Applications of inductive types in artificial intelligence and inductive reasoning

Ekaterina Komendantskaya

School of Computing, University of Dundee

Research Seminar in the University of Osnabrueck

Outline

Outline

Outline

Outline

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, golden medal for excellency).
I did my PhD in the UCC, Ireland (2004-2007). (The University of the Famous George Boole) My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning)
- Higher-order Interactive Theorem Provers
- Neuro-Symbolic networks
- Categorical Semantics of Computations

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, golden medal for excellency).
I did my PhD in the UCC, Ireland (2004-2007). (The University of the Famous George Boole) My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning) (PhD thesis (2007))
- Higher-order Interactive Theorem Provers
- Neuro-Symbolic networks
- Categorical Semantics of Computations

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, golden medal for excellency).
I did my PhD in the UCC, Ireland (2004-2007). (The University of the Famous George Boole) My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning) (PhD thesis (2007))
- Higher-order Interactive Theorem Provers (Postdoc in INRIA, France)
- Neuro-Symbolic networks

- Categorical Semantics of Computations

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, golden medal for excellency).
I did my PhD in the UCC, Ireland (2004-2007). (The University of the Famous George Boole) My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning) (PhD thesis (2007))
- Higher-order Interactive Theorem Provers (Postdoc in INRIA, France)
- Neuro-Symbolic networks (PhD Thesis, current EPSRC fellowship in Universities of St Andrews and Dundee, Scotland)
- Categorical Semantics of Computations

About myself

I did my undergraduate degree in Logic, Moscow State University (1998-2003); (1st class honours, golden medal for excellency).
I did my PhD in the UCC, Ireland (2004-2007). (The University of the Famous George Boole) My research interests can be classified into four main themes:

- Logic Programming (its applications in Artificial Intelligence and Automated reasoning) (PhD thesis (2007))
- Higher-order Interactive Theorem Provers (Postdoc in INRIA, France)
- Neuro-Symbolic networks (PhD Thesis, current EPSRC fellowship in Universities of St Andrews and Dundee, Scotland)
- Categorical Semantics of Computations (Research grant parallel to PhD and postdoc studies)

School of Computing, University of Dundee

- Assistive and healthcare technologies;
- Computational systems (Computer Vision; Theory of Argumentation (C. Reed));
- Interactive systems design;
- Space technology centre.

Computational Logic in Neural Networks

Symbolic Logic as Deductive System

- Deduction in logic calculi;
- Logic programming;
- Higher-order proof assistants...

Sound symbolic methods we can trust

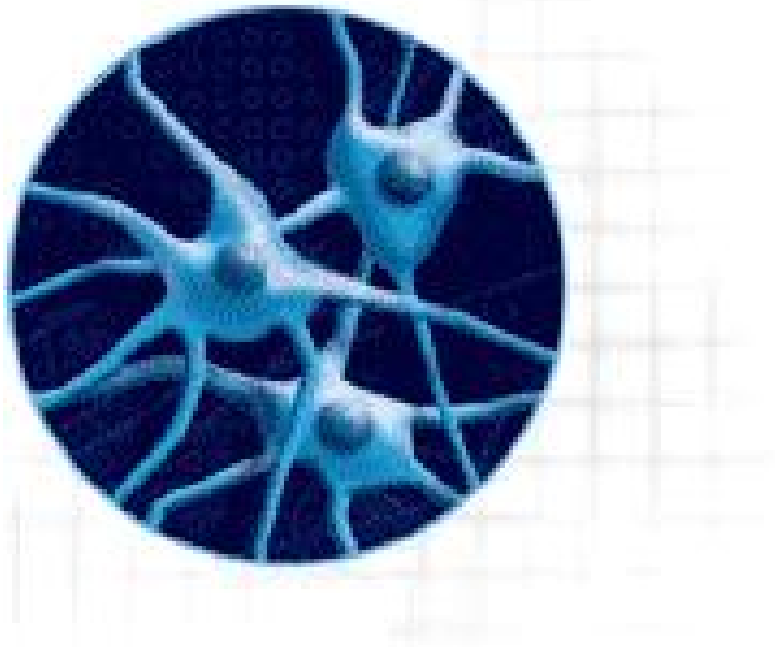
Computational Logic in Neural Networks

Symbolic Logic as Deductive System

- Deduction in logic calculi;
- Logic programming;
- Higher-order proof assistants...

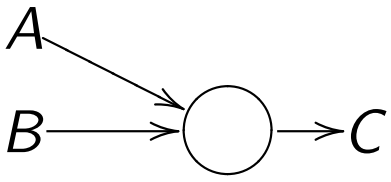
Sound symbolic methods we can trust

Neural Networks

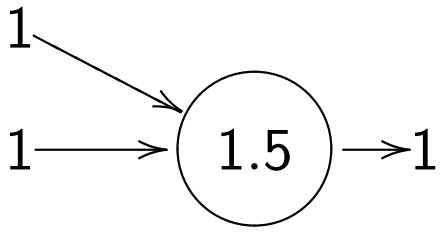


- spontaneous behavior;
- learning and adaptation;
- parallel computing.

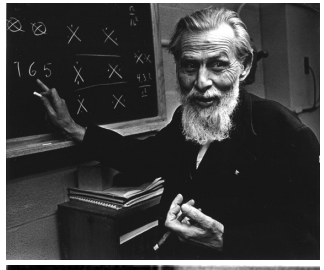
Boolean Networks of McCullogh and Pitts, 1943.

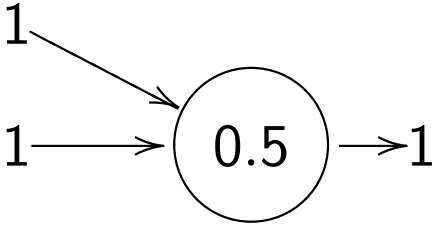


If A and B then C.

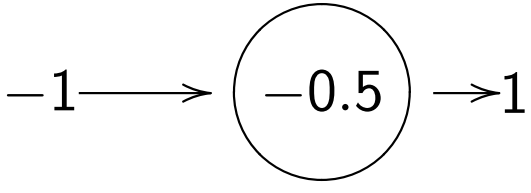


$(A = 1)$ and $(B = 1)$.





$(A = 1)$ or $(B = 1)$.



Not $(A = -1)$.

Neuro-symbolic architectures of other kinds based on the same methodology:

The approach of McCulloch and Pitts to processing truth values has dominated the area, and many modern neural network architectures consciously or unconsciously follow and develop this old method.

