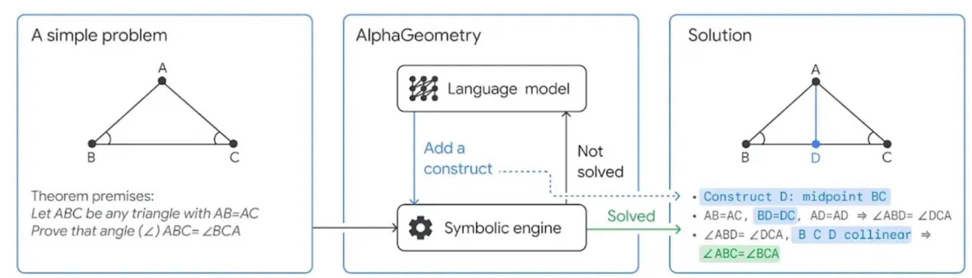
Grégoire Mialon* et al *gmialon@meta.com*

See below

Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks

Subbarao Kambhampati[1] Karthik Valmeekam[1] Lin Guan[1] Mudit Verma[1] Kaya Stechly[1] Siddhant Bhambri[1] Lucas Saldyt[1] Anil Murthy[1]

AlphaZero -> AlphaGeometry, AlphaProof



Lachlan's PhD - "AlphaPhysics"

LLM + Logic: LLMs *Are* Logic Reasoners?

Task LLM with Reasoning

ProntoQa [Saparov and He, 2023]

Synthetic Data

Varying redundancy (distractors)

Varying length of reasoning chains

Each composite number is not liquid. Every composite number is a fraction. Every composite number is a number. Negative numbers are not large. Every fraction is large. Each fraction is a real number. Fractions are integers. Integers are temperate. Each number is slow. Each even number is loud. Even numbers are natural numbers. Alex is an even number. Alex is a composite number.

True or false: Alex is large.

Prompt Engineering

In-prompt training one/view shot

Chain-of-thought "explain your reasoning"

Instruct LLM to use strategies

(backward/forward/SOS - own work)

Self-critique

Explainability?

LLM explanation can be nonsense

Correctness and Scalability?

More complex logic, e.g. quantifiers

Planning task, see Subbarao Kambhampati

Reasoning at all?

Or lookup?

LLM + Logic: Hierarchical Combination

Reliable Natural Language Understanding with Large Language Models and Answer Set Programming [Rajasekharan et al, ICLP 2023]

Example 3.1:

```
Question: Alan noticed that his toy car rolls further on a wood
floor than on a thick carpet. This suggests that:
(world1: wood floor, world2: thick carpet)
        (A) The carpet has more resistance (Solution)
        (B) The floor has more resistance
```

Approach

(1) LLM w/ fine tuning translates problem into logic programming query

(2) Logic programming system answers query modulo background knowledge

```
qplus(friction, heat).                qminus(friction, speed).
qplus(speed, distance).               qminus(distance, loudness).
positive(X, Y) :- qplus(X, Y).        negative(X, Y) :- qminus(X, Y).
positive(X, Y) :- qplus(Y, X).        negative(X, Y) :- qminus(Y, X).

opposite_w(world1,world2).          opposite_v(higher,lower).
opposite_w(world2,world1).          opposite_v(lower,higher).
conc(P, V, W) :- obs(P, Vr, Wr), property(P),
                 opposite_w(W,Wr),  opposite_v(V,Vr).

property(friction).     property(heat).     property(speed).
```

Autocorrecting Translation Errors

Automated Theorem Provers Help Improve Large Language Model Reasoning [McGinness, B., LPAR 2024]

```
Each integer is not fruity.
Negative numbers are brown.
Wren is an integer.
```

LLM (wrong):

```
! [X] : (fruity(X) => integer(X)))
integer(wren)
! [X] : integer(X)
brown(negative)
```

Auto-corrected:

```
! [X] : (fruity(X) => ~ integer(X))
integer(wren)
% ! [X] : integer(X) is an NonFixableError
! [I] : (negative_number(I) => brown(I))
```

LLM + Logic: Hierarchical Combination

Reliable Natural Language Understanding with Large Language Models and Answer Set Programming [Rajasekharan et al, ICLP 2023]

