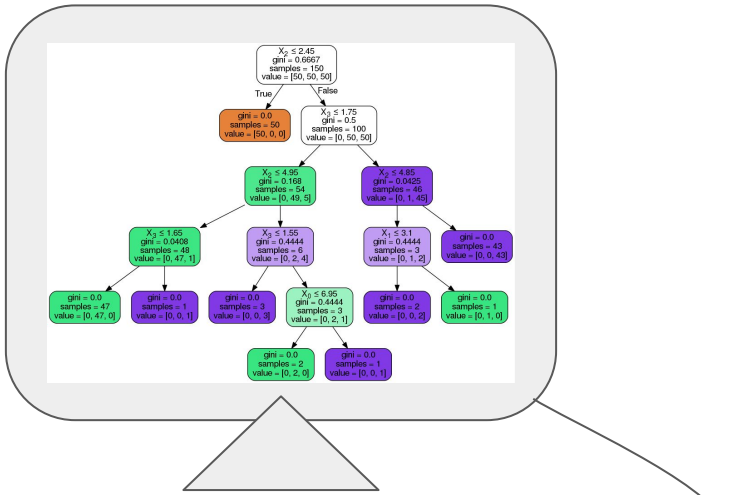
Towards Comprehensible representation of controllers using Machine Learning: Inductive Logic Programming

Under Prof. Dr. Jan Křetínský

As part of my summer internship
(DAAD)
at
Fakultät für Informatik
TU Munich

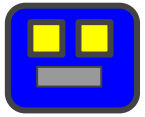
Gargi Balasubramaniam
Bachelors of Computer Science
Birla Institute of Technology and Science (BITS) Pilani, Goa, India
(4th Year)



- **Comprehensible representation of controllers**
  - What are they?
  - Model Checking??
  - MDPS???
- **Machine Learning**
  - Decision Trees
  - Inductive Logic Programming....

Towards Comprehensible representation of **controllers** using Machine Learning: Inductive Logic Programming

Real Life System + Specifications

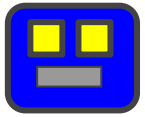


Safety is critical

Testing, Validation..

Write code manually and test

Real Life System + Specifications



Safety is critical

Testing, Validation..

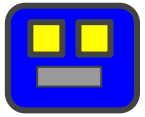
Model Checker (Model + Properties)

Controller Strategy

.
.
.
.
.

Write code manually and test
OR
Mathematically model the system and its properties (requirements) and feed it to a model checker.

Real Life System + Specifications



Safety is critical

Testing, Validation..

Model Checker (Model + Properties)

Controller Strategy

.
.
.
.
.

Write code manually and test
OR
Mathematically model the system and its properties (requirements) and feed it to a model checker.

The strategy i.e. the logic of the software which is to be followed by the physical system.

Controller Strategy

.

.

.

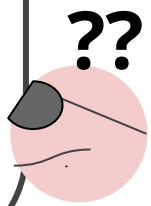
.

.

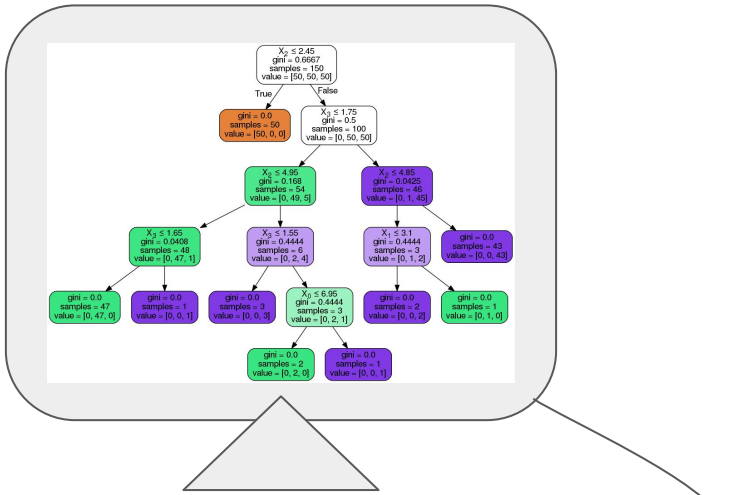

Controller
Strategy

State 1: Action(s) allowed
State 2: Action(s) allowed
State 3: Action(s) allowed
.
.
.
.
.
.
..

State 1: Action(s) allowed
State 2: Action(s) allowed
State 3: Action(s) allowed



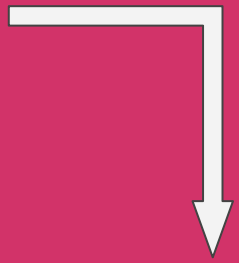
Towards **Comprehensible representation** of controllers using Machine Learning: Inductive Logic Programming



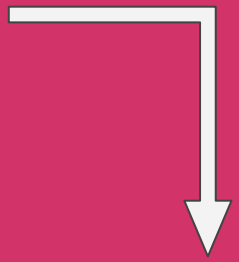
- **Comprehensible representation of controllers**
  - What are they?
  - Model Checking??
  - MDPS???
- **Machine Learning**
  - Decision Trees
  - Inductive Logic Programming....

Towards Comprehensible representation of controllers using **Machine Learning**: Inductive Logic Programming

Input: Sample data

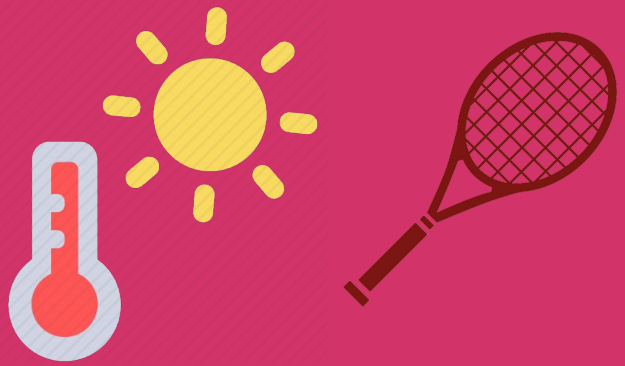


Build a model



Predict on unseen data

Input: Sample data



Build a model

Predict on unseen data

Input: Sample data

Build a model

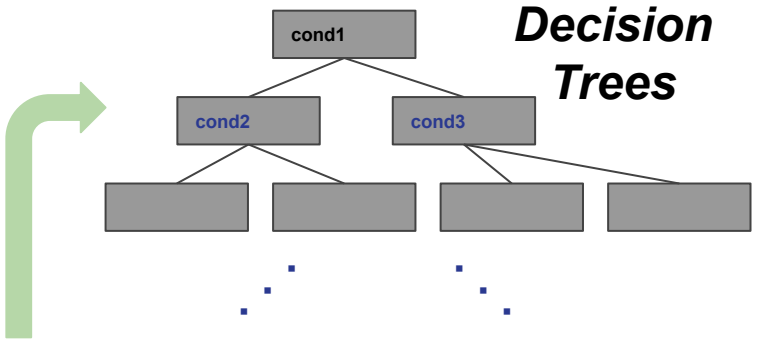
??

Predict on unseen data

AIM:
Why ML at all?

- Learn a strategy and represent (predict)
- In a much smaller form
- Capture the most essential decisions of a strategy
- While not compromising on safety, i.e. not predicting spuriously.