- Core Method: massively parallel way to compute minimal models of logic programs. [Holldobler et al, 1999 - 2009]
- Markov Logic and Markov networks: statistical AI and Machine learning implemented in NN. [Domingos et al., 2006-2009]
- Inductive Reasoning in Neural Networks [Broda, Garcez et al. 2002,2008]
- Fuzzy Logic Programming in Fuzzy Networks [Zadeh at al].

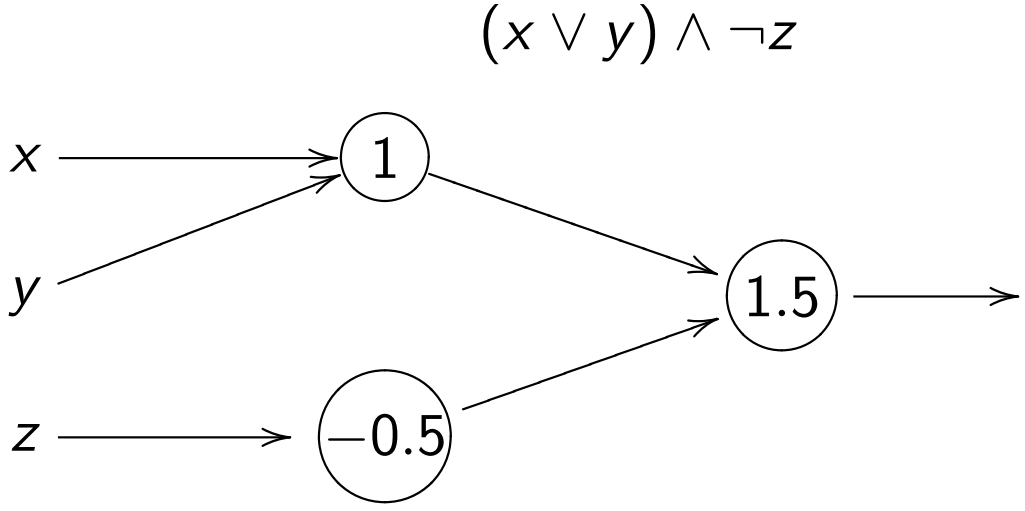
Markov Networks applied by [Domigos et al.]

Markov networks have been successfully applied in a variety of areas.

A system based on them recently won a competition on information extraction for biology. They have been successfully applied to problems in information extraction and integration, natural language processing, robot mapping, social networks, computational biology, and others, and are the basis of the open-source Alchemy system. Applications to Web mining, activity recognition, natural language processing, computational biology, robot mapping and navigation, game playing and others are under way.

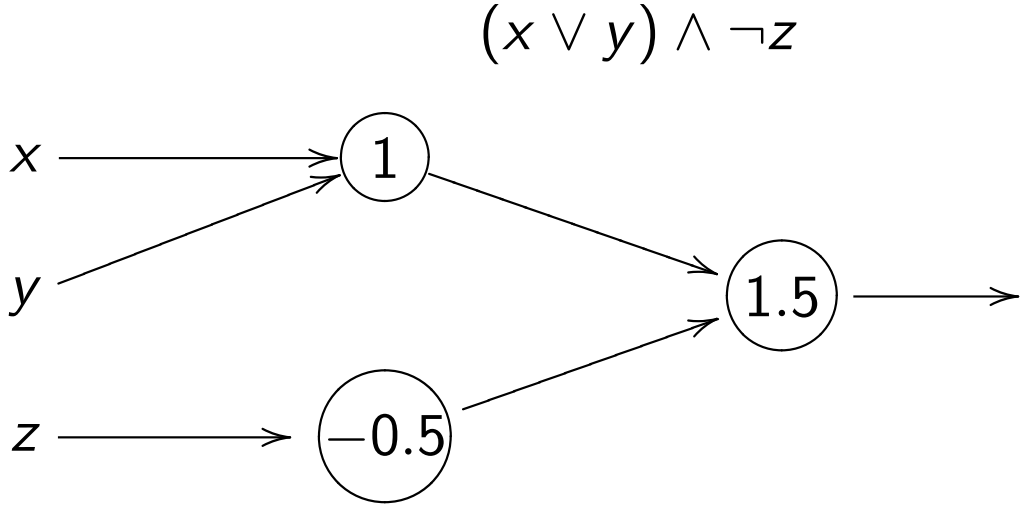
P. Domingos and D. Lowd. Markov Logic: An Interface Layer for Artificial Intelligence. San Rafael, CA: Morgan and Claypool, 2009.

How do we know that they are correct?



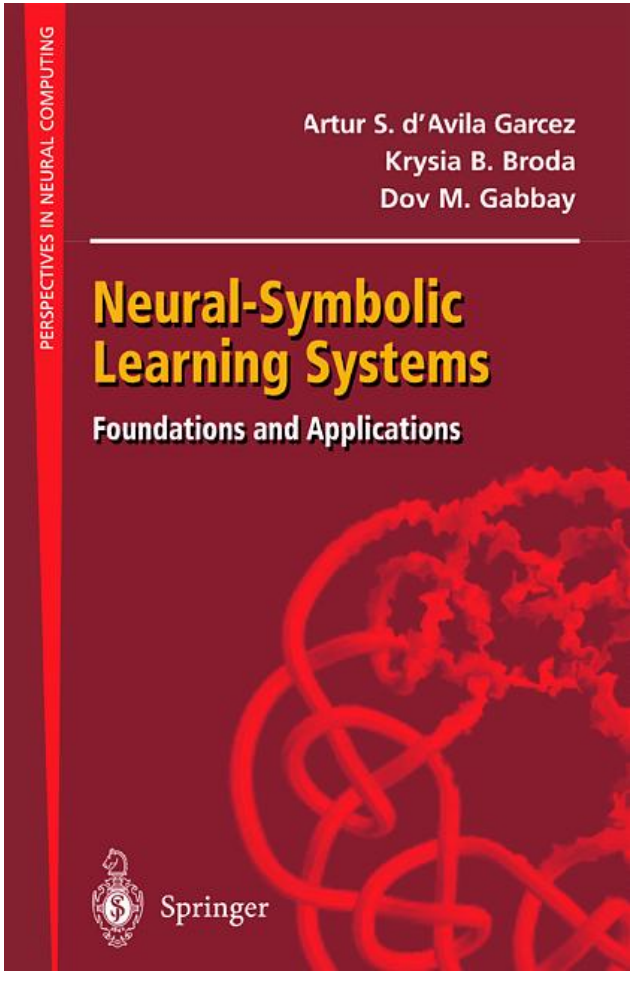
Such network would not distinguish "logical" data (values 0 and 1) from any other type of data, and would output the same result both for sound inputs like $x := 1, y := 1, z := 0$, and for non-logical values such as $x := 100.555, y := 200.3333\ldots, z := 0$. Imagine a user monitors the outputs of a big network, and sees outputs 1, standing for "true", whereas in reality the network is receiving some uncontrolled data.

