From ProbLog to CompLog

From Probabilistic to Complexity-based Logic Programming

1st October 2023, HYDRA workshop @ ECAI 2023

2nd International Workshop on Hybrid Models for coupling Deductive and Inductive Reasoning

Giovanni Sileno
g.sileno@uva.nl

University of Amsterdam

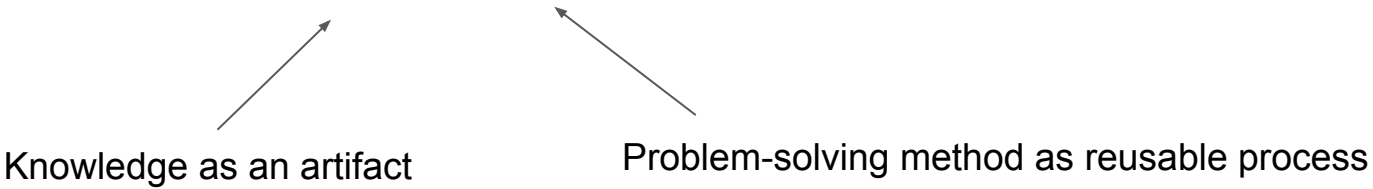
Jean-Louis Dessalles
jean-louis.dessalles@telecom-paris.fr

Télécom Paris

Logic programming

- Logic programming is a form of declarative programming introduced to **reproduce formal reasoning mechanisms.**

Algorithm = Logic + Control [Kowalski, 1979]

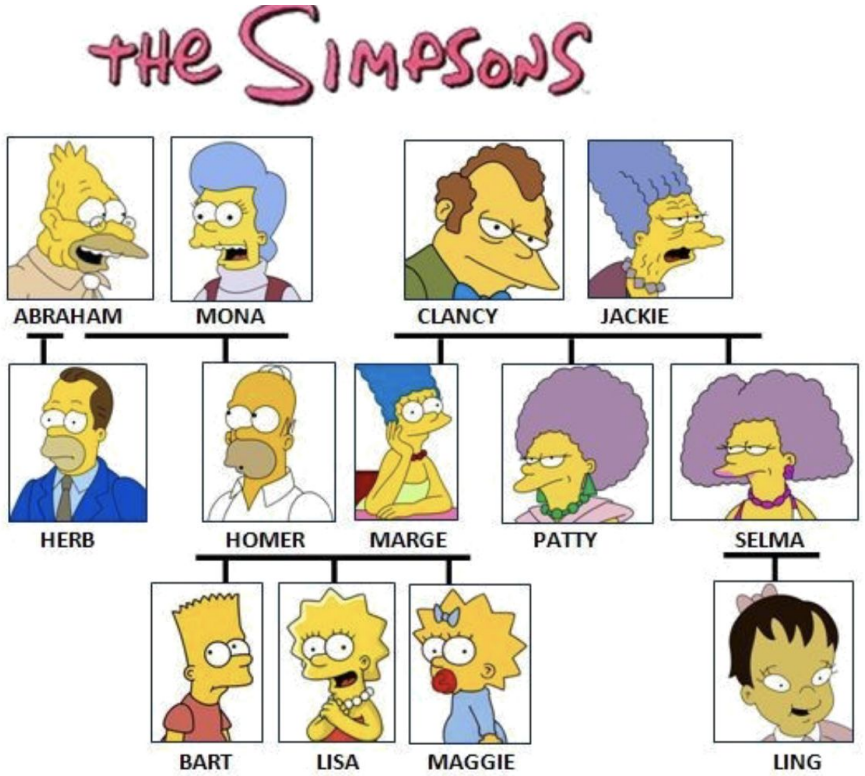


Example of Prolog program

```
parent(marge, lisa).
parent(marge, bart).
parent(marge, maggie).
parent(homer, lisa).
parent(homer, bart).
parent(homer, maggie).
parent(abraham, homer).
parent(abraham, herb).
parent(mona, homer).
parent(jackie, marge).
parent(clancy, marge).
parent(jackie, patty).
parent(clancy, patty).
parent(jackie, selma).
parent(clancy, selma).
parent(selma, ling).

child(X,Y) :- parent(Y,X).

?- child(lisa, marge).
true
```



Extension to probabilistic programs

ProbLog: A Probabilistic Prolog and its Application in Link Discovery

Luc De Raedt*, **Angelika Kimmig and Hannu Toivonen**†
Machine Learning Lab, Albert-Ludwigs-University Freiburg, Germany

Abstract

We introduce ProbLog, a probabilistic extension of Prolog. A ProbLog program defines a distribution over logic programs by specifying for each clause the probability that it belongs to a randomly sampled program, and these probabilities are mutually independent. The semantics of ProbLog is then defined by the success probability of a query, which corresponds to the probability that the query succeeds in a randomly sampled program. The key contribution of this paper is the intr
effective solver for computing suc
ties. It essentially combines SLD-r
methods for computing the probabi
formulae. Our implementation fu
an approximation algorithm that cor
deepening with binary decision dia
port on experiments in the context
links in real biological networks, a
of the practical usefulness of the ap

it seems that there are – despite a great potential – still only few real-life applications of these probabilistic logics. The reasons for this might well be that the assumptions are too strong and sometimes hard to manage by the user, or that the solvers are often too slow or too limited.

We introduce ProbLog which is – in a sense – the simplest probabilistic extension of Prolog one can design. ProbLog is essentially Prolog where all clauses are labeled with the probability that they are true, and – similar as pD but unlike the other approaches mentioned – these probabilities are mutually independent. ProbLog has been motivated by the real-life

Neural probabilistic logic programming in DeepProbLog ☆

Robin Manhaeve [a,*], Sebastijan Dumančić [a], Angelika Kimmig [a], Thomas Demeester [b,1], Luc De Raedt [a,1]

[a] KU Leuven, Belgium
[b] Ghent University - imec, Belgium

ARTICLE INFO

Article history:
Received 26 July 2019
Received in revised form 1 April 2021
Accepted 3 April 2021
Available online 15 April 2021

Keywords:
Logic

ABSTRACT

We introduce DeepProbLog, a neural probabilistic logic programming language that incorporates deep learning by means of neural predicates. We show how existing inference and learning techniques of the underlying probabilistic logic programming language ProbLog can be adapted for the new language. We theoretically and experimentally demonstrate that DeepProbLog supports (i) both symbolic and subsymbolic representations and inference, (ii) program induction, (iii) probabilistic (logic) programming, and (iv) (deep) learning from examples. To the best of our knowledge, this work is the first to propose

Bayesian Synthesis of Probabilistic Programs for Automatic Data Modeling

FERAS A. SAAD, Massachusetts Institute of Technology, USA
MARCO F. CUSUMANO-TOWNER, Massachusetts Institute of Technology, USA
ULRICH SCHAECHTLE, Massachusetts Institute of Technology, USA
MARTIN C. RINARD, Massachusetts Institute of Technology, USA
VIKASH K. MANSINGHKA, Massachusetts Institute of Technology, USA

We present new techniques for automatically constructing probabilistic programs for data analysis, interpretation, and prediction. These techniques work with probabilistic domain-specific data modeling languages that capture key properties of a broad class of data generating processes, using Bayesian inference to synthesize probabilistic programs in these modeling languages given observed data. We provide a precise formulation of Bayesian synthesis for automatic data modeling that identifies sufficient conditions for the resulting synthesis procedure to be sound. We also derive a general class of synthesis algorithms for domain-specific

DeepStochLog: Neural Stochastic Logic Programming

Thomas Winters*[1], Giuseppe Marra*[1], Robin Manhaeve[1], and Luc De Raedt[1,2]