Build a *model:*
Previous work



Decision Trees

Controller Strategy

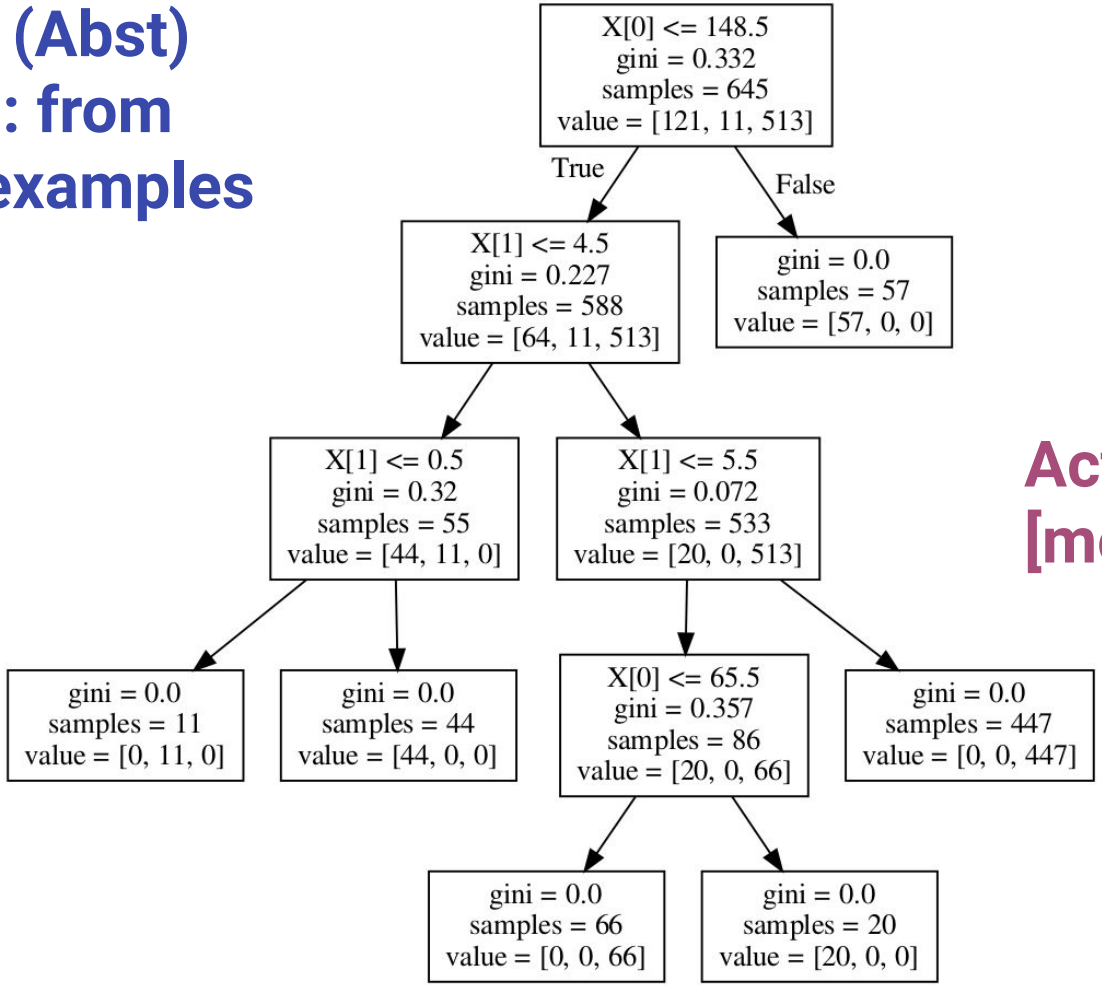
??

Machine Learning

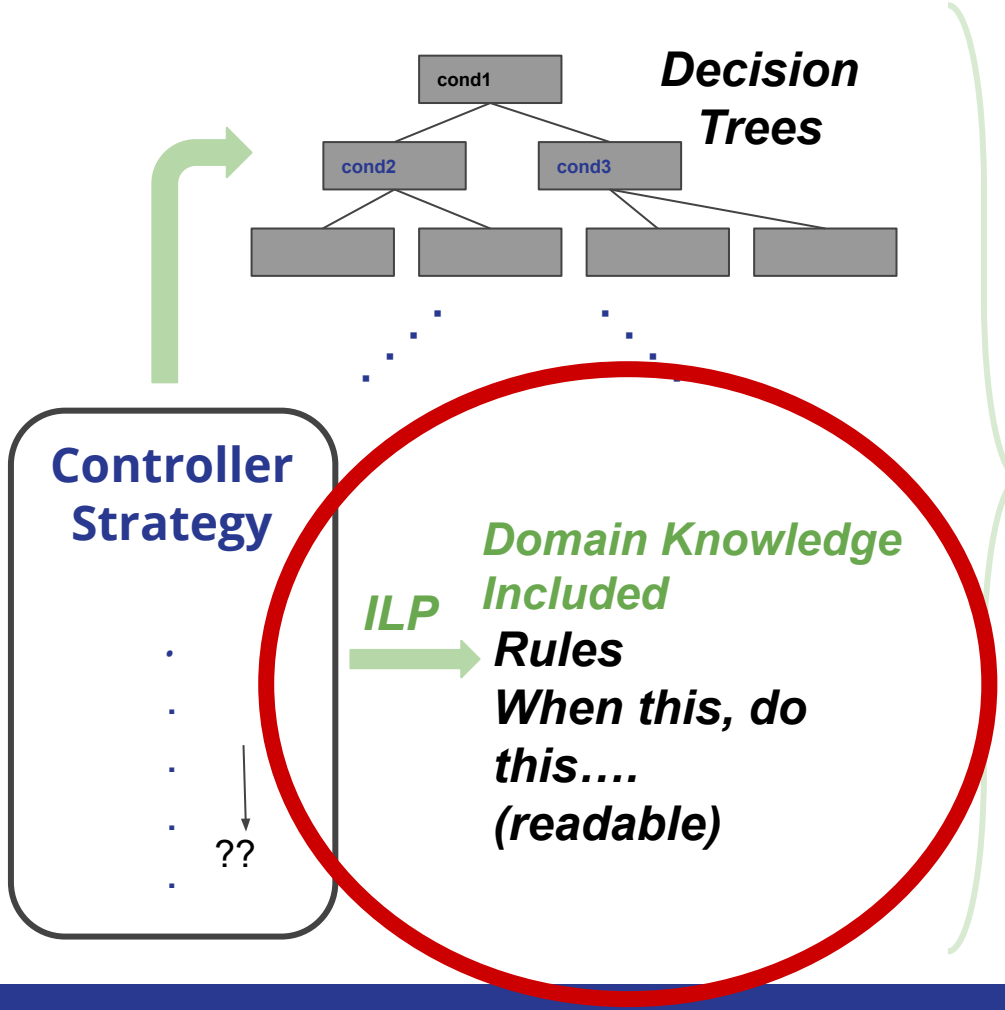
- MDPs
- Stochastic Hybrid Games
- Graph Games

Firewire (Abst) Dataset : from PRISM examples

X[0] : x
X[1] : s



Actions: [move,round,time]



Now our model is Inductive Logic Programming

Towards Comprehensible representation of controllers using Machine Learning: **Inductive Logic Programming**

For Starters :
Inductive Logic Programming

Deductive vs Inductive?

Hypothesis: All Girls are Smart

Deductive vs Inductive?

Hypothesis: All Girls are Smart

and

Jill is a girl

Deductive vs Inductive?

Hypothesis: All Girls are Smart

and

Jill is a girl

Jill is smart

Deductive vs Inductive?