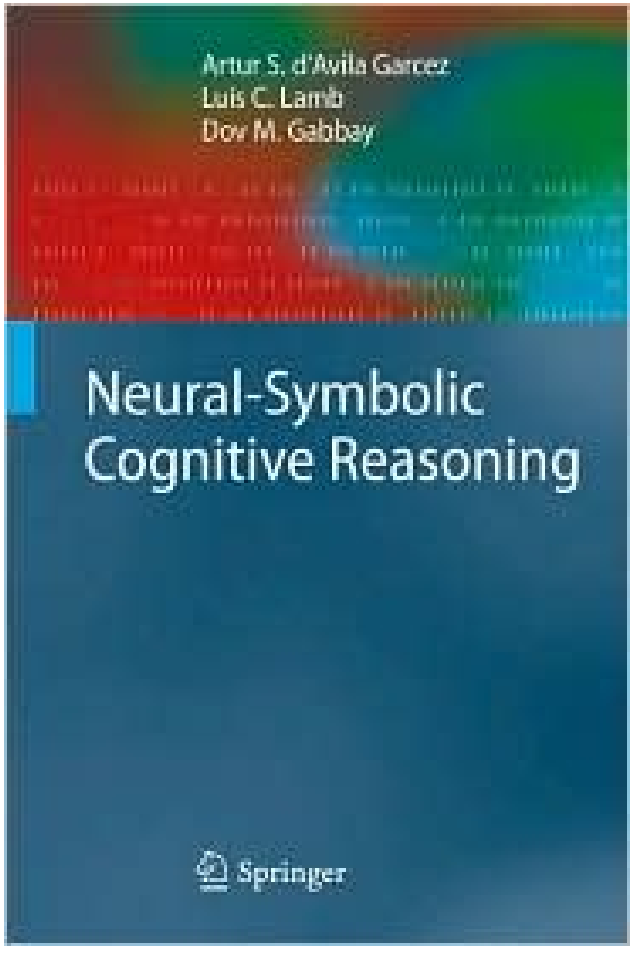
How do we know that they are correct?



Such network would not distinguish "logical" data (values 0 and 1) from any other type of data, and would output the same result both for sound inputs like $x := 1, y := 1, z := 0$, and for non-logical values such as $x := 100.555, y := 200.3333\ldots, z := 0$. Imagine a user monitors the outputs of a big network, and sees outputs 1, standing for "true", whereas in reality the network is receiving some uncontrolled data. The network gives correct answers on the condition that the input is well-typed.

Relational learning





Relational Reasoning and Learning.

In [Garcez et al, 2009], were built networks that can learn relations. E.g., given examples $Q(b, c) \rightarrow P(a, b)$ and $Q(d, e) \rightarrow P(c, d)$, they can infer a more general relation $Q(y, z) \rightarrow P(x, y)$.

Example

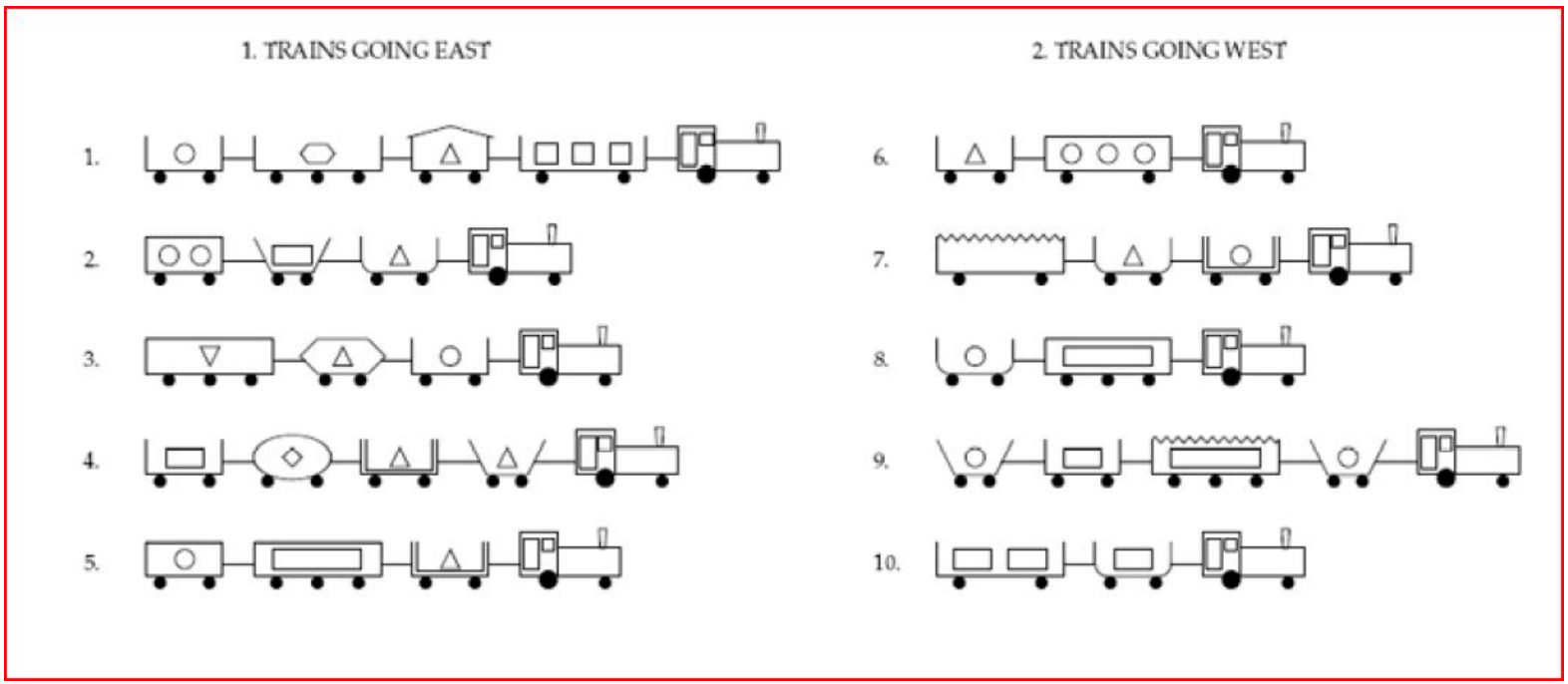
Learning a relation "grandparent" by examining families.
Classification of trains according to certain characteristics.

Relational Reasoning and Learning.