[1]KU Leuven, Dept. of Computer Science; Leuven.AI, B-3000 Leuven, Belgium,
{firstname.lastname}@kuleuven.be
[2]AASS, Örebro University, Sweden

Abstract

Recent advances in neural symbolic learning, such as DeepProbLog, extend probabilistic logic programs with neural predicates. Like graphical models, these probabilistic logic programs define a probability distribution over possible worlds, for which inference is computationally hard. We propose DeepStochLog, an alternative neural symbolic framework based on stochastic definite clause grammars, a type of stochastic logic program, which defines a probability distribution over possible derivations. More specifically, we introduce neural grammar rules

Example of ProbLog program

```
0.5 :: friendof(john, mary).
0.5 :: friendof(mary, pedro).
0.5 :: friendof(mary, tom).
0.5 :: friendof(pedro, tom).

1.0 :: likes(X, Y) :- friendof(X, Y).
0.8 :: likes(X, Y) :- friendof(X, Z), likes(Z, Y).

evidence(likes(mary, tom)).
query(likes(mary, pedro)).


probability: 0.58333333
```

Example of ProbLog program

```
0.5 :: friendof(john, mary).
0.5 :: friendof(mary, pedro).
0.5 :: friendof(mary, tom).
0.5 :: friendof(pedro, tom).

1.0 :: likes(X, Y) :- friendof(X, Y).
0.8 :: likes(X, Y) :- friendof(X, Z), likes(Z, Y).

evidence(likes(mary, tom)).
query(likes(mary, pedro)).
```

```
probability: 0.58333333
```

Two interpretations of the output:

- Mary likes Pedro a bit
- It is slightly more probable that Mary likes Pedro

Example of ProbLog program

```
0.5 :: friendof(john, mary).
0.5 :: friendof(mary, pedro).
0.5 :: friendof(mary, tom).
0.5 :: friendof(pedro, tom).

1.0 :: likes(X, Y) :- friendof(X, Y).
0.8 :: likes(X, Y) :- friendof(X, Z), likes(Z, Y).

evidence(likes(mary, tom)).
query(likes(mary, pedro)).
```

```
probability: 0.58333333
```

matter of **description**
(cf. fuzzy logics)

Two interpretations of the output:

- Mary likes Pedro a bit
- It is slightly more probable that Mary likes Pedro

matter of **world** (possible extractions)

Example of ProbLog program

```
0.5 :: friendof(john, mary).
0.5 :: friendof(mary, pedro).
0.5 :: friendof(mary, tom).
0.5 :: friendof(pedro, tom).

1.0 :: likes(X, Y) :- friendof(X, Y).
0.8 :: likes(X, Y) :- friendof(X, Z), likes(Z, Y).

evidence(likes(mary, tom)).
query(likes(mary, pedro)).
```

```
probability: 0.58333333
```

Can we truly pass from one to the other with no issue?

matter of **description** (cf. fuzzy logics)

Two interpretations of the output:

- Mary likes Pedro a bit
- It is slightly more probable that Mary likes Pedro

matter of **world** (possible extractions)

A fundamental distinction!

Epistemic uncertainty

(not been unveiled/proven yet)

Ontological uncertainty

(not been created/extracted yet)

A fundamental distinction!

Epistemic uncertainty
(not been unveiled/proven yet) → matter of conditions holding in the world

Ontological uncertainty
(not been created/extracted yet) → matter of events occurring in the world

A fundamental distinction!

Epistemic uncertainty
(not been unveiled/proven yet) → matter of conditions holding in the world

Ontological uncertainty
(not been created/extracted yet) → matter of events occurring in the world

The passage from conditions to events is a known challenge in AI!

Conditions vs Events (1)

- In symbolic AI, it became soon clear that it is rather difficult to directly reason about events by means of deduction.
- Several axiomatizations have been proposed, the most known being *Situation Calculus, Event Calculus, Fluent Calculus.*

Yale shooting problem

Conditions vs Events (1)

- In symbolic AI, it became soon clear that it is rather difficult to directly reason about events by means of deduction.
- Several axiomatizations have been proposed, the most known being *Situation Calculus, Event Calculus, Fluent Calculus.*

Yale shooting problem

```
% event calculus axioms
holds(F, T) :- holds(F, 0), not clipped(0, F, T).
holds(F, T2) :- occurs(E, T1), initiates(E, F, T1), T1 < T2, not clipped(T1, F, T2).
clipped(T1, F, T2) :- occurs(E, T), T1 <= T, T < T2, terminates(E, F, T).
```

Conditions vs Events (1)

- In symbolic AI, it became soon clear that it is rather difficult to directly reason about events by means of deduction.
- Several axiomatizations have been proposed, the most known being *Situation Calculus, Event Calculus, Fluent Calculus.*

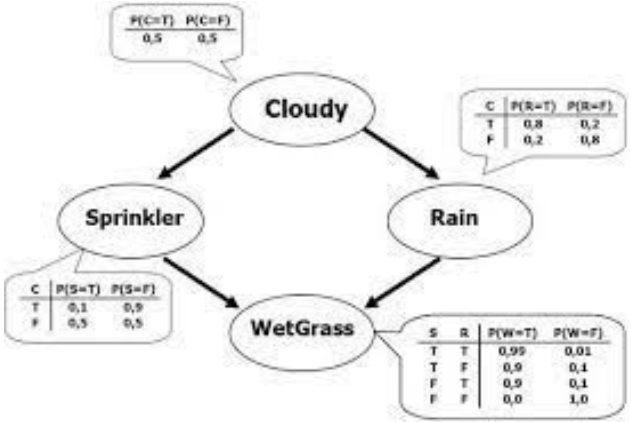
Yale shooting problem

meta-level predicates about two different types of entities: fluents and events

```
% event calculus axioms
holds(F, T) :- holds(F, 0), not clipped(0, F, T).
holds(F, T2) :- occurs(E, T1), initiates(E, F, T1), T1 < T2, not clipped(T1, F, T2).
clipped(T1, F, T2) :- occurs(E, T), T1 <= T, T < T2, terminates(E, F, T).
```

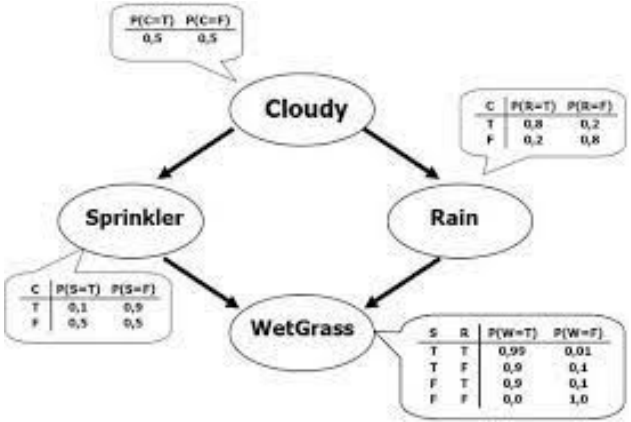
Conditions vs Events (2)

- Formally, Bayesian Networks capture only associationistic relationships. The causal reading exists only in the mind of the modeler.



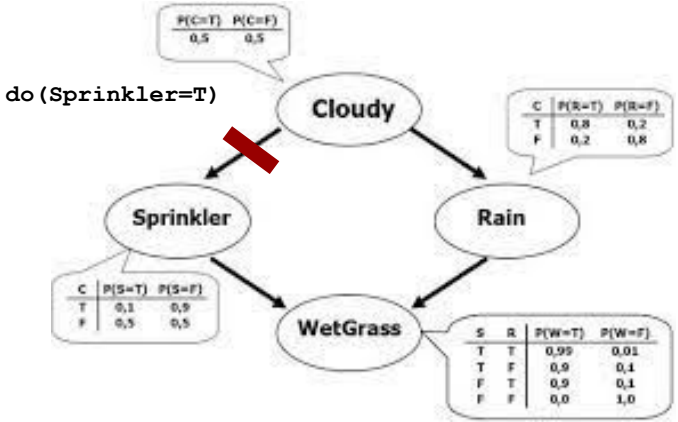
Conditions vs Events (2)

- Formally, Bayesian Networks capture only associationistic relationships. The causal reading exists only in the mind of the modeler.
- We need to consider a **do** operator to take into account *interventions*.