Hypothesis: All Girls are Smart

and

Jill is a girl

Jill is smart

DEDUCTIVE

Deducing ***Specific*** details
from ***General*** Hypothesis

Deductive vs Inductive?

Jill is smart

Deductive vs Inductive?

Jill is smart

and

Jill is a girl

Deductive vs Inductive?

Jill is smart

and

Jill is a girl

Hypothesis: All Girls are Smart

Deductive vs Inductive?

Jill is smart

and

Jill is a girl

Hypothesis: All Girls are Smart

INDUCTIVE

"Inducing" ***General*** hypothesis from ***Specific*** Details

Indeed, Machine Learning in some sense..

For Starters: ILP

What we have :

1. Background Knowledge (B)
2. Set of Positive Examples (E+)
3. Set of Negative Examples (E-)

For Starters: ILP

What we have :

1. Background Knowledge (B)
2. Set of Positive Examples (E+)
3. Set of Negative Examples (E-)

What we want:

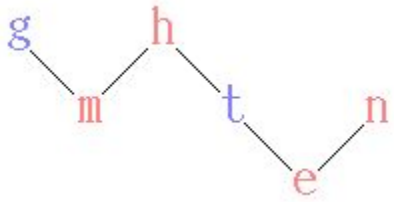
A Hypothesis which is ***"good"***:

-> Maximises the number of positive examples satisfied

-> Does NOT satisfy any negative examples

-> Has literals in the body as less as possible

For Starters: *ILP*



par: *parent*, *fem*: *female*, *dau*: *daughter*, *g*: *George*, *h*: *Helen*, *m*: *Mary*, *t*: *Tom*, *n*: *Nancy*, and *e*: *Eve*.

What we have :

1. Background Knowledge (B)
2. Set of Positive Examples (E+)
3. Set of Negative Examples (E-)

What we want:

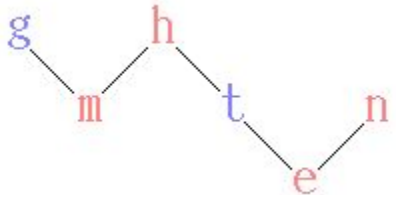
A Hypothesis which is ***"good"***:

-> Maximises the number of positive examples satisfied

-> Does NOT satisfy any negative examples

-> Has literals in the body as less as possible

For Starters: *ILP*



par: *parent*, *fem*: *female*, *dau*: *daughter*, *g*: *George*, *h*: *Helen*, *m*: *Mary*, *t*: *Tom*, *n*: *Nancy*, and *e*: *Eve*.

$$par(h,m) \wedge par(h,t) \wedge par(g,m) \wedge par(t,e) \wedge par(n,e) \wedge fem(h) \wedge fem(m) \wedge fem(n) \wedge fem(e),$$

What we have :

1. Background Knowledge (B)
2. Set of Positive Examples (E+)
3. Set of Negative Examples (E-)

$$dau(m,h) \wedge dau(e,t),$$

What we want:

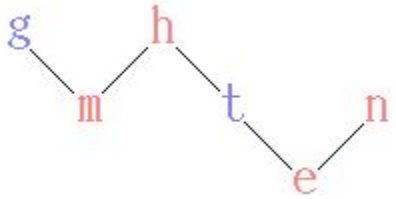
A Hypothesis which is ***"good"***:

-> Maximises the number of positive examples satisfied

-> Does NOT satisfy any negative examples

-> Has literals in the body as less as possible

For Starters: *ILP*



par: *parent*, *fem*: *female*, *dau*: *daughter*, *g*: *George*, *h*: *Helen*, *m*: *Mary*, *t*: *Tom*, *n*: *Nancy*, and *e*: *Eve*.

$$par(h,m) \wedge par(h,t) \wedge par(g,m) \wedge par(t,e) \wedge par(n,e) \wedge fem(h) \wedge fem(m) \wedge fem(n) \wedge fem(e),$$

What we have :

1. Background Knowledge (B)
2. Set of Positive Examples (E+)
3. Set of Negative Examples (E-)

$$dau(m,h) \wedge dau(e,t),$$

What we want:

A Hypothesis which is ***"good"***:

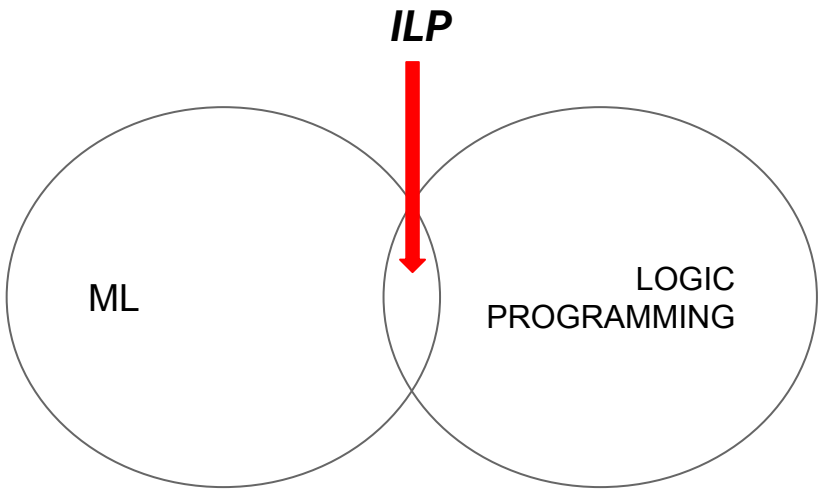
$$dau(x_{me}, x_{ht}) \leftarrow par(x_{ht}, x_{me}) \wedge fem(x_{me})$$

-> Maximises the number of positive examples satisfied

-> Does NOT satisfy any negative examples

-> Has literals in the body as less as possible

For Starters: *ILP*



What we have :

1. Background Knowledge (B)
2. Set of Positive Examples (E+)
3. Set of Negative Examples (E-)

What we want:

A Hypothesis which is ***"good"***:

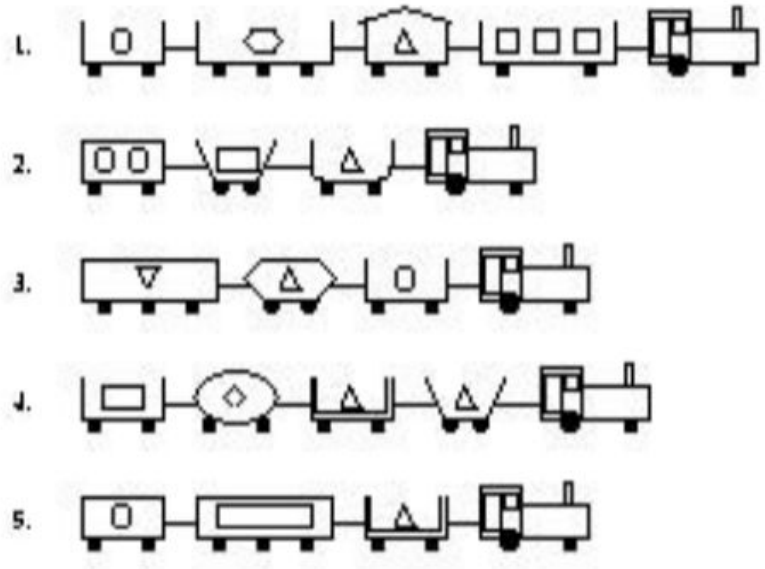
-> Maximises the number of positive examples satisfied