Example 3.1:

```
Question: Alan noticed that his toy car rolls further on a wood
floor than on a thick carpet. This suggests that:
(world1: wood floor, world2: thick carpet)
       (A) The carpet has more resistance (Solution)
       (B) The floor has more resistance
```

Approach

LLMs as intelligent parsers

(1) LLM w/ fine tuning translates problem into logic programming query

(2) Logic programming system answers query modulo background knowledge

```
qplus(friction, heat).            qminus(friction, speed).
qplus(speed, distance).           qminus(distance, loudness).
positive(X, Y) :- qplus(X, Y).    negative(X, Y) :- qminus(X, Y).
positive(X, Y) :- qplus(Y, X).    negative(X, Y) :- qminus(Y, X).

opposite_w(world1,world2).        opposite_v(higher,lower).
opposite_w(world2,world1).        opposite_v(lower,higher).
conc(P, V, W) :- obs(P, Vr, Wr), property(P),
                 opposite_w(W,Wr),  opposite_v(V,Vr).

property(friction).    property(heat).    property(speed).
```

Autocorrecting Translation Errors

Automated Theorem Provers Help Improve Large Language Model Reasoning

[McGinness, B., LPAR 2024]

```
Each integer is not fruity.
Negative numbers are brown.
Wren is an integer.
```

LLM (wrong):

```
! [X] : (fruity(X) => integer(X)))
integer(wren)
! [X] : integer(X)
brown(negative)
```

Auto-corrected:

```
! [X] : (fruity(X) => ~ integer(X))
integer(wren)
% ! [X] : integer(X) is an NonFixableError
! [I] : (negative_number(I) => brown(I))
```

LLM + Logic: Hierarchical Combination

Reliable Natural Language Understanding with Large Language Models and Answer Set Programming [Rajasekharan et al, ICLP 2023]

Example 3.1:

```
Question: Alan noticed that his toy car rolls further on a wood
floor than on a thick carpet. This suggests that:
(world1: wood floor, world2: thick carpet)
        (A) The carpet has more resistance (Solution)
        (B) The floor has more resistance
```

Approach

LLMs as intelligent parsers

(1) LLM w/ fine tuning translates problem into logic programming query

(2) Logic programming system answers query modulo background knowledge

```
qplus(friction, heat).               qminus(friction, speed).
qplus(speed, distance).              qminus(distance, loudness).
positive(X, Y) :- qplus(X, Y).       negative(X, Y) :- qminus(X, Y).
positive(X, Y) :- qplus(Y, X).       negative(X, Y) :- qminus(Y, X).

opposite_w(world1,world2).        opposite_v(higher,lower).
opposite_w(world2,world1).        opposite_v(lower,higher).
conc(P, V, W) :- obs(P, Vr, Wr), property(P),
                 opposite_w(W,Wr),  opposite_v(V,Vr).

property(friction).     property(heat).     property(speed).
```

Autocorrecting Translation Errors

Automated Theorem Provers Help Improve Large Language Model Reasoning

[McGinness, B., LPAR 2024]

```
Each integer is not fruity.
Negative numbers are brown.
Wren is an integer.
```

LLM (wrong):

```
! [X] : (fruity(X) => integer(X)))
integer(wren)
! [X] : integer(X)
brown(negative)
```

Auto-corrected:

```
! [X] : (fruity(X) => ~ integer(X))
integer(wren)
% ! [X] : integer(X) is an NonFixableError
! [I] : (negative_number(I) => brown(I))
```

LLM + Logic: Hierarchical Combination

Reliable Natural Language Understanding with Large Language Models and Answer Set Programming [Rajasekharan et al, ICLP 2023]

Example 3.1:

```
Question: Alan noticed that his toy car rolls further on a wood
floor than on a thick carpet. This suggests that:
(world1: wood floor, world2: thick carpet)
       (A) The carpet has more resistance (Solution)
       (B) The floor has more resistance
```