In [Garcez et al, 2009], were built networks that can learn relations. E.g., given examples $Q(b, c) \rightarrow P(a, b)$ and $Q(d, e) \rightarrow P(c, d)$, they can infer a more general relation $Q(y, z) \rightarrow P(x, y)$.

Example

Learning a relation "grandparent" by examining families.
Classification of trains according to certain characteristics.



Problems with this method

Such relational learning works as long as input data is well-typed.

"Well-typed" means that only related people, and not any other objects, are given to the network that learns relation "grandparent". And there are only trains of particular, known in advance, configuration, that are considered by the network that classifies trains.

This means that users have to make the preliminary classification and filtering of data before it is given to such networks; and NNs would not be able to warn the users if the data are ill-typed :-(.

Generally, as it turns out, typing is important for correct reasoning.

One can generalise from "This dog has four legs, and hence it can run" to "Everything that has four legs can run". However, we know that there are some objects, such as chairs, that have four legs but do not move. Hence we (often unconsciously) use typing in such cases, e.g., apply the generalisation only to all animals.

Analogical Reasoning and Types

Analogical reasoning is in reality closely connected to reasoning with types

We do not make analogies blindly, but we somehow filter certain object as suitable for analogical comparison, and some - not.

Analogical Reasoning and Types

Analogical reasoning is in reality closely connected to reasoning with types

We do not make analogies blindly, but we somehow filter certain object as suitable for analogical comparison, and some - not.

Coming back to the previous example

Taking two objects - a dog and a chair - we are unlikely to form any particularly useful kind of analogy. Unless we find a particular type of features that make the analogy useful...

Solutions: K.K., K. Broda, A.Garcez, to be presented at CiE'2010

Solution

As an alternative to the manual pre-processing of data, we propose neural networks that can do the same automatically. We use *neural networks called type recognisers*; and implement such networks to ensure the correctness of neural computations; both for classical cases (McCulloch & Pitts) and for the relational reasoning and learning.

Solutions: K.K., K. Broda, A.Garcez, to be presented at CiE'2010

Solution

As an alternative to the manual pre-processing of data, we propose neural networks that can do the same automatically. We use *neural networks called type recognisers*; and implement such networks to ensure the correctness of neural computations; both for classical cases (McCulloch & Pitts) and for the relational reasoning and learning.

The solution involves techniques like pattern-matching, inductive type definitions, etc. that are used in functional programming, type theory, and interactive theorem provers!

The main result

First ever method of using Types for ensuring the correctness of Neural or Neuro-Symbolic computations.

Theorem

For any a type A, given an expression E presented in a form of a numerical vector, we can construct a neural network that recognises whether E is of type A.

Such networks are called Type recognisers, and for each given type A, the network that recognises A is called an A-recogniser. This construction covers simple types, such as Bool, as well as more complex inductive types, such as natural numbers, lists; or even dependent inductive types, such as lists of natural numbers.

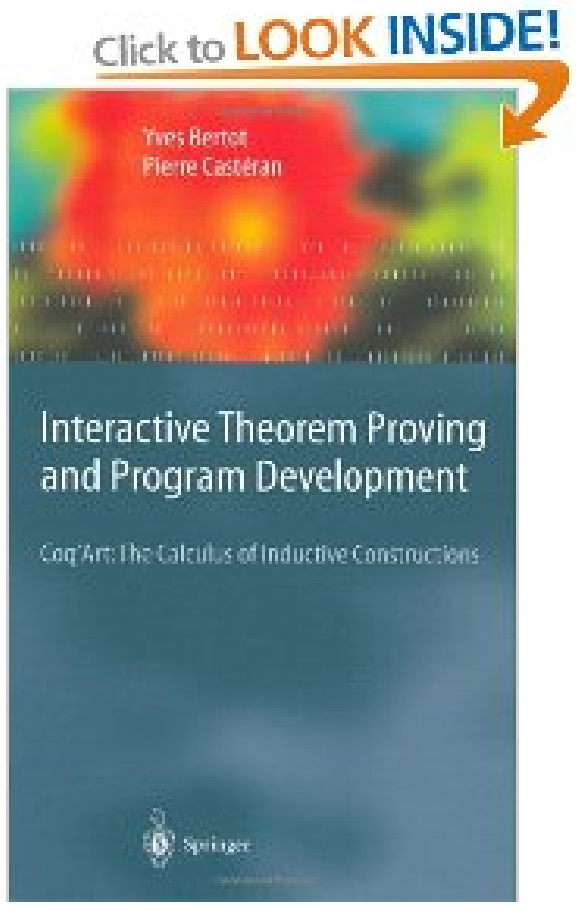
Source books for reading on Interactive Theorem Provers

Programming in Martin-Löf's Type Theory. An Introduction
by
Bengt Nordström, Kent Petersson, Jan M. Smith

http://www.cse.chalmers.se/research/group/logic/book/

Source books for reading on Interactive Theorem Provers

Coq Art by
Bertot and Casteran.



Programming in Martin-Löf's Type Theory. An Introduction by
Bengt Nordström, Kent Petersson, Jan M. Smith

http://www.cse.chalmers.se/research/group/logic/book/

Some examples of inductive types

Primitive:

```
Inductive bool :  Type :=  | t :  bool
 | f :  bool.
```

Some examples of inductive types

Primitive:

```
Inductive bool :  Type :=  | t :  bool
 | f :  bool.
```

Recursive:

```
Inductive nat :  Set :=
 | O : nat
 | S : nat -> nat.
```

Typical element of the set `SSSO`.

Some examples of inductive types

Primitive:

```
Inductive bool :  Type :=  | t :  bool
 | f :  bool.
```

Recursive:

```
Inductive nat :  Set :=
 | O : nat
 | S : nat -> nat.
```

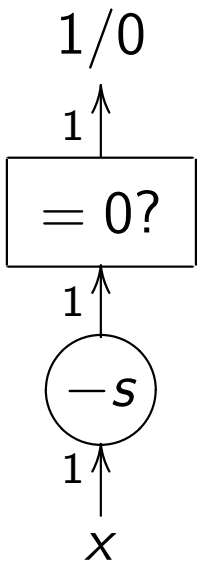
Typical element of the set `SSSO`.