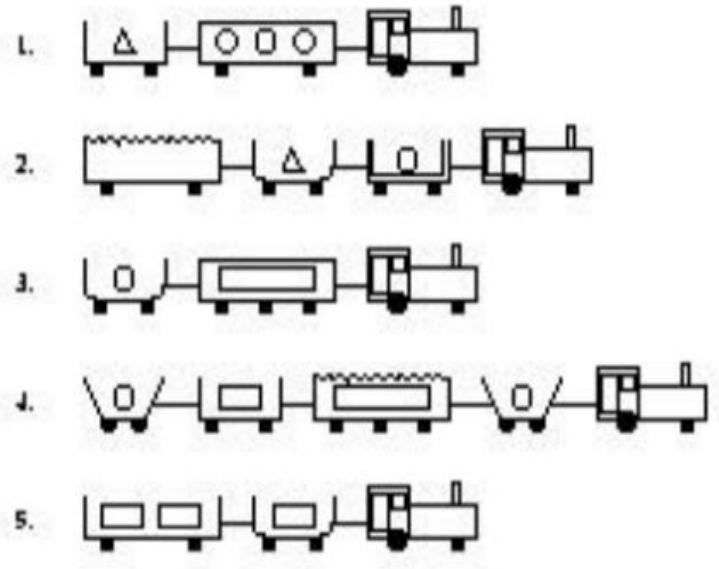
-> Does NOT satisfy any negative examples

-> Has literals in the body as less as possible

EXAMPLES

TRAINS





```
car(car_51).  car(car_52).  car(car_53).
car(car_61).  car(car_62).
car(car_71).  car(car_72).  car(car_73).
car(car_81).  car(car_82).
car(car_91).  car(car_92).  car(car_93).  car(car_94).
car(car_101).  car(car_102).

shape(elipse).  shape(hexagon).  shape(rectangle).  shape(u_shaped).
shape(triangle). shape(circle). shape(nil).

train(east1).  train(east2).  train(east3).  train(east4).  train(east5).
train(west6).  train(west7).  train(west8).  train(west9).  train(west10).


% eastbound train 1
short(car_12).      % 0
closed(car_12).     % 1
long(car_11).       % 2
long(car_13).
short(car_14).
open_car(car_11).       % 3
open_car(car_13).
open_car(car_14).
shape(car_11,rectangle). % 4,5
shape(car_12,rectangle).
shape(car_13,rectangle).
shape(car_14,rectangle).
load(car_11,rectangle,3). % 6,7,8
load(car_12,triangle,1).
load(car_13,hexagon,1).
load(car_14,circle,1).
wheels(car_11,2).     % 9,10
wheels(car_12,2).
wheels(car_13,3).
wheels(car_14,2).
has_car(east1,car_11). % 11,12
has_car(east1,car_12).
has_car(east1,car_13).
```

B

```
eastbound(east1).
eastbound(east2).
eastbound(east3).
eastbound(east4).
eastbound(east5).
```

E+

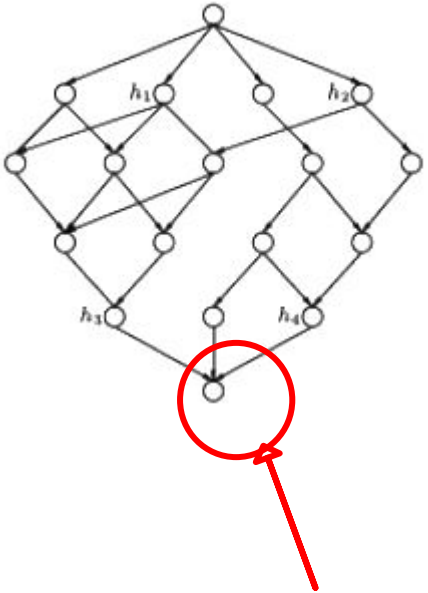
```
eastbound(west6).
eastbound(west7).
eastbound(west8).
eastbound(west9).
eastbound(west10).
```

E−

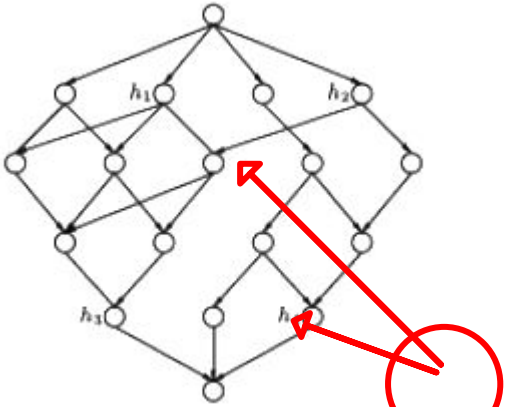
```
eastbound(A) :-
   has_car(A,B), has_car(A,C), has_car(A,D), has_car(A,E),
   short(E), short(C), closed(C), long(D),
   long(B), open_car(E), open_car(D), open_car(B),
   shape(E,rectangle), shape(D,rectangle), shape(C,rectangle), shape(B,rectangle),
   wheels(E,2), wheels(D,3), wheels(C,2), wheels(B,2),
   load(E,circle,1), load(D,hexagon,1), load(C,triangle,1), load(B,rectangle,3).
```

A general hypothesis about eastbound trains, given the background knowledge and positive and negative instances

Greedy search along the lattice of hypothesis

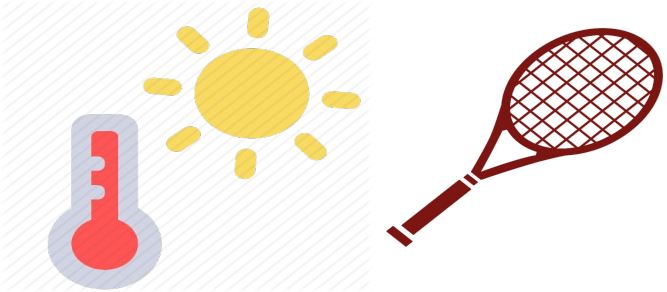


Most Specific Hypothesis (MSH) for a positive example e



More General Hypotheses

To play or not to play



```
[Rule 1] [Pos cover = 2 Neg cover = 0]
class(A,B) :-
   not (outlook(A,rain),windy(A,true)), outlook(A,sunny), humidity(A,C), lteq(C,70),
   random(B,[0.75-play,0.25-dont_play]).

[Rule 2] [Pos cover = 3 Neg cover = 0]
class(A,B) :-
   not (outlook(A,rain),windy(A,true)), outlook(A,sunny), not (humidity(A,C),lteq(C,70)), random(
   B,[0.8-dont_play,0.2-play]).

[Rule 3] [Pos cover = 7 Neg cover = 0]
class(A,B) :-
   not (outlook(A,rain),windy(A,true)), not outlook(A,sunny), random(B,[0.888889-play,0.111111-dont_play]).

[Rule 4] [Pos cover = 2 Neg cover = 0]
class(A,B) :-
   outlook(A,rain), windy(A,true), random(B,[0.75-dont_play,0.25-play]).
```

In the context of strategies..