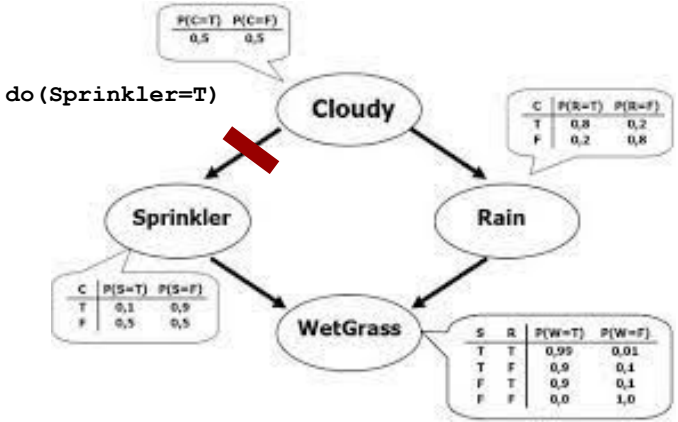
Conditions vs Events (2)

- Formally, Bayesian Networks capture only associationistic relationships. The causal reading exists only in the mind of the modeler.
- We need to consider a **do** operator to take into account *interventions*.
- The **do** operator provides local conterfactuality by **performing operations** on the Bayesian network: *it cuts the nodes which are parent to the intervened node.*



Conditions vs Events (2)

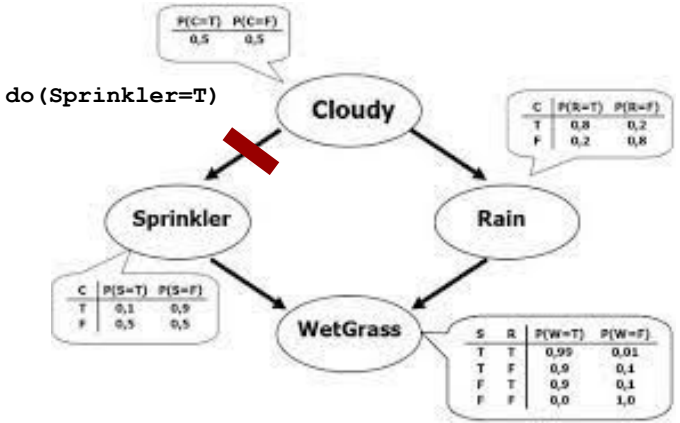
- Formally, Bayesian Networks capture only associationistic relationships. The causal reading exists only in the mind of the modeler.
- We need to consider a **do** operator to take into account *interventions*.
- The **do** operator provides local conterfactuality by **performing operations** on the Bayesian network: *it cuts the nodes which are parent to the intervened node.*



at a meta-level again!

Conditions vs Events (2)

- Formally, Bayesian Networks capture only associationistic relationships. The causal reading exists only in the mind of the modeler.
- We need to consider a **do** operator to take into account *interventions*.
- The **do** operator provides local conterfactuality by **performing operations** on the Bayesian network: *it cuts the nodes which are parent to the intervened node.*



at a meta-level again!

It seems we should not interchange the two dimensions so easily…

Looking for alternatives...

- **Simplicity Theory** (ST) is a computational model of cognition which explicitly relies on two different machines: one for the “world” (making events occurring), and one for the “mind” (determining/unveiling conditions)!

Looking for alternatives...

- **Simplicity Theory** (ST) is a computational model of cognition which explicitly relies on two different machines: one for the “world” (making events occurring), and one for the “mind” (determining/unveiling conditions)!

➡ **It offers a more principled solution to separate ontological/epistemic uncertainty!**

Simplicity Theory: Motivation

According to Shannon's **theory of information**:

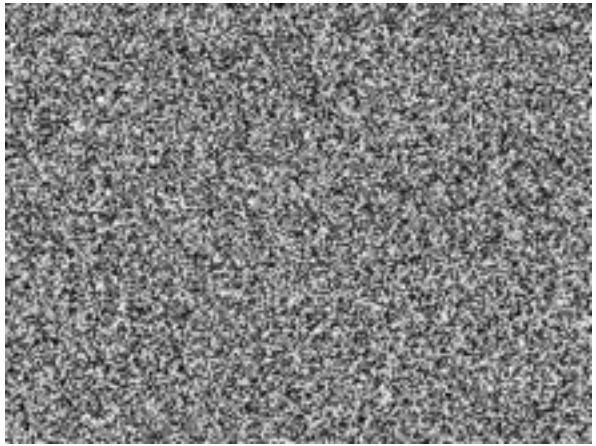
$$I(x) = -\log P(x)$$

Simplicity Theory: Motivation

According to Shannon's **theory of information**:

$$I(x) = -\log P(x)$$

NOISE SOURCE
maximally informative



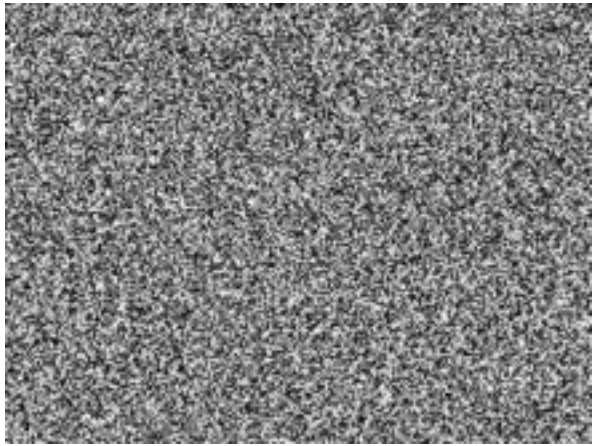
Simplicity Theory: Motivation

According to Shannon's **theory of information**:

$$I(x) = -\log P(x)$$

NOISE SOURCE
maximally informative

but this mismatches
relevance judgments
given by humans



SO WHAT?

Simplicity Theory: Unexpectedness

- In contrast to standard information theory, ST starts from the observation that humans are highly susceptible to *complexity drops*, ie. for them

  situations are ***relevant*** if they are *simpler* **to describe** than **to explain**

Simplicity Theory: Unexpectedness

- In contrast to standard information theory, ST starts from the observation that humans are highly susceptible to *complexity drops*, ie. for them

  situations are ***relevant*** if they are *simpler* **to describe** than **to explain**

- Formally, this is captured by the formula of **unexpectedness**, expressed as divergence of complexity computed on two distinct machines

$$U(s) = C_W(s) - C_D(s)$$

causal complexity
via world machine

description complexity
via description machine

Simplicity Theory: Unexpectedness

- In contrast to standard information theory, ST starts from the observation that humans are highly susceptible to *complexity drops*, ie. for them

  situations are ***relevant*** if they are *simpler* **to describe** than **to explain**

- Formally, this is captured by the formula of **unexpectedness**, expressed as divergence of complexity computed on two distinct machines

$$U(s) = C_W(s) - C_D(s)$$

causal complexity
via world machine

description complexity
via description machine

$$\overbrace{\text{world} \rightarrow \text{situation}}^{C_W}$$

$$\overbrace{\text{situation} \leftarrow \text{mind}}^{C_D}$$

Simplicity Theory: Unexpected

- In contrast to standard information theory, ST starts from t
  humans are highly susceptible to *complexity drops*, ie. for

  situations are ***relevant*** if they are *simpler* **to describe** th

- Formally, this is captured by the formula of **unexpectedne**
  divergence of complexity computed on two distinct machir