Approach

LLMs as intelligent parsers

(1) LLM w/ fine tuning translates problem into logic programming query

(2) Logic programming system answers query modulo background knowledge

```
qplus(friction, heat).            qminus(friction, speed).
qplus(speed, distance).           qminus(distance, loudness).
positive(X, Y) :- qplus(X, Y).    negative(X, Y) :- qminus(X, Y).
positive(X, Y) :- qplus(Y, X).    negative(X, Y) :- qminus(Y, X).

opposite_w(world1,world2).        opposite_v(higher,lower).
opposite_w(world2,world1).        opposite_v(lower,higher).
conc(P, V, W) :- obs(P, Vr, Wr), property(P),
                 opposite_w(W,Wr),  opposite_v(V,Vr).

property(friction).     property(heat).     property(speed).
```

Autocorrecting Translation Errors

Automated Theorem Provers Help Improve Large Language Model Reasoning

[McGinness, B., LPAR 2024]

```
Each integer is not fruity.
Negative numbers are brown.
Wren is an integer.
```

LLM (wrong):

```
! [X] : (fruity(X) => integer(X)))
integer(wren)
! [X] : integer(X)
brown(negative)
```

Auto-corrected:

```
! [X] : (fruity(X) => ~ integer(X))
integer(wren)
% ! [X] : integer(X) is an NonFixableError
! [I] : (negative_number(I) => brown(I))
```

LLM + Logic: Hierarchical Combination

Reliable Natural Language Understanding with Large Language Models and Answer Set Programming [Rajasekharan et al, ICLP 2023]

Example 3.1:

```
Question: Alan noticed that his toy car rolls further on a wood
floor than on a thick carpet. This suggests that:
(world1: wood floor, world2: thick carpet)
       (A) The carpet has more resistance (Solution)
       (B) The floor has more resistance
```

Approach

LLMs as intelligent parsers

(1) LLM w/ fine tuning translates problem into logic programming query

(2) Logic programming system answers query modulo background knowledge

```
qplus(friction, heat).              qminus(friction, speed).
qplus(speed, distance).             qminus(distance, loudness).
positive(X, Y) :- qplus(X, Y).      negative(X, Y) :- qminus(X, Y).
positive(X, Y) :- qplus(Y, X).      negative(X, Y) :- qminus(Y, X).

opposite_w(world1,world2).          opposite_v(higher,lower).
opposite_w(world2,world1).          opposite_v(lower,higher).
conc(P, V, W) :- obs(P, Vr, Wr), property(P),
                 opposite_w(W,Wr),  opposite_v(V,Vr).

property(friction).     property(heat).     property(speed).
```

Autocorrecting Translation Errors

Automated Theorem Provers Help Improve Large Language Model Reasoning

[McGinness, B., LPAR 2024]

```
Each integer is not fruity.
Negative numbers are brown.
Wren is an integer.
```

LLM (wrong):

```
! [X] : (fruity(X) => integer(X)))
integer(wren)
! [X] : integer(X)
brown(negative)
```

Auto-corrected:

```
! [X] : (fruity(X) => ~ integer(X))
integer(wren)
% ! [X] : integer(X) is an NonFixableError
! [I] : (negative_number(I) => brown(I))
```

LLM + Logic: Hierarchical Combination

Reliable Natural Language Understanding with Large Language Models and Answer Set Programming [Rajasekharan et al, ICLP 2023]

Example 3.1:

```
Question: Alan noticed that his toy car rolls further on a wood
floor than on a thick carpet. This suggests that:
(world1: wood floor, world2: thick carpet)
       (A) The carpet has more resistance (Solution)
       (B) The floor has more resistance
```

Approach   LLMs as intelligent parsers

(1) LLM w/ fine tuning translates problem into logic programming query

(2) Logic programming system answers query modulo background knowledge

```
qplus(friction, heat).             qminus(friction, speed).
qplus(speed, distance).            qminus(distance, loudness).
positive(X, Y) :- qplus(X, Y).     negative(X, Y) :- qminus(X, Y).
positive(X, Y) :- qplus(Y, X).     negative(X, Y) :- qminus(Y, X).

opposite_w(world1,world2).        opposite_v(higher,lower).
opposite_w(world2,world1).        opposite_v(lower,higher).
conc(P, V, W) :- obs(P, Vr, Wr), property(P),
                 opposite_w(W,Wr),  opposite_v(V,Vr).

property(friction).    property(heat).    property(speed).
```