Cruise Control

Ego

Ego is controlled by us

Front

Front is environment controlled : "given to us"

The goal of the adaptive cruise control in Ego is

1. To stay safe (by keeping the distance between the cars greater than a given safe distance) ***SAFETY***

2. To drive as close to Front as possible. ***OPTIMALITY***

Cruise Control

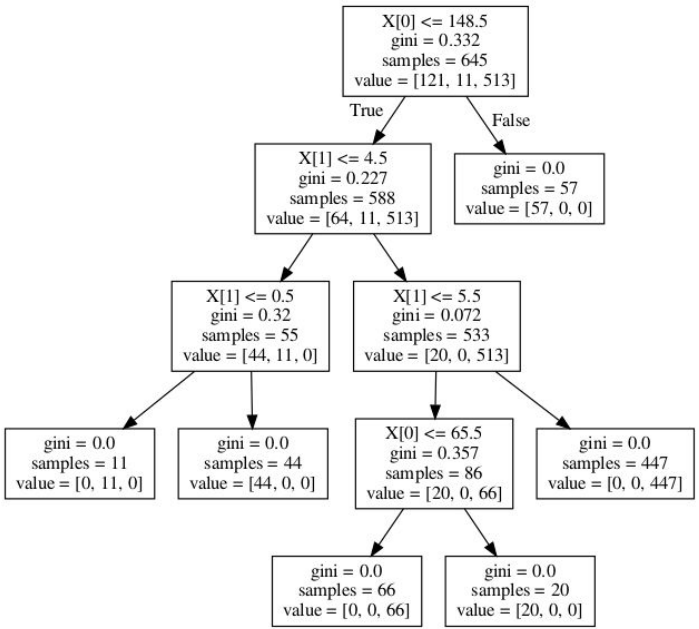
Ego

Ego is controlled by us

Front

Front is environment controlled : "given to us"

- **B** : Equations of Motion
- **E+**: When to accelerate
- **E-:** When to not accelerate



× firewire_abst_10__p
× firewire_exported_dt
× final_results.txt

```
[theory]

[Rule 1] [Pos cover = 55 Neg cover = 0]
class(A,B) :-
   variab(A,C), lteq(C,148), state(A,D), lteq(
   D,4),
   random(B,[0.775862068965517-move,0.20689655172
   4138-round,0.0172413793103448-time]).

[Rule 2] [Pos cover = 533 Neg cover = 0]
class(A,B) :-
   variab(A,C), lteq(C,148), not (state(A,D),lteq
   (D,4)), random(B,[0.958955223880597-time,0.039
   1791044776119-move,0.00186567164179104-round])
   .

[Rule 3] [Pos cover = 57 Neg cover = 0]
class(A,B) :-
   not (variab(A,C),lteq(C,148)), random(B,[0.966
   666666666667-move,0.0166666666666667-round,0.0
   166666666666667-time]).

[Training set performance]
           Actual
        +          -
     + 645         0         645
Pred
     -  0          0          0

        645        0         645

Accuracy = 1.0
[Training set summary] [[645,0,0,0]]
```

WHY?

For Starters: *ILP*

What we have :

1. Background Knowledge (B)
2. Set of Positive Examples (E+)
3. Set of Negative Examples (E-)

What we want:

A Hypothesis which is ***"good"***:

-> Maximises the number of positive examples satisfied

-> Does NOT satisfy any negative examples

-> Has literals in the body as less as possible

Towards a succinct representation ?
You decide the vocabulary!

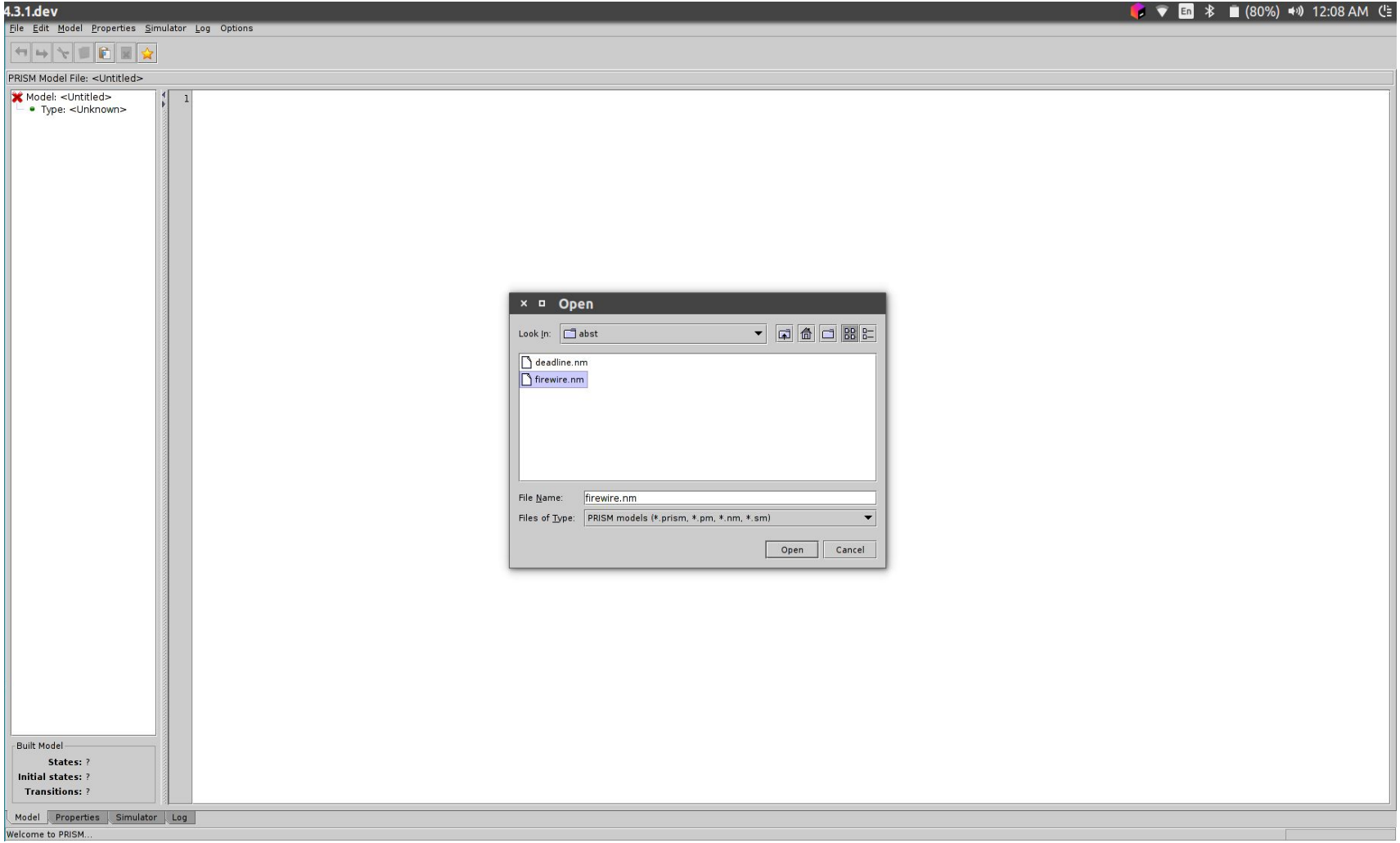
Dataset	Decision Tree Leaves	ILP Rules	Accuracy on Training Set
Cruise [a]	68	130	100%
Firewire	14	3	100%