Dependent:

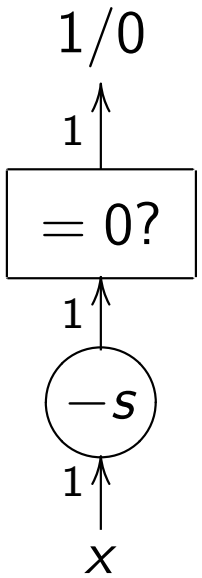
```
Inductive list :  Set :=
 | nil :  list
 | cons :  nat -> list -> list.
```

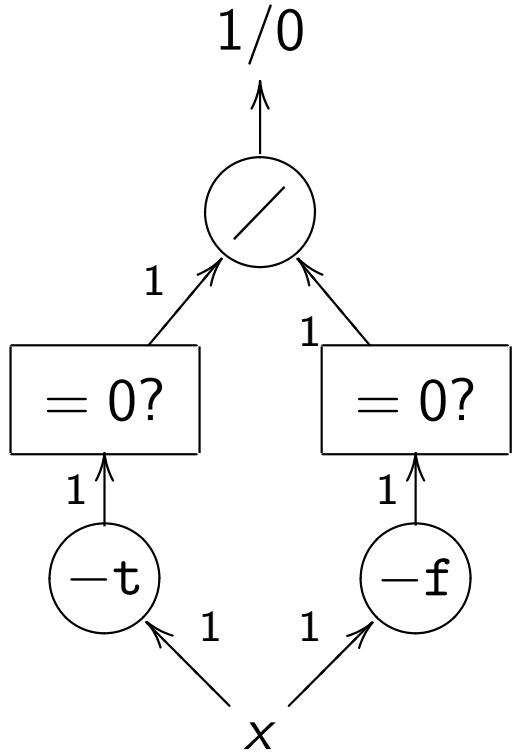
Typical element of the set is `O::SO::SSSO::O::nil` also written `cons O cons SO cons SSSO cons O cons nil`.