"Delphin island" in Sardinia

$$U(s) = C_W(s) - C_D(s)$$

causal complexity *via world machine*

description complexity *via description machine*

$$\overbrace{\text{world} \rightarrow \text{situation}}^{C_W}$$

$$\overbrace{\text{situation} \leftarrow \text{mind}}^{C_D}$$

Simplicity Theory: Unexpected

- In contrast to standard information theory, ST starts from t
  humans are highly susceptible to *complexity drops*, ie. for

  situations are ***relevant*** if they are *simpler* **to describe** th

- Formally, this is captured by the formula of **unexpectedne**
  divergence of complexity computed on two distinct machi

"Delphin island" in Sardinia

$$U(s) = C_W(s) - C_D(s)$$

situation

causal complexity *via world machine*

description complexity *via description machine*

$$\overbrace{\text{world} \rightarrow \text{situation}}^{C_W}$$

$$\overbrace{\text{situation} \leftarrow \text{mind}}^{C_D}$$

Simplicity Theory: Unexpected

- In contrast to standard information theory, ST starts from t
  humans are highly susceptible to *complexity drops*, ie. for

  situations are ***relevant*** if they are *simpler* **to describe** th

- Formally, this is captured by the formula of **unexpectedne**
  divergence of complexity computed on two distinct machi

"Delphin island" in Sardinia

$$U(s) = C_W(s) - C_D(s)$$

situation

causal complexity
via world machine

description complexity
via description machine

$$\overbrace{\text{world} \rightarrow \text{situation}}^{C_W}$$

$$\overbrace{\text{situation} \leftarrow \text{mind}}^{C_D}$$

functionally similar to **posterior subjective probability**

How to compute complexity?

- In algorithmic information theory (AIT), the *complexity* of a string is the minimal length of a program that, given a certain optional input parameter, produces that string as an output (**Kolmogorov complexity**)

$$K_\phi(x|y) = \min_p \{|p| : p(y) = x\}$$

executable program

target string

additional input in support

underlying
Turing machine

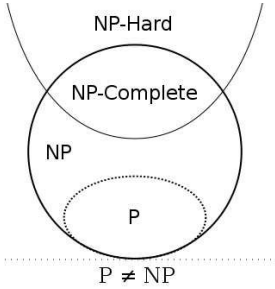
How to compute complexity?

- In algorithmic information theory (AIT), the *complexity* of a string is the minimal length of a program that, given a certain optional input parameter, produces that string as an output (**Kolmogorov complexity**)

how much information is needed for a program constructing the object (Kolmogorov complexity)

$\neq$

how much time or space is needed for running it (algorithmic or time-complexity)



$$K_\phi(x|y) = \min_p \{|p| : p(y) = x\}$$

executable program

target string

additional input in support

underlying Turing machine

How to compute complexity?

- In algorithmic information theory (AIT), the *complexity* of a string is the minimal length of a program that, given a certain optional input parameter, produces that string as an output (**Kolmogorov complexity**)

$$K_\phi(x|y) = \min_p \{|p| : p(y) = x\}$$

executable program

underlying Turing machine

target string

additional input in support

- Kolmogorov complexity is generally incomputable (due to the halting problem), but it is computable on ***bounded Turing machines***.

We denote bounded complexities with C

From ProbLog to CompLog: intuition

- Rather than computing probability, we compute unexpectedness by measuring complexity on two different machines, causal and descriptive.

$$U(s) = C_W(s) - C_D(s)$$

causal complexity
via world machine

description complexity
via description machine

From ProbLog to CompLog: intuition

- Rather than computing probability, we compute unexpectedness by measuring complexity on two different machines, causal and descriptive.

$$U(s) = C_W(s) - C_D(s)$$

causal complexity
via world machine

description complexity
via description machine

- For simple machines, the computation of Kolmogorov complexity can be seen as min-path search on graphs:

$$K_\phi(x|y) = \min_p \{|p| : p(y) = x\}$$

underlying Turing machine

target string

additional input in support

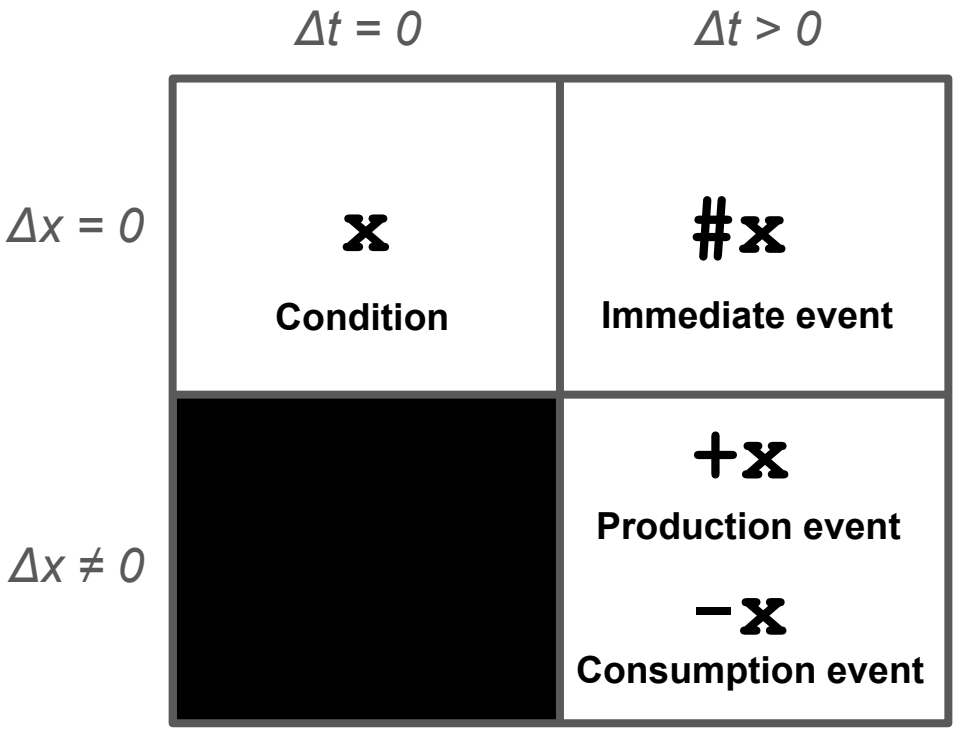
executable program

CompLog: main characteristics

CompLog: main characteristics (1)

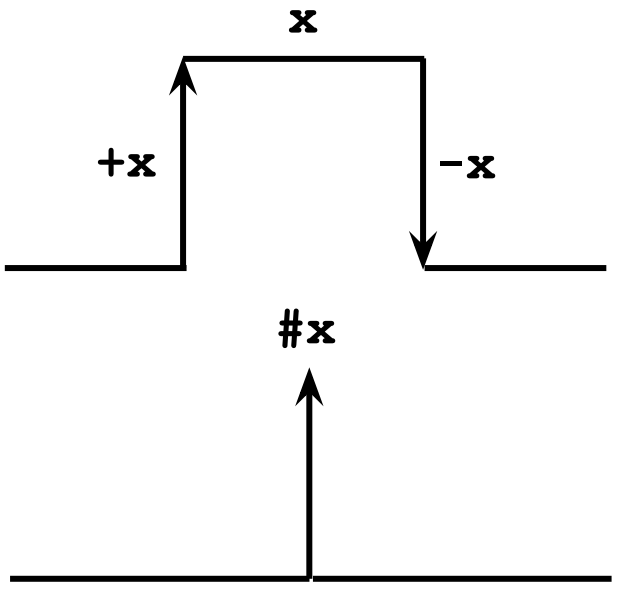
- A complexity-based program consists of a world model (with **causal relationships**, centered on ***events***), and a mental model (with **associationistic relationships**, including logical, centered on ***conditions***).