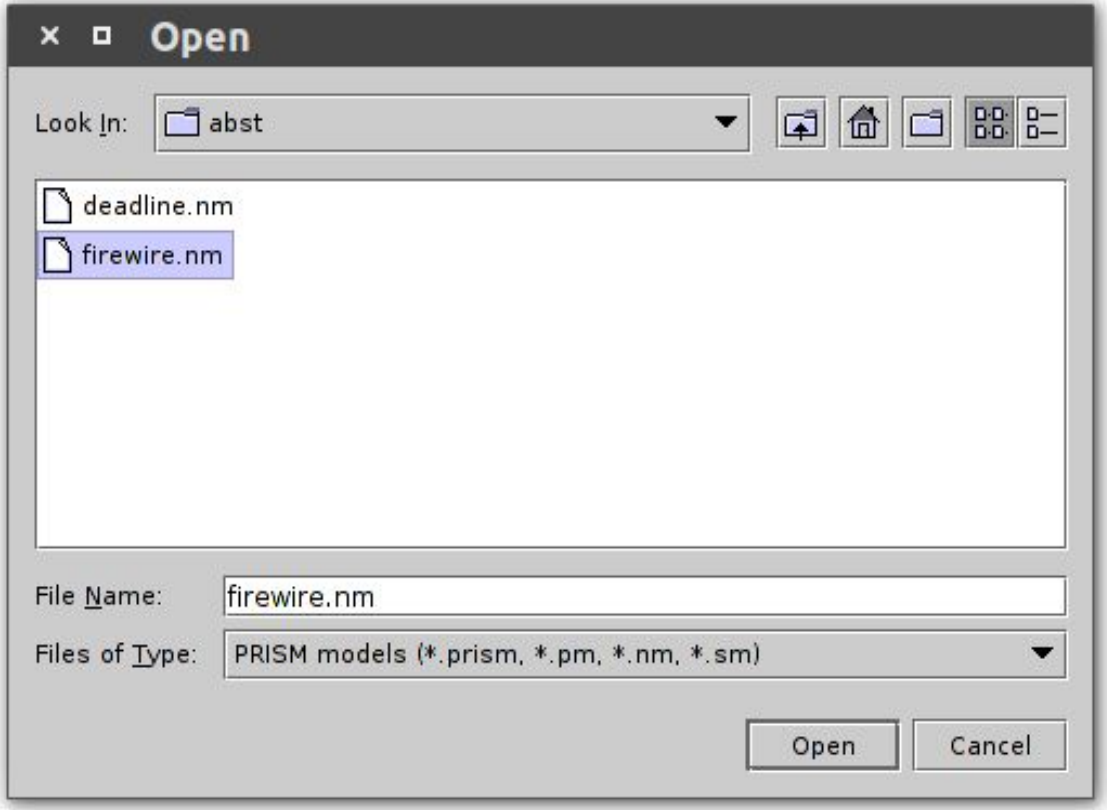
[a] A smaller sample of 5000 points.

Dataset	Decision Tree Leaves	ILP Rules	Accuracy on Training Set
Cruise [a]	68	130	100%
Firewire	14	3	100%

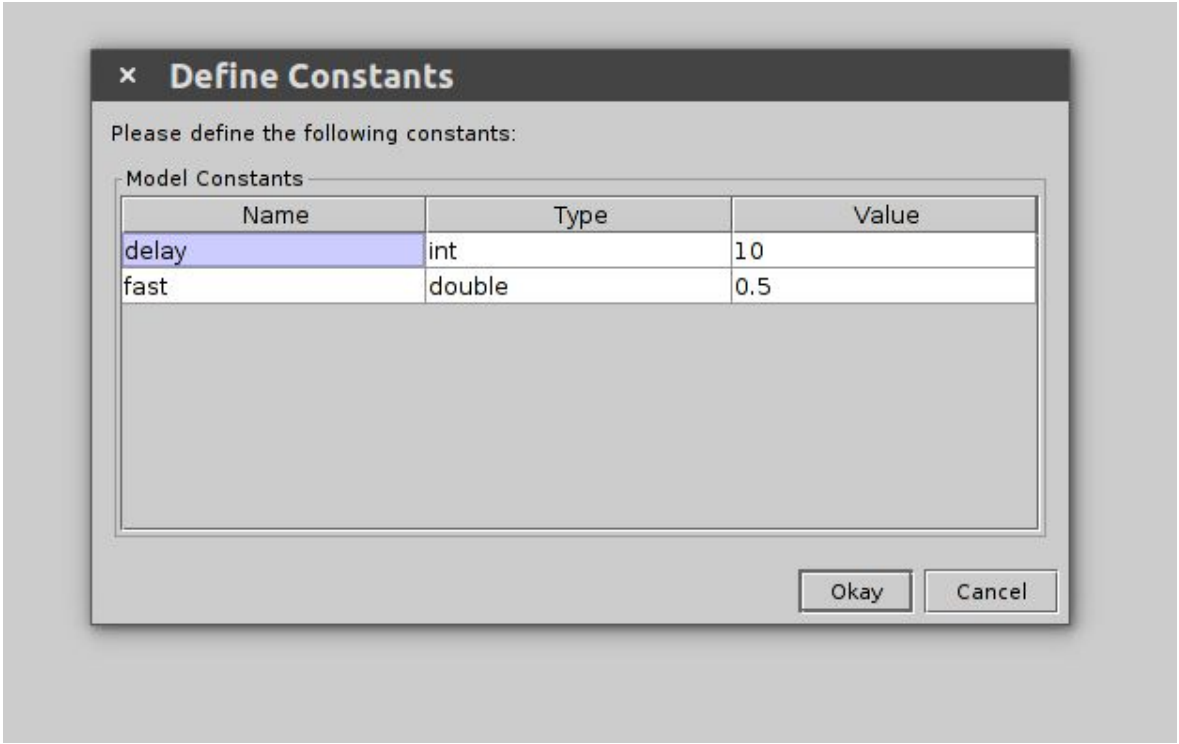
[a] A smaller sample of 5000 points.

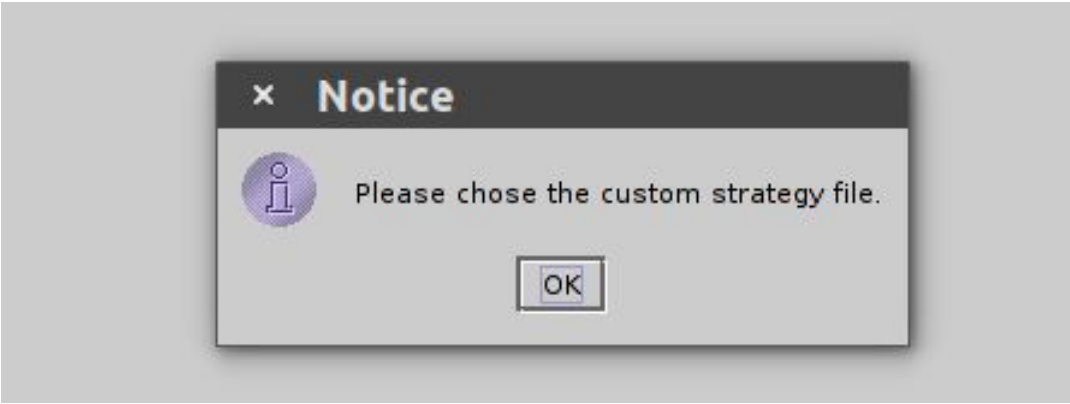
Simulating Custom Decision Tree Strategy in PRISM


4.3.1.dev
(80%) 12:08 AM
File Edit Model Properties Simulator Log Options
PRISM Model File: <Untitled>
Model: <Untitled>
Type: <Unknown>
1
Open
Look In:
abst
deadline.nm
firewire.nm
File Name:
firewire.nm
Files of Type:
PRISM models (*.prism, *.pm, *.nm, *.sm)
Open
Cancel
Built Model
States: ?
Initial states: ?
Transitions: ?
Model
Properties
Simulator
Log
Welcome to PRISM...