Autocorrecting Translation Errors

Automated Theorem Provers Help Improve Large Language Model Reasoning

[McGinness, B., LPAR 2024]

```
Each integer is not fruity.
Negative numbers are brown.
Wren is an integer.
```

LLM (wrong):

```
! [X] : (fruity(X) => integer(X)))
integer(wren)
! [X] : integer(X)
brown(negative)
```

Auto-corrected:

```
! [X] : (fruity(X) => ~ integer(X))
integer(wren)
% ! [X] : integer(X) is an NonFixableError
! [I] : (negative_number(I) => brown(I))
```

LLM + Logic: Hierarchical Combination

Translation errors?

Reliable Natural Language Understanding with Large Language Models and Answer Set Programming [Rajasekharan et al, ICLP 2023]

Example 3.1:

```
Question: Alan noticed that his toy car rolls further on a wood
floor than on a thick carpet. This suggests that:
(world1: wood floor, world2: thick carpet)
        (A) The carpet has more resistance  (Solution)
        (B) The floor has more resistance
```

Approach

LLMs as intelligent parsers

(1) LLM w/ fine tuning translates problem into logic programming query

(2) Logic programming system answers query modulo background knowledge

```
qplus(friction, heat).             qminus(friction, speed).
qplus(speed, distance).            qminus(distance, loudness).
positive(X, Y) :- qplus(X, Y).     negative(X, Y) :- qminus(X, Y).
positive(X, Y) :- qplus(Y, X).     negative(X, Y) :- qminus(Y, X).

opposite_w(world1,world2).        opposite_v(higher,lower).
opposite_w(world2,world1).        opposite_v(lower,higher).
conc(P, V, W) :- obs(P, Vr, Wr), property(P),
                 opposite_w(W,Wr),  opposite_v(V,Vr).

property(friction).     property(heat).     property(speed).
```

Autocorrecting Translation Errors

Automated Theorem Provers Help Improve Large Language Model Reasoning

[McGinness, B., LPAR 2024]

```
Each integer is not fruity.
Negative numbers are brown.
Wren is an integer.
```

LLM (wrong):

```
! [X] : (fruity(X) => integer(X)))
integer(wren)
! [X] : integer(X)
brown(negative)
```

Auto-corrected:

```
! [X] : (fruity(X) => ~ integer(X))
integer(wren)
% ! [X] : integer(X) is an NonFixableError
! [I] : (negative_number(I) => brown(I))
```

Neural Networks + Symbolic Reasoning

DeepProbLog

Neural probabilistic logic programming in DeepProbLog

[Manhaeve et al, AIJ, 2021]

Inference

Query - does the following hold true?

addition(3, 5, 8)

addition([3, 8], [2, 5], 63)

Use backward chaining with

NN classifier for probabilistic facts

Returns query probability

Learning

End-to-end differentiable

-> back propagation modulo background knowledge

Here: learns digit image classifier from addition examples

Backward Chaining

```
?- addition(3, 5, 8)

addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.
```

Hard constraint

```
0.8 :: digit(0,0); 0.1 :: digit(0,1).
0.2 :: digit(1,0); 0.6 :: digit(1,1).
...
```

Neural Networks + Symbolic Reasoning

DeepProbLog

Neural probabilistic logic programming in DeepProbLog

[Manhaeve et al, AIJ, 2021]

Inference

Query - does the following hold true?

```
addition( , , 8)

addition([ , ], [ , ], 63)
```

Use backward chaining with

NN classifier for probabilistic facts

Returns query probability

Learning

End-to-end differentiable

-> back propagation modulo background knowledge

Here: learns digit image classifier from addition examples

Backward Chaining

```
?- addition( , , 8)

addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.
```

Hard constraint

```
0.8 :: digit( ,0); 0.1 :: digit( ,1).
0.2 :: digit( ,0); 0.6 :: digit( ,1).
...
```

Neural Networks + Symbolic Reasoning

DeepProbLog

Neural probabilistic logic programming in DeepProbLog

[Manhaeve et al, AIJ, 2021]

Inference

Query - does the following hold true?

```
addition(3, 5, 8)

addition([3, 8], [2, 5], 63)
```

