Scallop and Neuro-Symbolic Programming

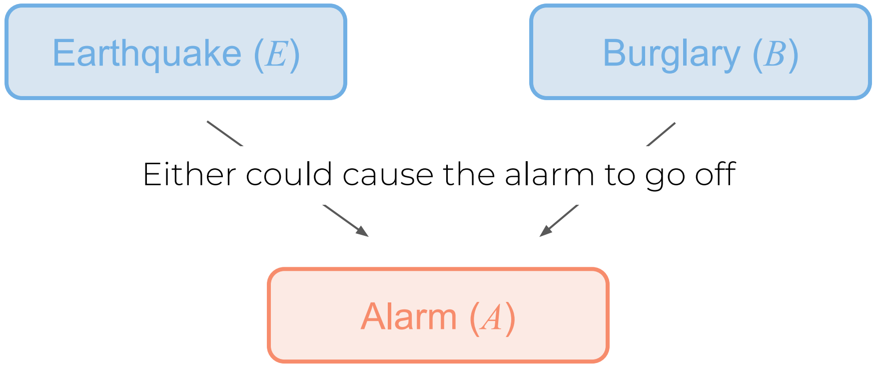
Lecture 2: Tags, Instrumentation, and Provenance

Agenda

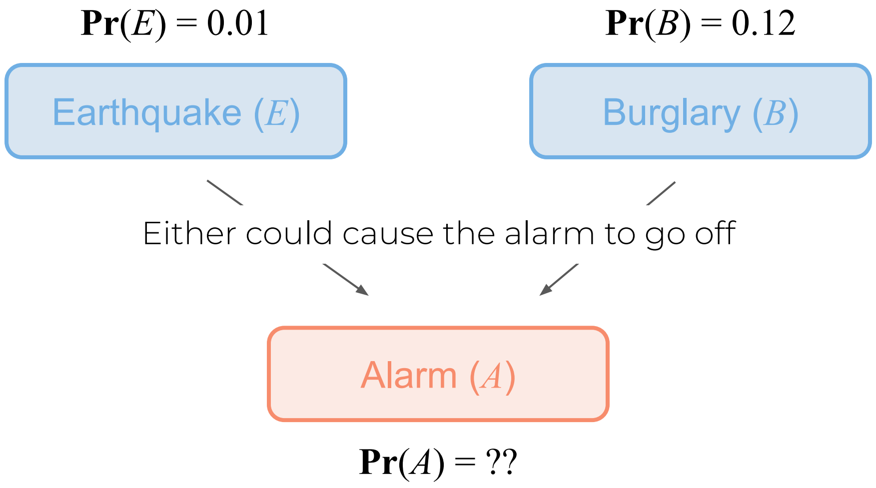
1. **Using Probabilities in Scallop**
   Discrete Reasoning Augmented
2. **Tagging and Instrumentation**
   How to Associate Extra Information when Reasoning
3. **The Ultimate Tag: Provenance**
   Tracking, Recover, and WMC
4. **Provenance Framework**
   Minmaxprob, AddMultProb, TopKProofs, TopBottomKClauses