"Atomic" symbol recognisers

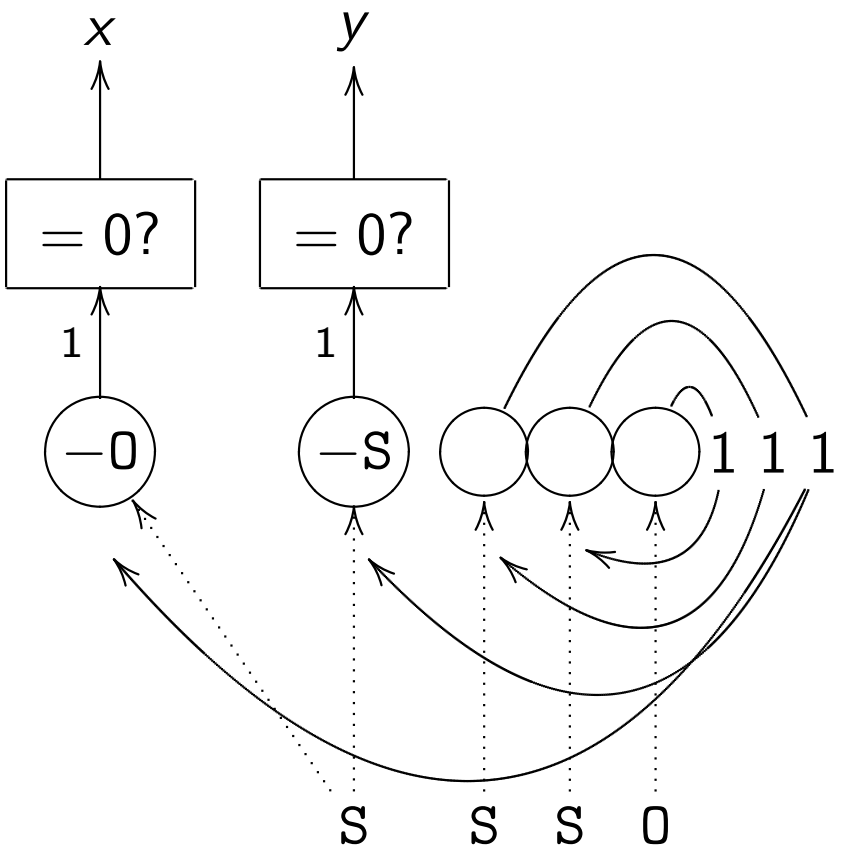


Inductive recogniser for bool



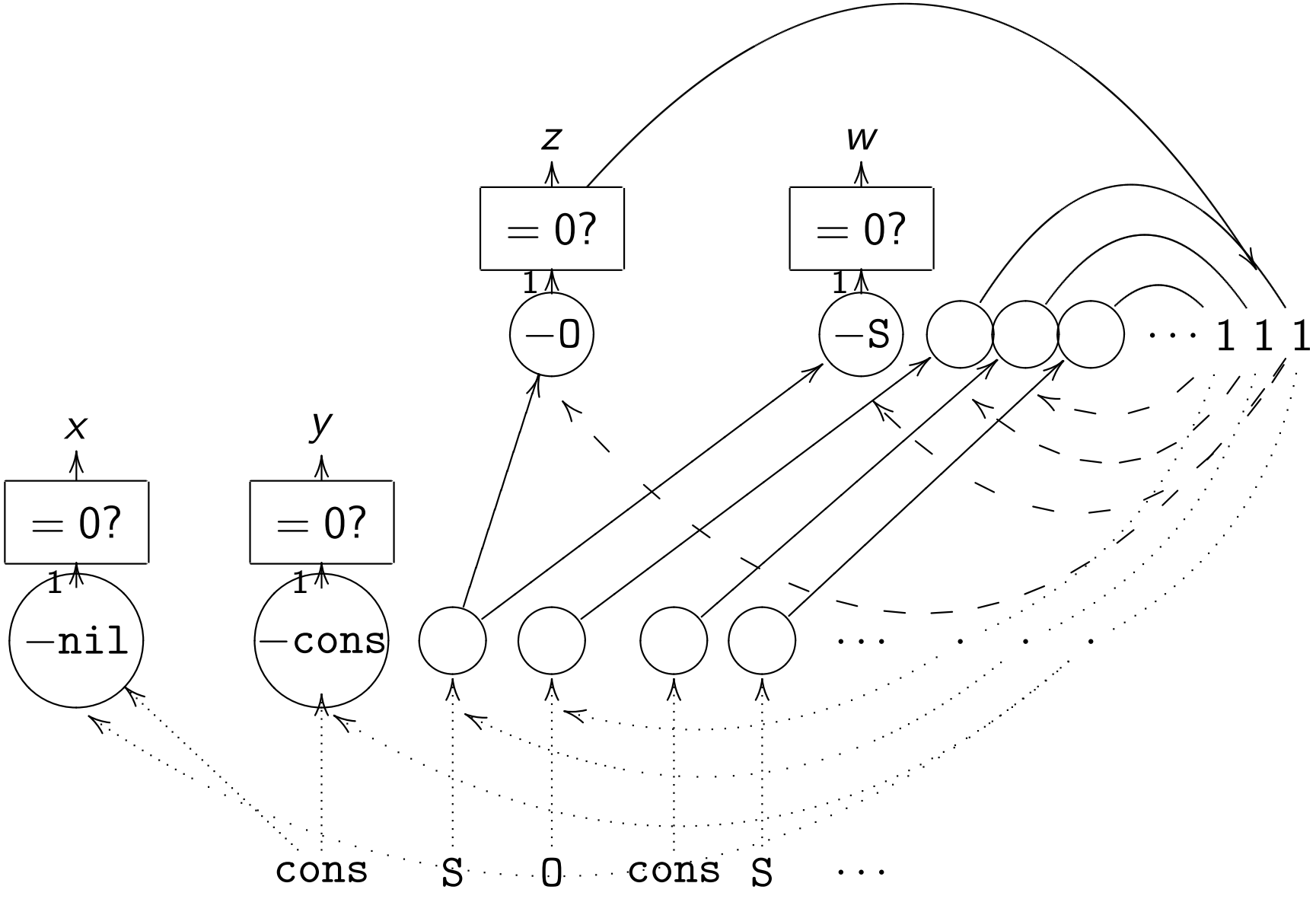


Inductive recogniser for `nat`

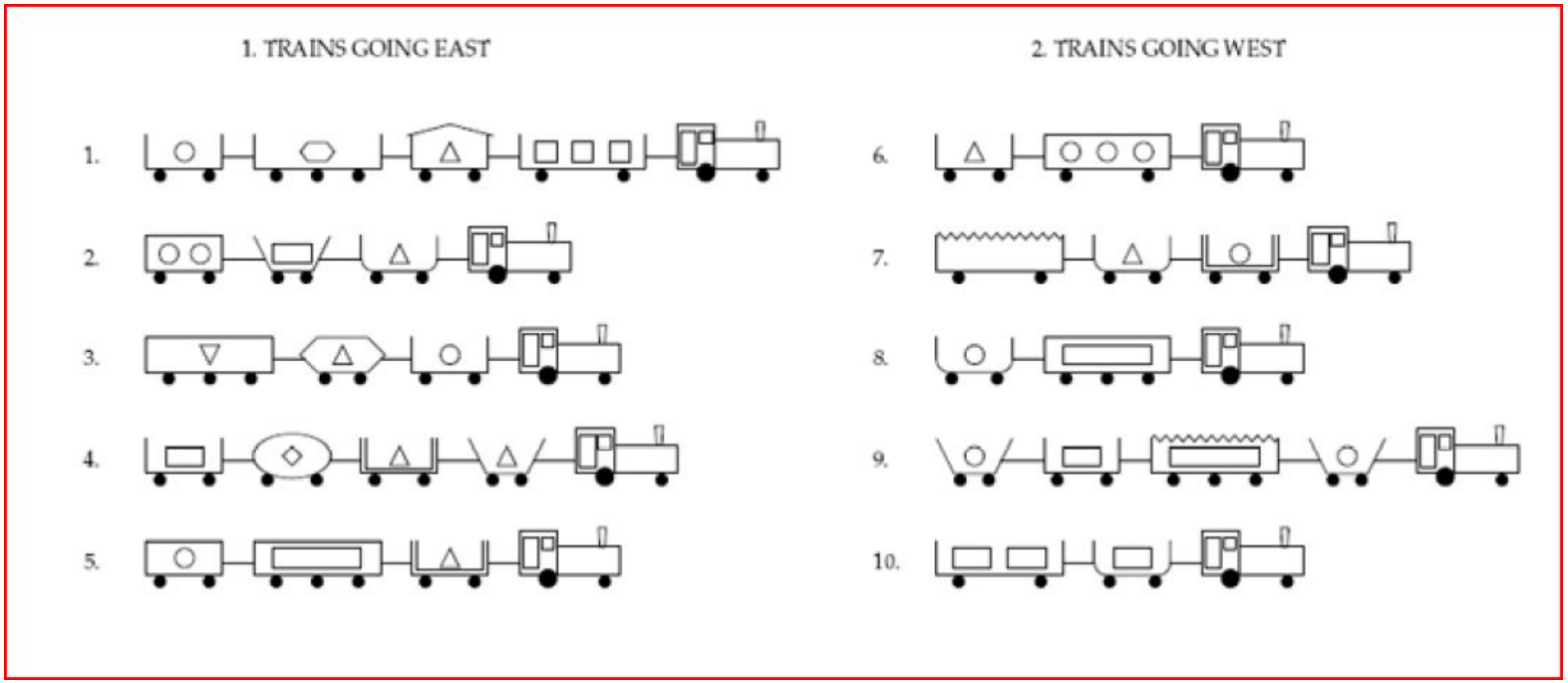


x	y	
1	0	- success
0	1	- working
1	1	- impossible
0	0	- failure

Inductive recogniser for `list nat`



Example with trains




```
Inductive shape :  Type :=
 | oval :  shape
 | triangle :  shape.
```

```
Inductive direction :  Set :=
 | west :  direction
 | east :  direction.
```

Example with trains

```
Definition train :  Set :=
< n:nat, m:nat, s:  list shape, t:  list shape>
```

```
Function t:train :  dir := match t with <n,m,s,t>
 if n = 3 and m = 2 and (exists n', nth n' s = s' and
s = triangle) => west
if ...
```

Inductive types in Neuro-symbolic networks

Inductive types can naturally be represented in neural networks:

- For finite sets, we use feed-forward networks
- For infinite sets defined recursively we use recursive connections in networks
- The networks can be integrated into big Neuro-symbolic systems to type-check inputs/outputs;
- The networks can be used for inductive generalisations and analogy. (connection to Osbnabrueck research?)
- Also note their relaton to coalgebra (possible connection to [Kai-Uwe et al.] topos-theoretic approach)

Logic programs

Logic Programs

- $A \leftarrow B_1, \ldots, B_n$

Logic programs

Logic Programs

- $A \leftarrow B_1, \ldots, B_n$
- $T_P(I) = \{A \in B_P :$
  $A \leftarrow B_1, \ldots, B_n$
  is a ground instance of a clause in P and $\{B_1, \ldots, B_n\} \subseteq I\}$

Logic programs

Logic Programs

- $A \leftarrow B_1, \ldots, B_n$
- $T_P(I) = \{A \in B_P :$
  $A \leftarrow B_1, \ldots, B_n$
  is a ground instance of a clause in P and $\{B_1, \ldots, B_n\} \subseteq I\}$
- $\mathtt{lfp}(T_P \uparrow \omega) =$ the least Herbrand model of P.

An Important Result, [Kalinke, Hölldobler, 94]

For each propositional program P, there exists a 3-layer feedforward neural network which computes T_P.

We will call such neural networks T_P-neural networks.