Syntax for conditions/events



Syntax for conditions/events

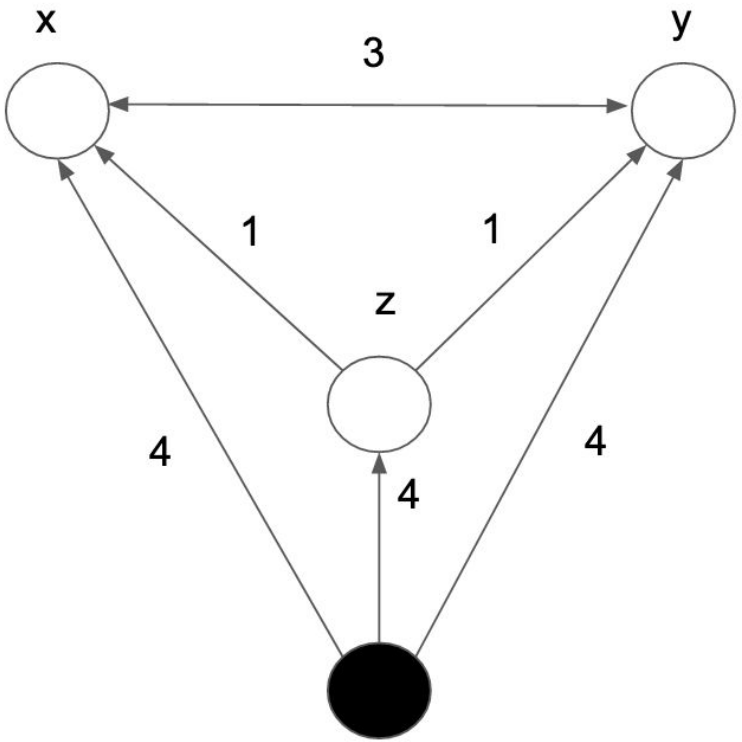
	$\Delta t = 0$	$\Delta t > 0$
$\Delta x = 0$	**x** Condition	**#x** Immediate event
$\Delta x \neq 0$		**+x** Production event **−x** Consumption event



CompLog: main characteristics (2)

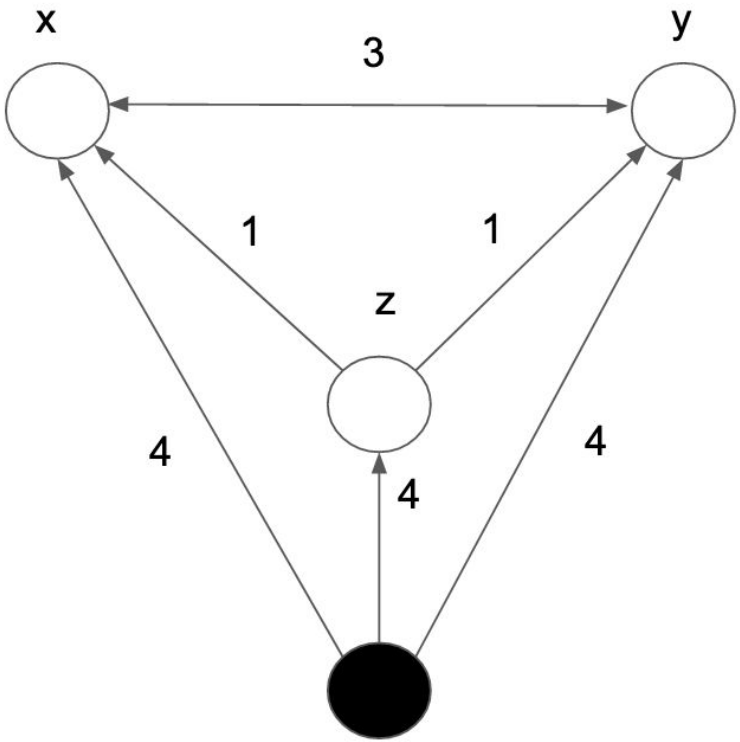
- A complexity-based program consists of a world model (with **causal relationships**, centered on ***events***), and a mental model (with **associationistic relationships**, including logical, centered on ***conditions***).
- The world and mental models can be represented as two networks, with different search algorithms, connecting to two different characterization of computation
    - **Productive** vs **Epistemic**

Productive vs epistemic



- Let us consider computation as a binary **colouring** task on a graph
- Queries are expressed as goal nodes, possibly with an order (*)

Productive vs epistemic

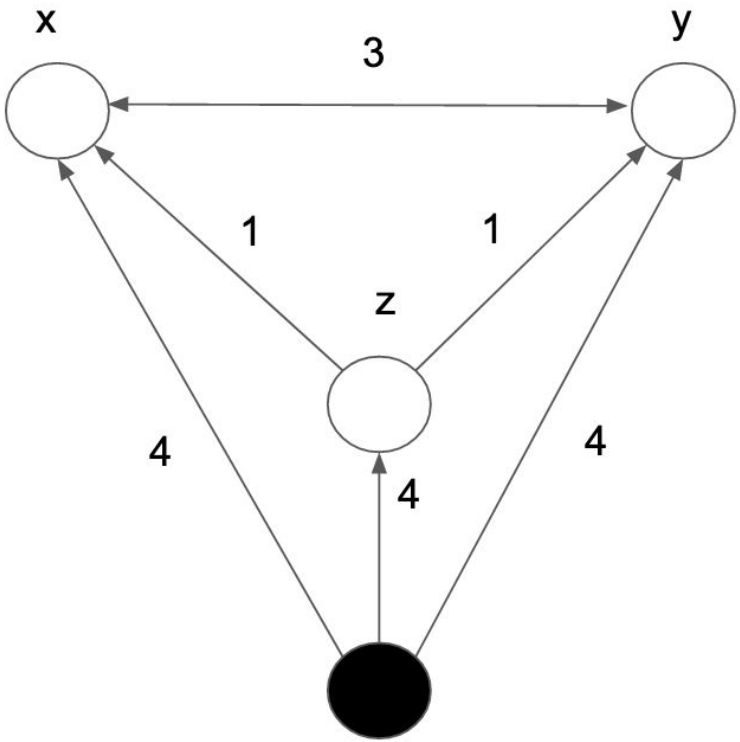


- Let us consider computation as a binary **colouring** task on a graph
- Queries are expressed as goal nodes, possibly with an order (`*`)

Epistemic
(no consumption)

```
C(x) = 4
C(x * y) = 7
C(<x, y>) = 6
```

Productive vs epistemic



- Let us consider computation as a binary **colouring** task on a graph
- Queries are expressed as goal nodes, possibly with an order (`*`)