Scallop: Working with Probabilities

Scallop with Probabilities

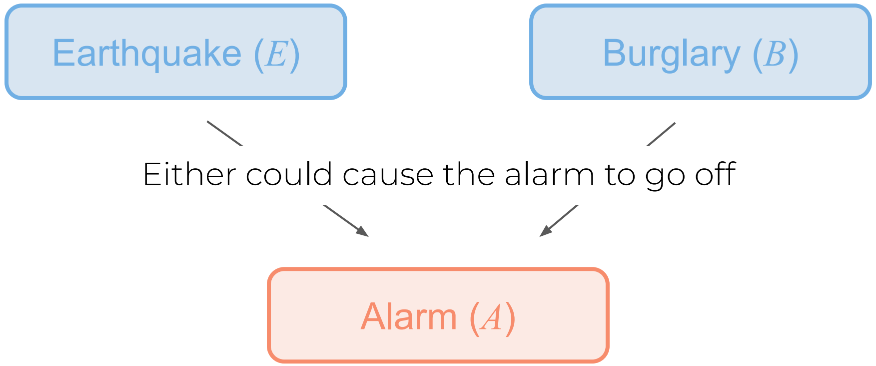


Scallop with Probabilities



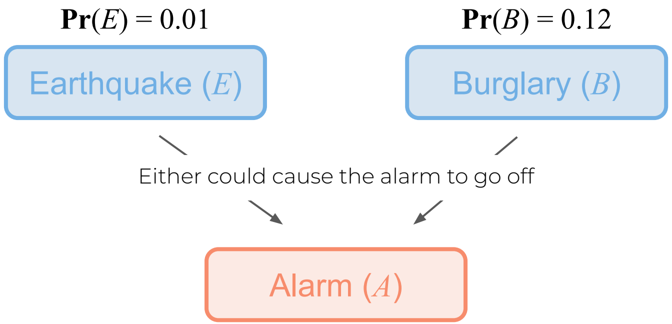
Scallop with Probabilities

$\mathbf{Pr}(E) = 0.01$ $\mathbf{Pr}(B) = 0.12$



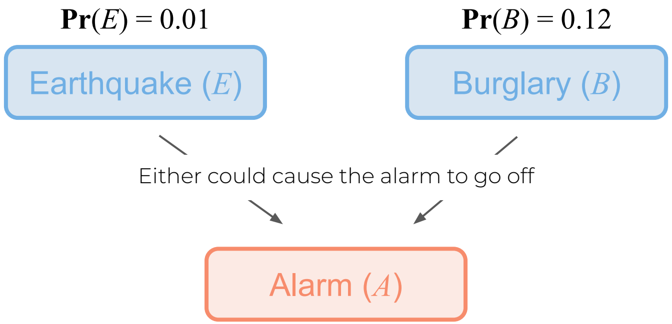
$$\mathbf{Pr}(A) = 1 - \mathbf{Pr}(\neg E \land \neg B) = 1 - \mathbf{Pr}(\neg E)\ \mathbf{Pr}(\neg \mathrm{B}) = 1 - (1 - \mathbf{Pr}(\mathrm{E}))\ (1 - \mathbf{Pr}(\mathrm{B})) = 0.1288$$

Scallop with Probabilities




```
rel 0.01::earthquake()
rel 0.12::burglary()
rel alarm() = earthquake() or burglary()
```

Scallop with Probabilities

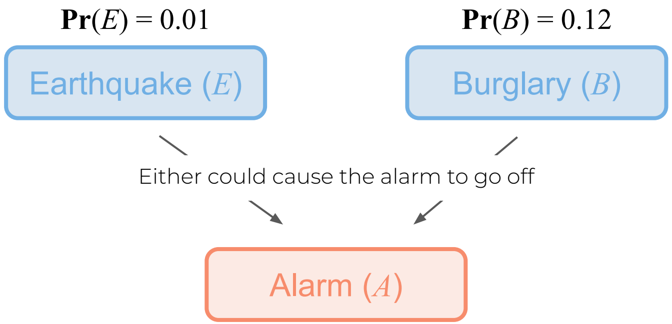



```
rel 0.01::earthquake()
rel 0.12::burglary()
rel alarm() = earthquake() or burglary()
```

```
> scli alarm.scl --provenance topkproofs
alarm: {0.1288::()}
```