Characteristic Properties of T_P-Neural Networks

Characteristic Properties of T_P-Neural Networks

1. The number of neurons in the input and output layers is the number of atoms in the Herbrand base B_P of a given program P.

Characteristic Properties of T_P-Neural Networks

1. The number of neurons in the input and output layers is the number of atoms in the Herbrand base B_P of a given program P.
2. First-order atoms are not presented in the neural network directly, and only truth values 1 and 0 are propagated.

Characteristic Properties of T_P-Neural Networks

1. The number of neurons in the input and output layers is the number of atoms in the Herbrand base B_P of a given program P.
2. First-order atoms are not presented in the neural network directly, and only truth values 1 and 0 are propagated.
3. Arise from Boolean networks of McCulloch and Pitts;

Characteristic Properties of T_P-Neural Networks

1. The number of neurons in the input and output layers is the number of atoms in the Herbrand base B_P of a given program P.
2. First-order atoms are not presented in the neural network directly, and only truth values 1 and 0 are propagated.
3. Arise from Boolean networks of McCulloch and Pitts;
4. Require infinitely long layers in the first-order case.

A Simple Example

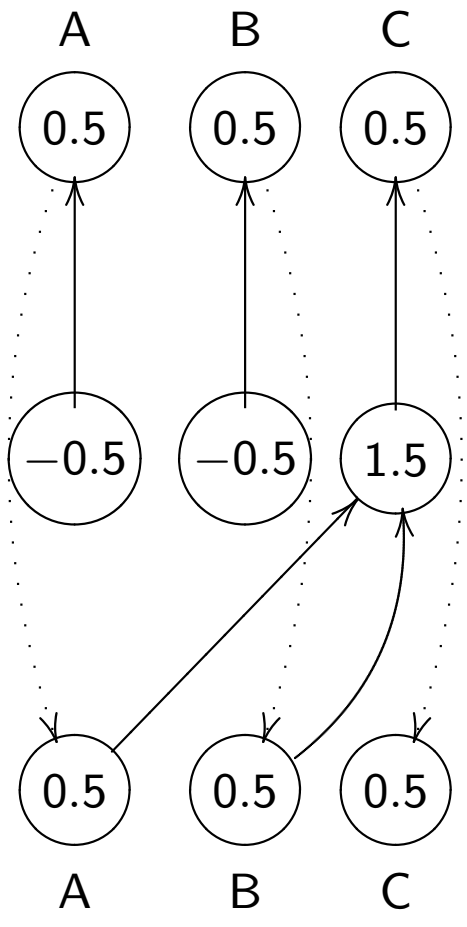
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$

A Simple Example

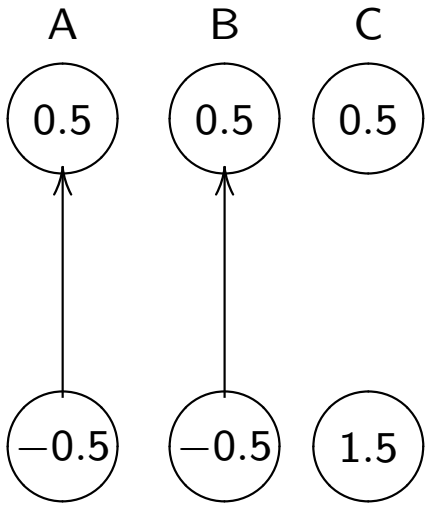
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$

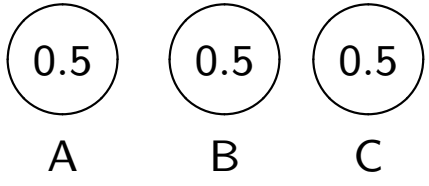
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



A Simple Example

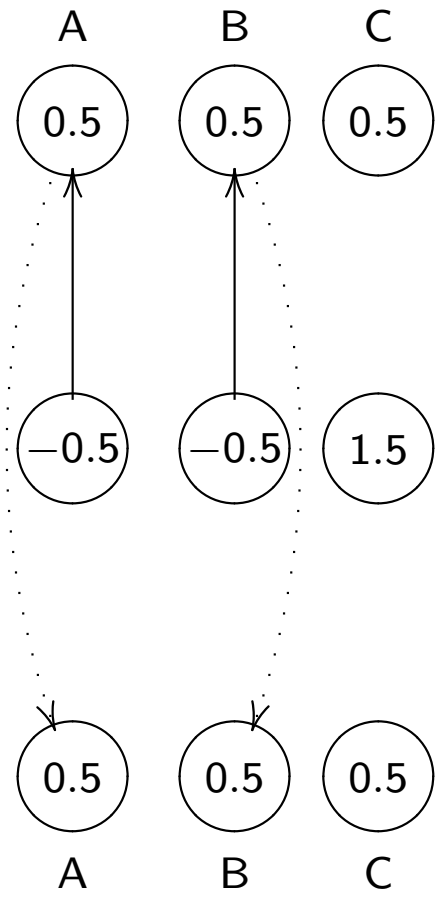
$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$