Open
Look In:
abst
deadline.nm
firewire.nm
File Name:
firewire.nm
Files of Type:
PRISM models (*.prism, *.pm, *.nm, *.sm)
Open
Cancel

```
// integer semantics version of abstract firewire protocol
// gxn 23/05/2001

mdp

// wire delay
const int delay;

// probability of choosing fast and slow
const double fast;
const double slow = 1-fast;

// largest constant the clock of the system is compared to
const int kx = 167;

module abstract_firewire

	// clock
	x : [0..kx+1];

	// local state
	s : [0..9];
	// 0 -start_start
	// 1 -fast_start
	// 2 -start_fast
	// 3 -start_slow
	// 4 -slow_start
	// 5 -fast_fast
	// 6 -fast_slow
	// 7 -slow_fast
	// 8 -slow_slow
	// 9 -done

	// initial state
	[time] s=0 & x<delay -> (x'=min(x+1,kx+1));
	[round] s=0 -> fast : (s'=1) + slow : (s'=4);
	[round] s=0 -> fast : (s'=2) + slow : (s'=3);
	// fast_start
	[time] s=1 & x<delay -> (x'=min(x+1,kx+1));
	[] s=1 -> fast : (s'=5) & (x'=0) + slow : (s'=6) & (x'=0);
	// start_fast
	[time] s=2 & x<delay -> (x'=min(x+1,kx+1));
	[] s=2 -> fast : (s'=5) & (x'=0) + slow : (s'=7) & (x'=0);
	// start_slow
	[time] s=3 & x<delay -> (x'=min(x+1,kx+1));
	[] s=3 -> fast : (s'=6) & (x'=0) + slow : (s'=8) & (x'=0);
	// slow_start
	[time] s=4 & x<delay -> (x'=min(x+1,kx+1));
	[] s=4 -> fast : (s'=7) & (x'=0) + slow : (s'=8) & (x'=0);
	// fast_fast
	[time] s=5 & (x<85) -> (x'=min(x+1,kx+1));
	[] s=5 & (x>=76) -> (s'=0) & (x'=0);
	[] s=5 & (x>=76-delay) -> (s'=9) & (x'=0);
	// fast_slow
	[time] s=6 & x<167 -> (x'=min(x+1,kx+1));
	[] s=6 & x>=159-delay -> (s'=9) & (x'=0);
	// slow_fast
	[time] s=7 & x<167 -> (x'=min(x+1,kx+1));
	[] s=7 & x>=159-delay -> (s'=9) & (x'=0);
	// slow_slow
```


Define Constants
Please define the following constants:
Model Constants
Name
Type
Value
delay
int
10
fast
double
0.5
Okay
Cancel


Notice
Please chose the custom strategy file.
OK

```
num_nodes=11
num_classes=3
depth=4
node lc rc predicate/action
0 1 10 X0<=148.5
1 2 5 X1<=4.5
2 3 4 X1<=0.5
3 -1 -1 round
4 -1 -1 move
5 6 9 X1<=5.5
6 7 8 X0<=65.5
7 -1 -1 time
8 -1 -1 move
9 -1 -1 time
10 -1 -1 move
num_features=2
X0:x
X1:s
```