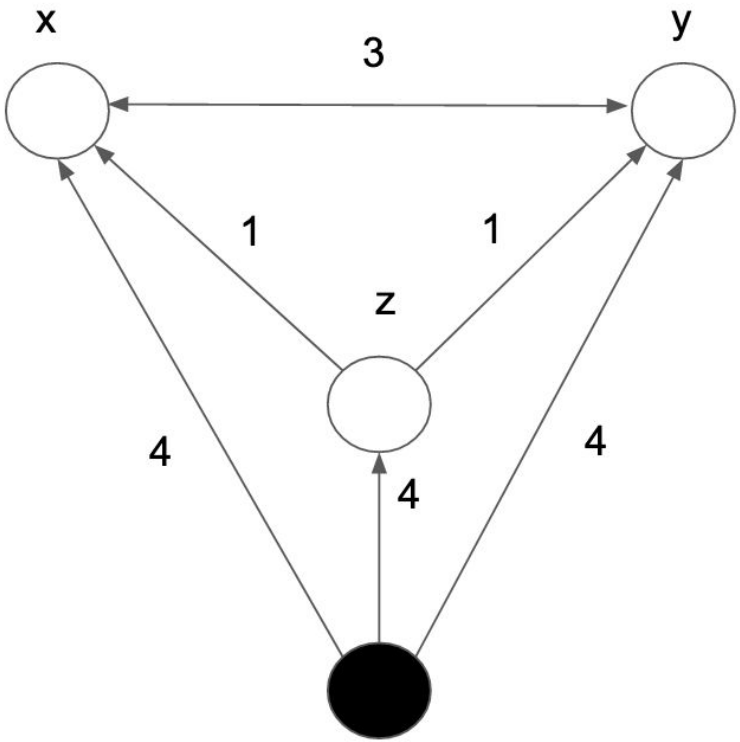
Epistemic
(no consumption)

```
C(x) = 4
C(x * y) = 7
C(<x, y>) = 6
```

Productive
(with consumption)

```
C(#x) = 4
C(#x * #y) = 7
C(<#x, #y>) = 7
```

Productive vs epistemic



- Let us consider computation as a binary **colouring** task on a graph
- Queries are expressed as goal nodes, possibly with an order (`*`)

what about catalysts?

Epistemic
(no consumption)

```
C(x) = 4
C(x * y) = 7
C(<x, y>) = 6
```

Productive
(with consumption)

```
C(#x) = 4
C(#x * #y) = 7
C(<#x, #y>) = 7
```

From declarative to active rules

- declarative rule (*if condition then conclusion*)

```
x -> y.    % prolog/ASP style y :- x.
```

From declarative to active rules

- declarative rule (*if condition then conclusion*)

```
x -> y.    % prolog/ASP style y :- x.
```

- active rule (*if antecedent then consequent*)

```
#push => +light.
```

```
#x => #y.
+x => +y. % with no consumption
+x => +y, -x . % with consumption
```

From declarative to active rules

- declarative rule (*if condition then conclusion*)

```
x -> y.    % prolog/ASP style y :- x.
```

- active rule (*if antecedent then consequent*)

```
#push => +light.
```

```
#x => #y.
+x => +y. % with no consumption
+x => +y, -x . % with consumption
```

- active rule in ECA template (*when event in condition then action*):

```
#push : electricity => +light.
```

Program augmentation

- Given a declarative program….

```
x.
y.
z.
z -> x.
z -> y.
x -> y.
y -> x.
```

Program augmentation

- Given a declarative program….

```
x.
y.
z.
z -> x.
z -> y.
x -> y.
y -> x.
```

- We can transform it in an active program with race conditions:

```
=> +x.
=> +y.
=> +z.
+z => +x.
+z => +y.
+x => +y.
+y => +x.
```

Program augmentation

- Given a declarative program….

```
x.
y.
z.
z -> x.
z -> y.
x -> y.
y -> x.
```

- We can transform it in an active program with race conditions:

```
=> +x.
=> +y.
=> +z.
+z => +x.
+z => +y.
+x => +y.
+y => +x.
```

or with no race conditions:

```
: => +x.
: => +y.
: => +z.
: z => +x.
: z => +y.
: x => +y.
: y => +x.
```

Program augmentation

- Given a declarative program….

```
x.
y.
z.
z -> x.
z -> y.
x -> y.
y -> x.
```

- We can transform it in an active program with race conditions:

```
=> +x.
=> +y.
=> +z.
+z => +x.
+z => +y.
+x => +y.
+y => +x.
```

or with no race conditions:

```
: => +x.
: => +y.
: => +z.
: z => +x.
: z => +y.
: x => +y.
: y => +x.
```

here we capture the **catalyst** case, and the relation with *ergodicity* (as asymptotic growth)

CompLog

web editor >

code

```
4 :: x.
4 :: y.
4 :: z.
z -> 1 :: x.
z -> 1 :: y.
x -> 3 :: y.
y -> 3 :: x.
```

14

Parse | Separate | Augment | Reset

See examples …

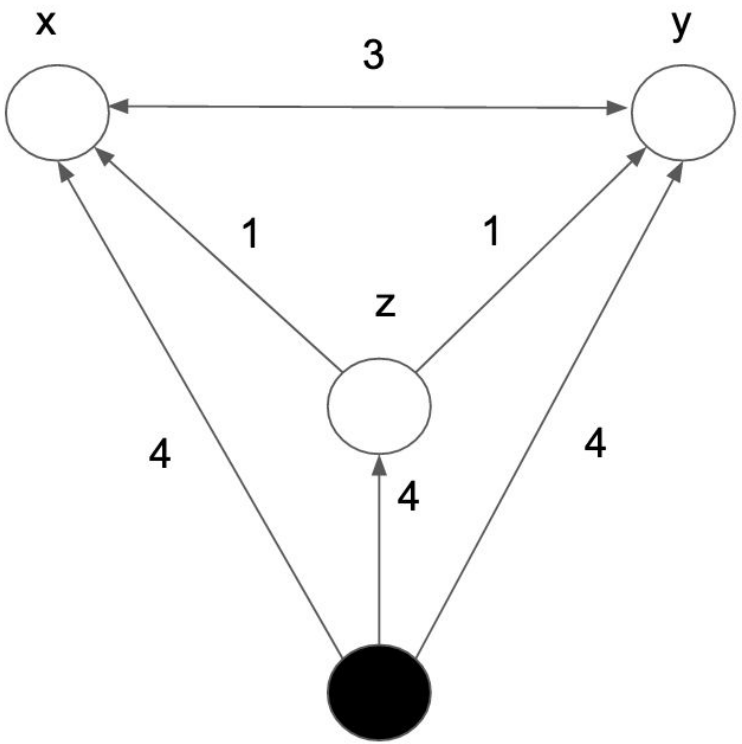
output

```
% generative model constructed from declarative model
=> 4 :: +x.
=> 4 :: +y.
=> 4 :: +z.
: z => 1 :: +x.
: z => 1 :: +y.
: x => 3 :: +y.
: y => 3 :: +x.
```

CompLog: main characteristics (3)

- A complexity-based program consists of a world model (with **causal relationships**), and a mental model (with **associationistic relationships**, including logical).
- The two models can be represented as two networks, with different search algorithms, connecting to two different characterization of computations
    - Productive characterization: with consumption of resources
    - Epistemic characterization: no consumption of resources
- **Unexpectedness** (related to posterior probability) becomes the primary target of inference. Priors are derived by adding back the determination cost.