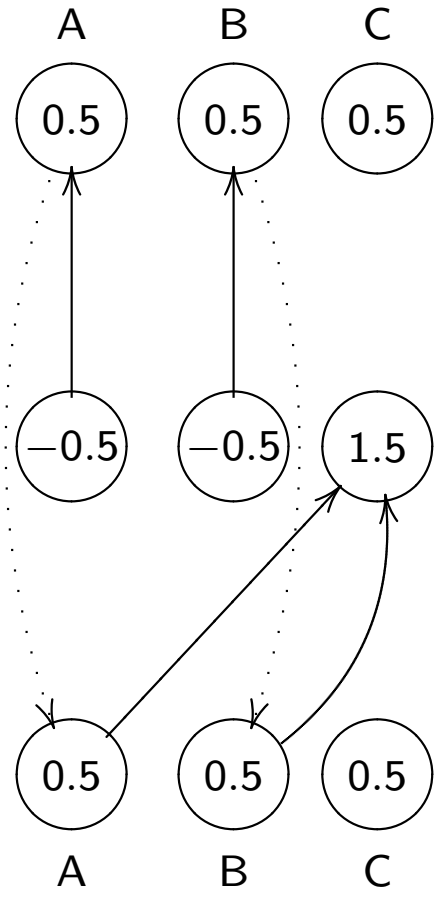
A Simple Example

$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



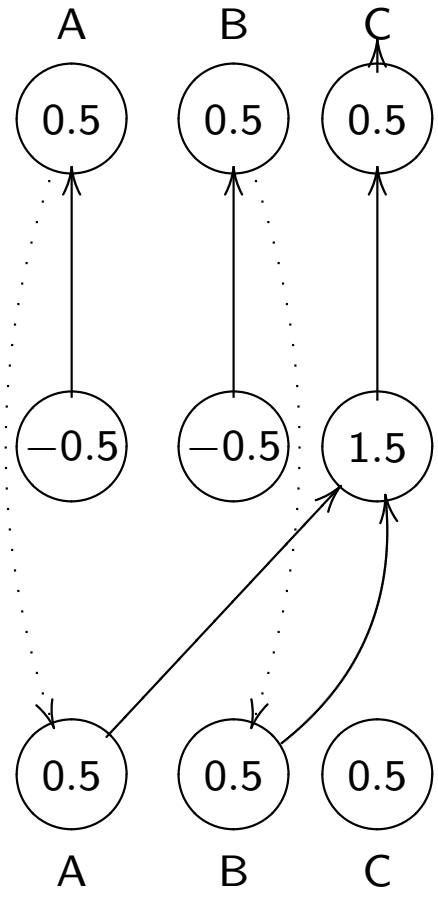
A Simple Example

$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



A Simple Example

$B \leftarrow$
$A \leftarrow$
$C \leftarrow A, B$
$T_P \uparrow 0 = \{B, A\}$
$lfp(T_P) = T_P \uparrow 1 = \{B, A, C\}$



Example 2

$P(0) \leftarrow$
$P(s(x)) \leftarrow P(x)$

$T_P \uparrow 0 = \{P(0)\}$
$lfp(T_P) = T_P \uparrow \omega =$
$\{0, s(0), s(s(0)),$
$s(s(s(0))), \ldots\}$

Example 2

$P(0) \leftarrow$
$P(s(x)) \leftarrow P(x)$

$T_P \uparrow 0 = \{P(0)\}$
$lfp(T_P) = T_P \uparrow \omega =$
$\{0, s(0), s(s(0)),$
$s(s(s(0))), \ldots\}$

Example 2

$P(0) \leftarrow$
$P(s(x)) \leftarrow P(x)$

$T_P \uparrow 0 = \{P(0)\}$
$lfp(T_P) = T_P \uparrow \omega =$
$\{0, s(0), s(s(0)),$
$s(s(s(0))), \ldots\}$
Paradox:
(computability,
complexity,
proof theory)

Solution

Solution to the problem can be found in the approach known as Logic Programs as Inductive Definitions. Consider the logic programs below:

```
bool(t) <-
bool(f) <-
```

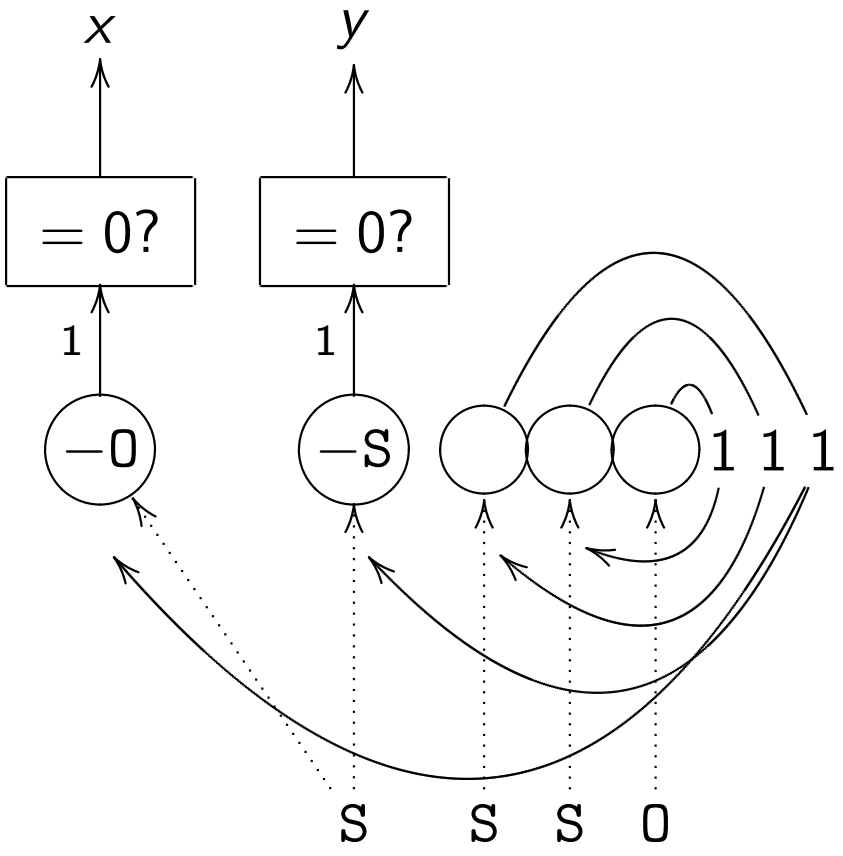
```
nat(0) <-
nat(S(n)) <- nat(n)
```

```
list(nil) <-
list(cons(n,s)) <- nat(n), list(s)
```

It turns out that most of "problematic" implementations of Neuro-Symbolic systems relate to the recursive structures.

Neural networks for inductive logic programs

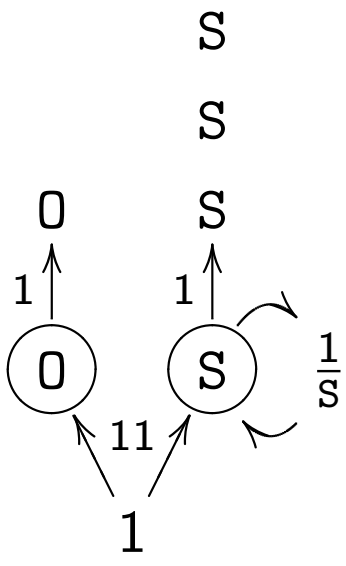
We can use precisely the same networks to handle inductive logic programs:



x	y	
1	0	- success
0	1	- working
1	1	- impossible
0	0	- failure

Relation to Logic programming semantics

The inductive definition of `nat` is built on the assumption that the set of natural numbers is computed at the least fixed point. This gives rise to two common applications for inductive definitions - they can be used to *generate* the elements of a set - if they are read from right to left; and they can be used for type-checking of expressions - if they are read from left to right. Both kinds of implementation require finite and terminating computations.




```
nat(0) <-
nat(S(n)) <- nat(n)
```

Conclusions

- Types and type-theoretic approach has a big future in AI: be it inductive reasoning, learning techniques, or neuro-symbolic integration.
- Inductive types are closely related to recursive structures that arise in Neural networks;
- Inductive types should be used to ensure safety and security of Neuro-Symbolic networks;
- Finally, they can be used to improve the performance of the exisiting state-of-the-art Neuro-Symbolic Systems.

Thank you!