Use backward chaining with

NN classifier for probabilistic facts

Returns query probability

Learning

End-to-end differentiable

-> back propagation modulo background knowledge

Here: learns digit image classifier from addition examples

Backward Chaining

```
?- addition(3, 5, 8)

addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.

0.8 :: digit(0,0); 0.1 :: digit(0,1).
0.2 :: digit(1,0); 0.6 :: digit(1,1).
...
```

Hard constraint

Neural Networks + Symbolic Reasoning

DeepProbLog

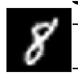
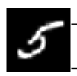
Neural probabilistic logic programming in DeepProbLog

[Manhaeve et al, AIJ, 2021]

Inference

Query - does the following hold true?

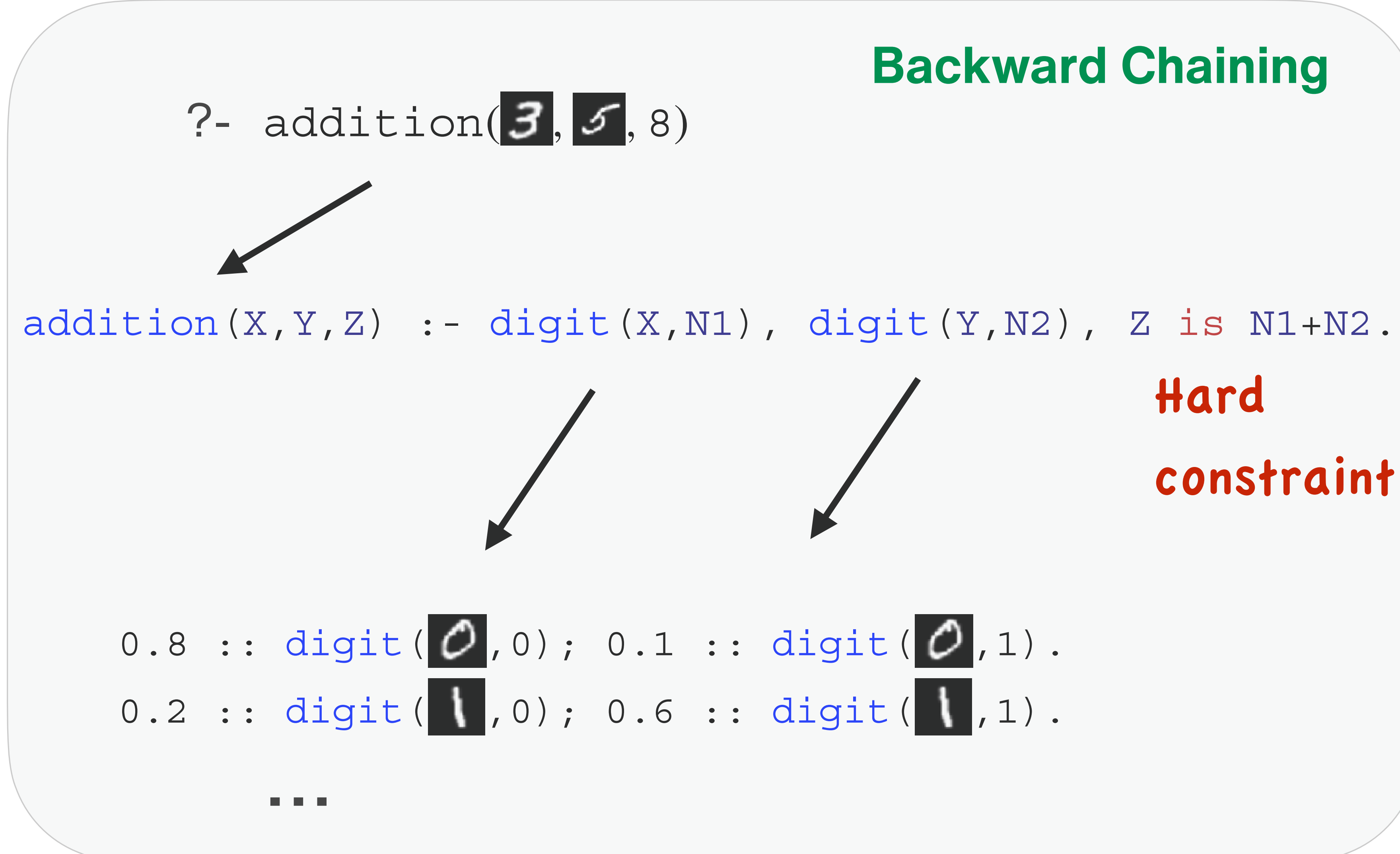
addition(, , 8)

addition([, ], [, ], 63)