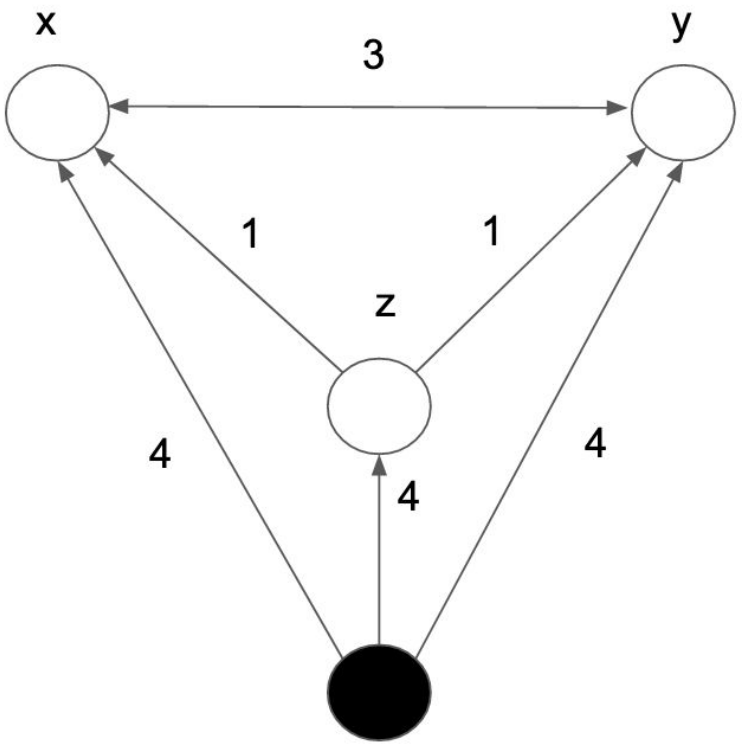
Example program




```
4 :: x.
4 :: y.
4 :: z.
1 :: z -> x.
1 :: z -> y.
3 :: x -> y.
3 :: y -> x.
```

```
Cd(x) = 4
Cd(x * y) = 7
Cd(<x, y>) = 6
```

Example program



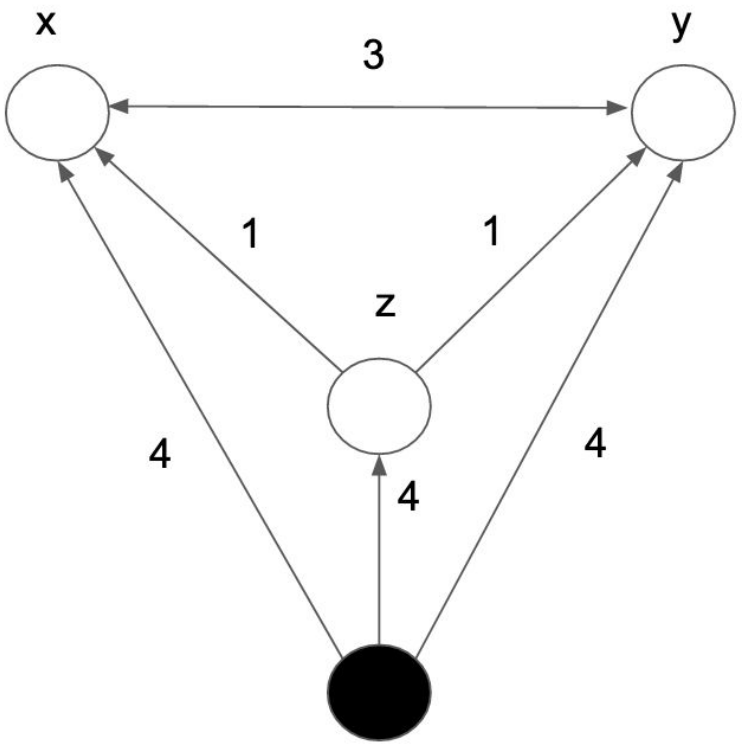

```
4 :: x.
4 :: y.
4 :: z.
1 :: z -> x.
1 :: z -> y.
3 :: x -> y.
3 :: y -> x.
```

```
Cd(x) = 4
Cd(x * y) = 7
Cd(<x, y>) = 6
```

```
4 :: +x.
4 :: +y.
4 :: +z.
1 :: +z -> +x.
1 :: +z -> +y.
3 :: +x -> +y.
3 :: +y -> +x.
```

```
Cw(+x) = 4
Cw(+x * +y) = 7
Cw(<+x, +y>) = 7
```

Example program




```
4 :: x.
4 :: y.
4 :: z.
1 :: z -> x.
1 :: z -> y.
3 :: x -> y.
3 :: y -> x.
```

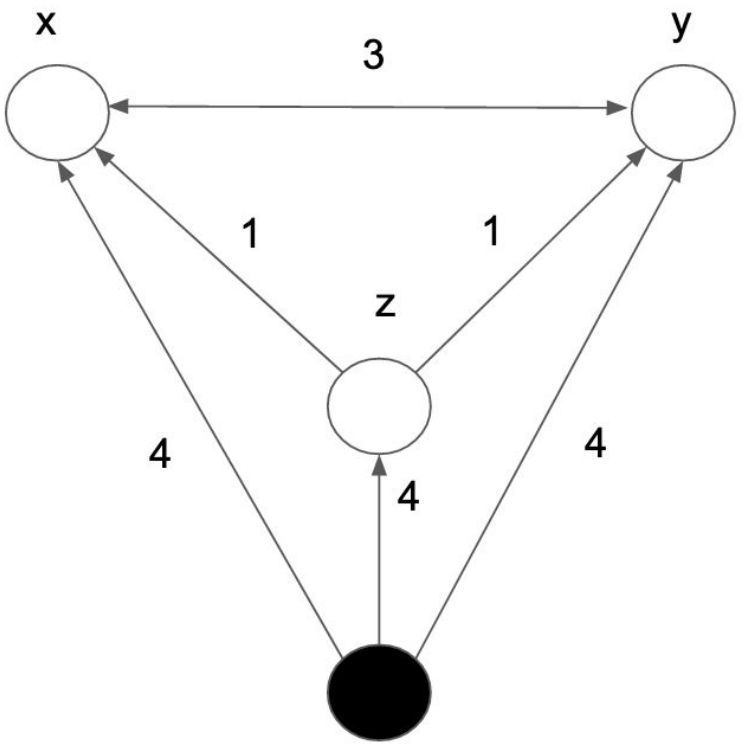
```
Cd(x) = 4
Cd(x * y) = 7
Cd(<x, y>) = 6
```

```
U(x) = 0
U(x * y) = 0
U(<x, y>) = 1
```

```
4 :: +x.
4 :: +y.
4 :: +z.
1 :: +z -> +x.
1 :: +z -> +y.
3 :: +x -> +y.
3 :: +y -> +x.
```

```
Cw(+x) = 4
Cw(+x * +y) = 7
Cw(<+x, +y>) = 7
```

Example program




```
4 :: x.
4 :: y.
4 :: z.
1 :: z -> x.
1 :: z -> y.
3 :: x -> y.
3 :: y -> x.
```

```
Cd(x) = 4
Cd(x * y) = 7
Cd(<x, y>) = 6
```

```
U(x) = 0
U(x * y) = 0
U(<x, y>) = 1
```

```
4 :: +x.
4 :: +y.
4 :: +z.
1 :: +z -> +x.
1 :: +z -> +y.
3 :: +x -> +y.
3 :: +y -> +x.
```

```
Cw(+x) = 4
Cw(+x * +y) = 7
Cw(<+x, +y>) = 7
```

base for **learning**?
perhaps we should introduce a new situation `z` that aggregates `x` and `y`?

CompLog: implementation

Derivation via ASP

COMPLOG PROGRAM

```
cost(s, x, 4).
cost(s, y, 4).
cost(s, z, 4).
cost(z, x, 1).
cost(z, y, 1).
cost(x, y, 3).
cost(y, x, 3).
start(s).

goal(x). goal(y).
```

EXPLORATION

```
% every outgoing edge from a reached node is a path
path(X, Y) :- reached(X, N), edge(X, Y).

% a starting node qualifies as reached
reached(X, 0) :- start(X).

% all goals should be reached
:- goal(Y), not reached(Y, _).
```

OPTIMIZATION

```
totalcost(T) :- T = #sum{C, X, Y : path(X, Y), cost(X,Y,C)}.
#minimize {T: totalcost(T)}.
```

CONSTRAINTS

```
% IF non-sequential search
{ reached(Y, N) } :- path(X, Y), reached(X, N).

% ELSE IF sequential search
{ reached(Y, N + 1) } :- path(X, Y), reached(X, N), N < 10.

      % IF race conditions (interleaved semantics)
      :- reached(X, N), reached(Y, N), X != Y.
```

CompLog: examples of application

Finding the most relevant description!

```
eagle -> bird.
pigeon -> bird.
canary -> bird.
tiger -> mammal.
dog -> mammal.
cat -> mammal.
dog -> pet.
cat -> pet.
canary -> pet.
```

eg. from frequency of presence in the communications

```
4 :: eagle.
4 :: pigeon.
6 :: canary.
4 :: tiger.
3 :: dog.
3 :: cat.
```

eg. from frequency of actual encounters

```
12 :: #eagle.
3 :: #pigeon.
7 :: #canary.
12 :: #tiger.
3 :: #dog.
3 :: #cat.
```

Finding the most relevant description!