3.1.dev

File Edit Model Properties Simulator Log Options

Automatic exploration

Custom Strategy S... | Simulate

Steps | 1

Backtracking

Backtrack

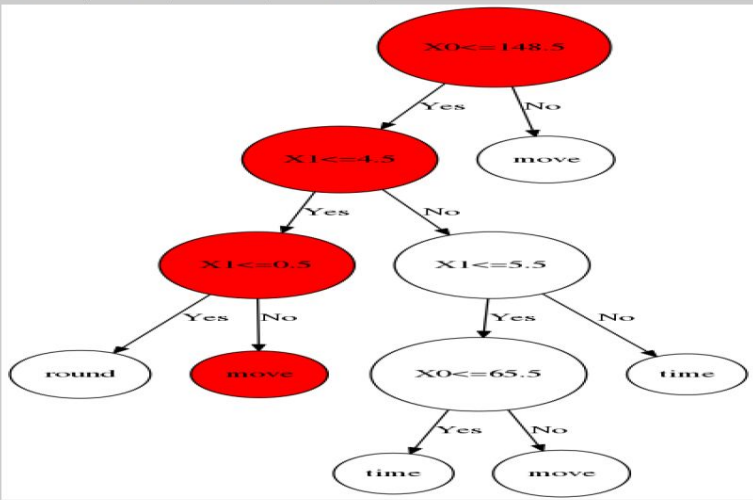
Steps | 1

Manual exploration

Module/[action]	Probability	Update
[time]	1.0	x'=1

Generate time automatically

Decision tree | State labels | Path formulae | Path information

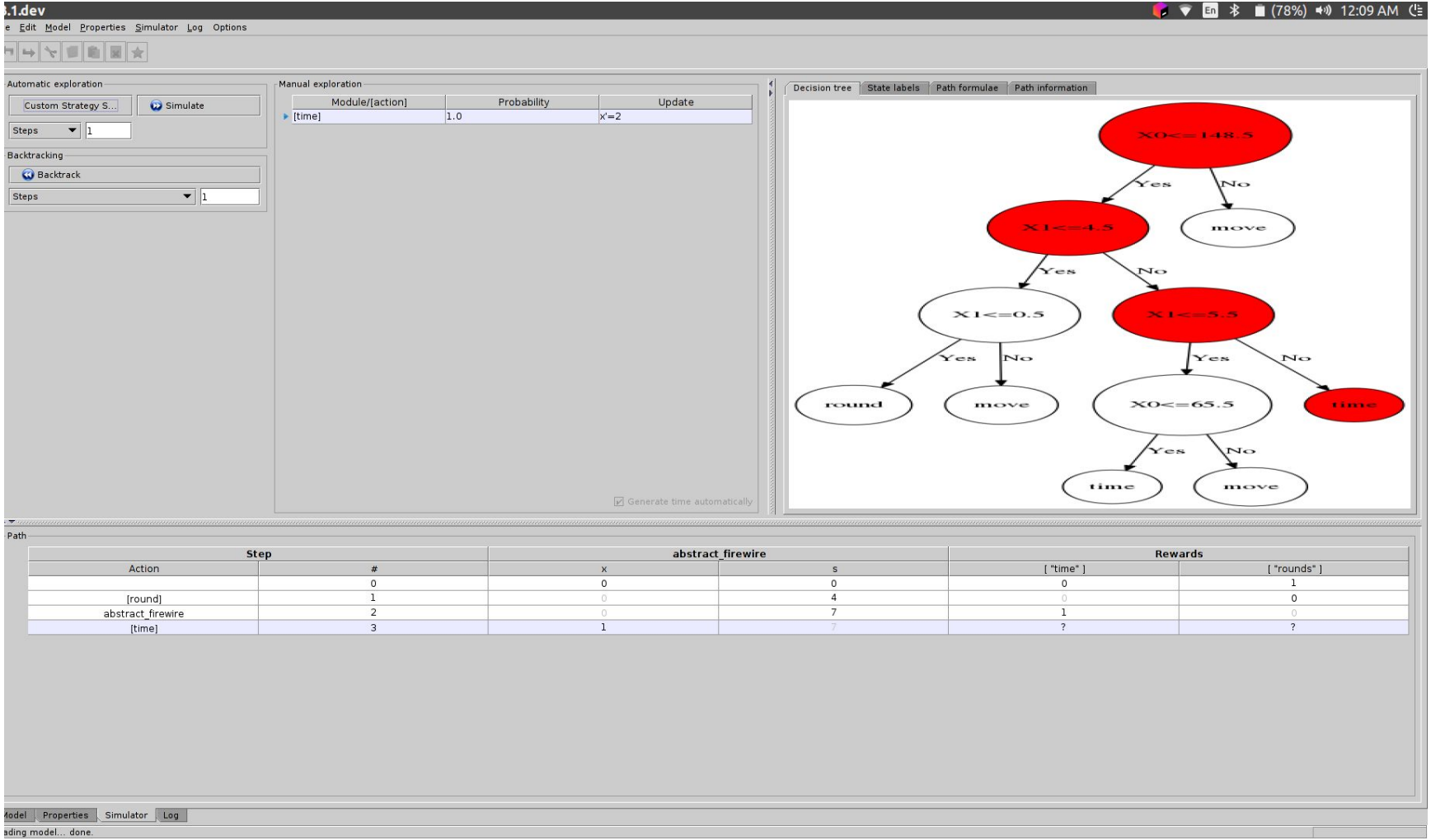


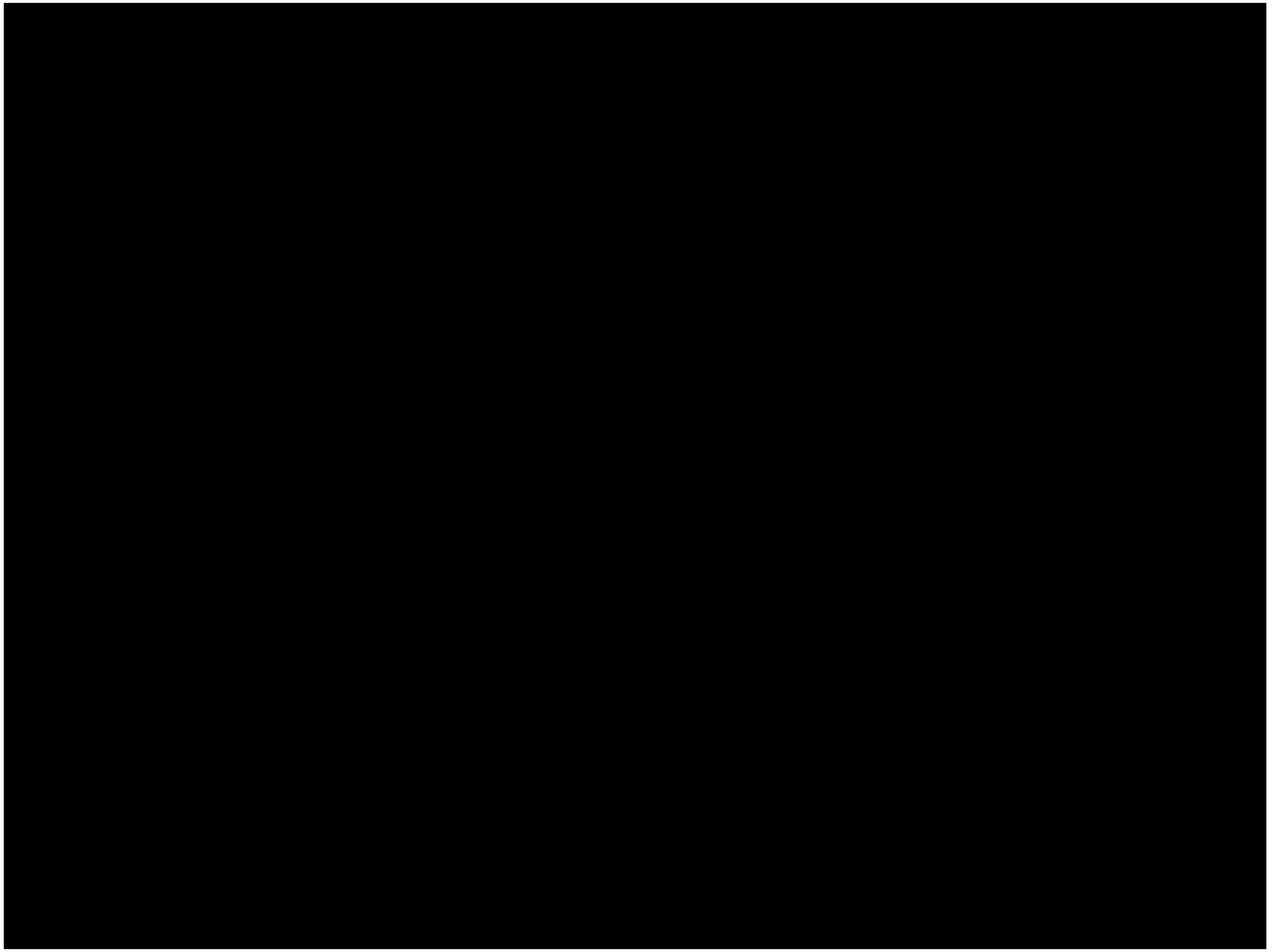
Path

Step		abstract_firewire		Rewards	
Action	#	x	s	["time"]	["rounds"]
	0	0	0	0	1
[round]	1	0	4	0	0
abstract_firewire	2	0	7	?	?

Model | Properties | Simulator | Log

oading model... done.


.1.dev
Edit Model Properties Simulator Log Options
Automatic exploration
Custom Strategy S...
Simulate
Steps
1
Backtracking
Backtrack
Steps
1
Manual exploration
Module/[action]
Probability
Update
[time]
1.0
x'=2
Generate time automatically
Decision tree
State labels
Path formulae
Path information
X0<=148.5
Yes
No
move
X1<=4.5
Yes
No
X1<=0.5
X1<=5.5
Yes
No
Yes
No
round
move
X0<=65.5
time
Yes
No
time
move
Path
Step
abstract_firewire
Rewards
Action
#
x
s
["time"]
["rounds"]
0
0
0
1
[round]
1
0
4
0
0
abstract_firewire
2
0
7
1
0
[time]
3
1
7
?
?
Model
Properties
Simulator
Log
ading model... done.



Real Life System
+ Specifications

Safety is critical

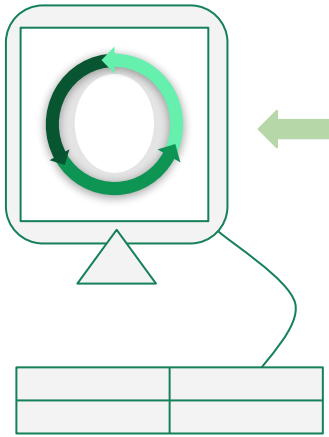
Testing, Validation..

Model Checker
(Model + Properties)

Controller Strategy

??

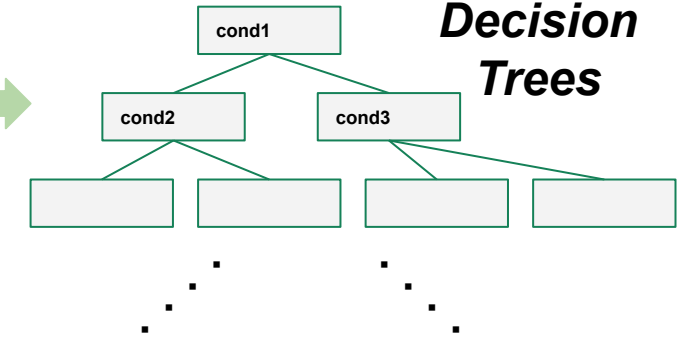
Simulating the system and visualising the strategy simultaneously



ILP

Domain Knowledge Included

Rules
When this, do this….
(readable)



Decision Trees

Machine Learning

THANK YOU