Use backward chaining with

NN classifier for probabilistic facts

Returns query probability



Learning

End-to-end differentiable

-> back propagation modulo background knowledge

Here: learns digit image classifier from addition examples

Neural Networks + Symbolic Reasoning

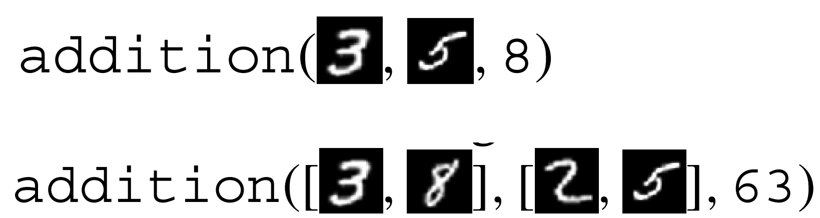
DeepProbLog

Neural probabilistic logic programming in DeepProbLog

[Manhaeve et al, AIJ, 2021]

Inference

Query - does the following hold true?



Use backward chaining with

NN classifier for probabilistic facts

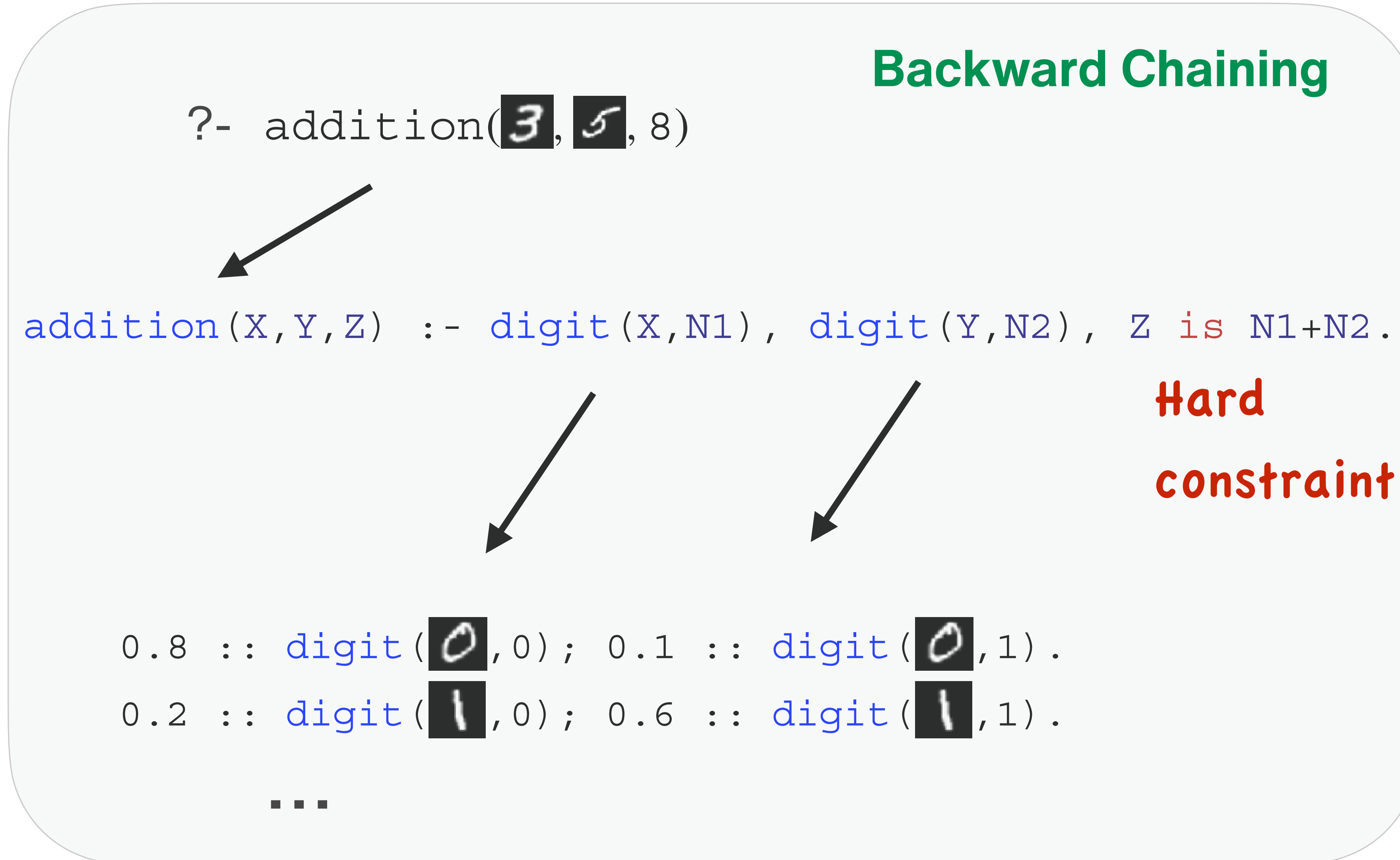
Returns query probability

Learning

End-to-end differentiable

-> back propagation modulo background knowledge

Here: learns digit image classifier from addition examples



Neural Networks + Symbolic Reasoning

DeepProbLog

Neural probabilistic logic programming in DeepProbLog

[Manhaeve et al, AIJ, 2021]

Inference

Query - does the following hold true?

```
addition( , , 8)

addition([ , ], [ , ], 63)
```

Use backward chaining with

NN classifier for probabilistic facts

Returns query probability

Learning

End-to-end differentiable

-> back propagation modulo background knowledge

Here: learns digit image classifier from addition examples

Backward Chaining

```
?- addition( , , 8)

addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.

0.8 :: digit( ,0); 0.1 :: digit( ,1).
0.2 :: digit( ,0); 0.6 :: digit( ,1).
...
```

Hard constraint

"Strong" coupling

Counterpart LLM modulo?

Neural Networks + Smbolic Reasoning