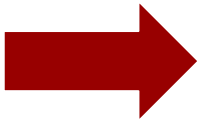
```
eagle -> bird.
pigeon -> bird.
canary -> bird.
tiger -> mammal.
dog -> mammal.
cat -> mammal.
dog -> pet.
cat -> pet.
canary -> pet.
```

eg. from frequency of presence in the communications

```
4 :: eagle.
4 :: pigeon.
6 :: canary.
4 :: tiger.
3 :: dog.
3 :: cat.
```

eg. from frequency of actual encounters

```
12 :: #eagle.
3 :: #pigeon.
7 :: #canary.
12 :: #tiger.
3 :: #dog.
3 :: #cat.
```



by computing the various unexpectedness we can settle on what is the best descriptor (min **U**) of the current situation, eg. given `#pigeon`, we may say `bird`

Disjunction?

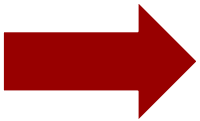
```
% declarative model with disjunction (as in ProbLog)
2 :: die1; 2 :: die2; 2 :: die3; 2 :: die4.

% correspondent procedural model with race conditions
2 :: => die1. 2 :: => die2. 2 :: => die3. 2 :: => die4.
```

Disjunction?

```
% declarative model with disjunction (as in ProbLog)
2 :: die1; 2 :: die2; 2 :: die3; 2 :: die4.

% correspondent procedural model with race conditions
2 :: => die1. 2 :: => die2. 2 :: => die3. 2 :: => die4.
```

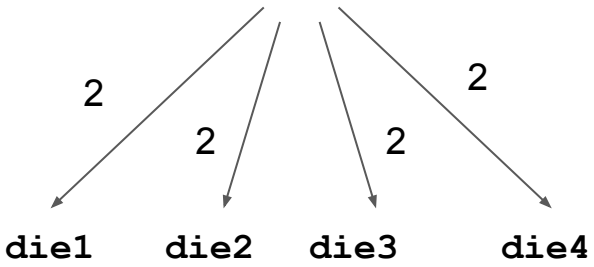


(exclusive) disjunction in probability specification
conveys implicitly the presence of race conditions!

Negation?

```
% declarative model with disjunction (as in ProbLog)
2 :: die1; 2 :: die2; 2 :: die3; 2 :: die4.

query ~die1.
```



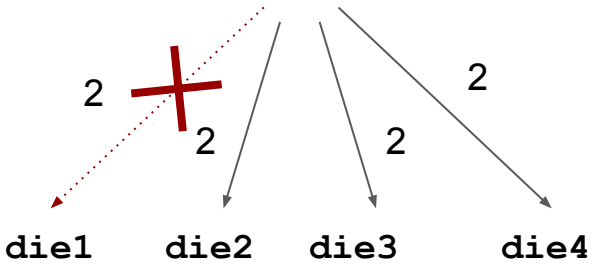
- We first compute the complexity of `die1`.

`C(die1) = 2`

Negation?

```
% declarative model with disjunction (as in ProbLog)
2 :: die1; 2 :: die2; 2 :: die3; 2 :: die4.

query ~die1.
```

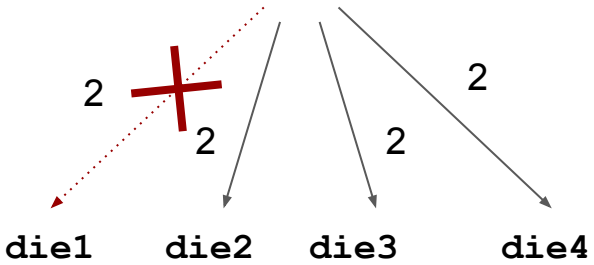


- We first compute the complexity of **die1**.
- We then remove **die1** from the graph.

Negation?

```
% declarative model with disjunction (as in ProbLog)
2 :: die1; 2 :: die2; 2 :: die3; 2 :: die4.

query ~die1.
```



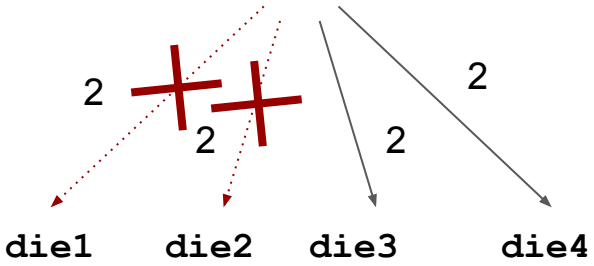
- We first compute the complexity of **die1**.
- We then remove **die1** from the graph.
- We compute the node with the best complexity, eg. **die2**.

C(~die1) = C(die2) = 2

Negation?

```
% declarative model with disjunction (as in ProbLog)
2 :: die1; 2 :: die2; 2 :: die3; 2 :: die4.

query ~die1.
```

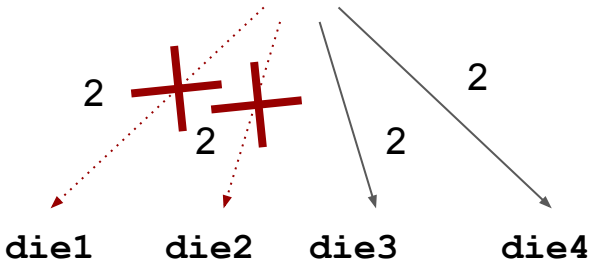


- We first compute the complexity of **`die1`**.
- We then remove **`die1`** from the graph.
- We compute the node with the best complexity, eg. **`die2`**.
- If needed, we can proceed incrementally, negating **`die2`** and so on, *aggregating* the complexities.

Negation?

```
% declarative model with disjunction (as in ProbLog)
2 :: die1; 2 :: die2; 2 :: die3; 2 :: die4.

query ~die1.
```



- We first compute the complexity of **`die1`**.
- We then remove **`die1`** from the graph.
- We compute the node with the best complexity, eg. **`die2`**.
- If needed, we can proceed incrementally, negating **`die2`** and so on, *aggregating* the complexities.

possibility of sequential, approximated computation!

Conclusions

- We presented a novel framework for automated inference in context of uncertainty based on **Simplicity Theory**, relying on the computation of two distinct Kolmogorov complexities by means of min-path search.
- Three additional practical reasons motivates continuing the exploration:
  - **Integration potential**: Enabling by design the distinction of descriptive and causal dimensions, it supports support the development of dedicated tools;
  - **Efficiency**: because of the greedy search related to the minimization of complexity, we hypothesize that it is faster than a probabilistic equivalent system;
  - **Cognitive modeling soundness**: the framework can mimic cognitive mechanisms observable in humans.

From ProbLog to CompLog

From Probabilistic to Complexity-based Logic Programming

1st October 2023, HYDRA workshop @ ECAI 2023

2nd International Workshop on Hybrid Models for coupling Deductive and Inductive Reasoning

Giovanni Sileno
g.sileno@uva.nl

University of Amsterdam

Jean-Louis Dessalles
jean-louis.dessalles@telecom-paris.fr

Télécom Paris