Many More Architectures

- *Differentiable Theorem Proving* [Rocktäschel]

$\texttt{parentOf}(\text{HOMER}, \text{BART}).$
$\texttt{grandfatherOf}(X, Y) :- \texttt{fatherOf}(X, Z), \texttt{parentOf}(Z, Y).$
$\texttt{grandfatherOf}(\text{ABE}, Q)? \quad \{Q/\text{LISA}\}, \{Q/\text{BART}\}$

Reasoning in embedding space:

Example: unify $\boldsymbol{v}_{\texttt{grandfatherOf}}(X, \boldsymbol{v}_{\text{BART}})$ with $\boldsymbol{v}_{\texttt{grandpaOf}}(\boldsymbol{v}_{\text{ABE}}, \boldsymbol{v}_{\text{BART}})$

$$\Psi = \{X/\boldsymbol{v}_{\text{ABE}}\}, \quad \tau = \min(e^{-\|\boldsymbol{v}_{\texttt{grandfatherOf}} - \boldsymbol{v}_{\texttt{grandpaOf}}\|_2}, e^{-\|\boldsymbol{v}_{\text{BART}} - \boldsymbol{v}_{\text{BART}}\|_2})$$

- *Semantic Probabilistic Layers for Neuro-Symbolic Learning* [Ahmed et al NeurIPS, 2022]

  Logic constraints at the output layer, e.g. exclusivity constraints for classification
- *FFNSL: Feed-Forward Neural-Symbolic Learner* [Cunnington, Law, Lobo, Russo 2023]
- *Encodings of logic within NNs*
- *Logic Tensor Networks*
- *Neural Datalog over time*

Conclusions

Fusemate

- Probabilistic Logic Programming system
- Good

  Expressivity, good Python interface, reasonably optimized for intended use case (HMM-ish)
- Needs work

  Documentation, efficiency

LMM + Logic

- Current focus of research and D61 applications for “Explainability”

  ML/LLM -> generate solution candidates

  Probabilistic logic -> validate/complete solution candidates