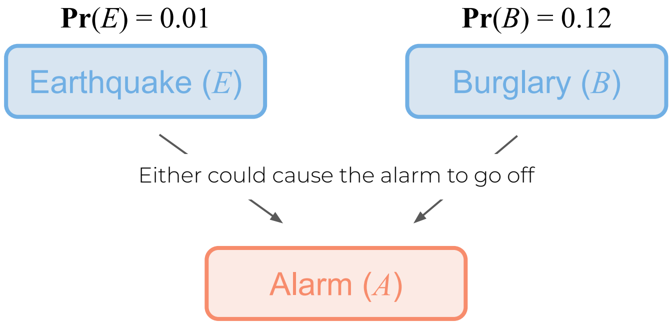
Tagging and Instrumentation

Proof Tree Revised

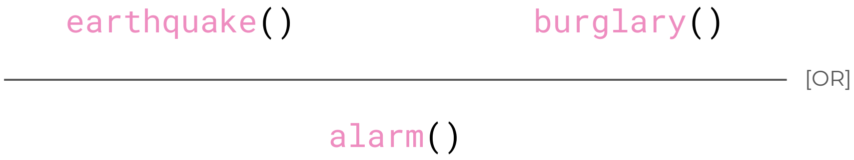



```
rel 0.01::earthquake()
rel 0.12::burglary()
rel alarm() = earthquake() or burglary()
```

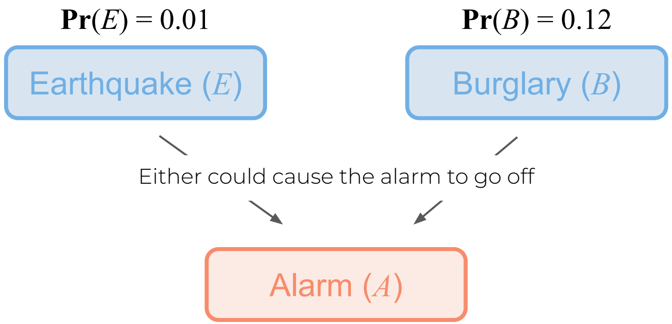
Proof Tree Revised




```
rel 0.01::earthquake()
rel 0.12::burglary()
rel alarm() = earthquake() or burglary()
```



Add Tags to the Facts




```
rel 0.01::earthquake()
rel 0.12::burglary()
rel alarm() = earthquake() or burglary()
```

$$\frac{x :: \texttt{earthquake()} \qquad y :: \texttt{burglary()}}{z :: \texttt{alarm()}}\ [\text{OR}]$$

Add Tags to the Facts

$$\frac{x :: \texttt{earthquake()} \qquad y :: \texttt{burglary()}}{z :: \texttt{alarm()}} \;[\text{OR}]$$

A simple and straightforward approach...

0.01

0.12

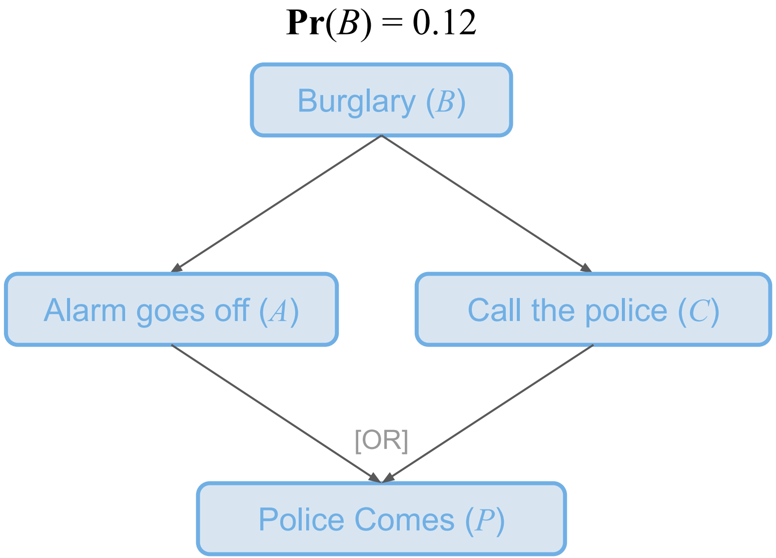
x :: `earthquake()` $\quad$ y :: `burglary()`

[OR]

z :: `alarm()`

$z = 1 - (1 - x)(1 - y) = 0.1288$

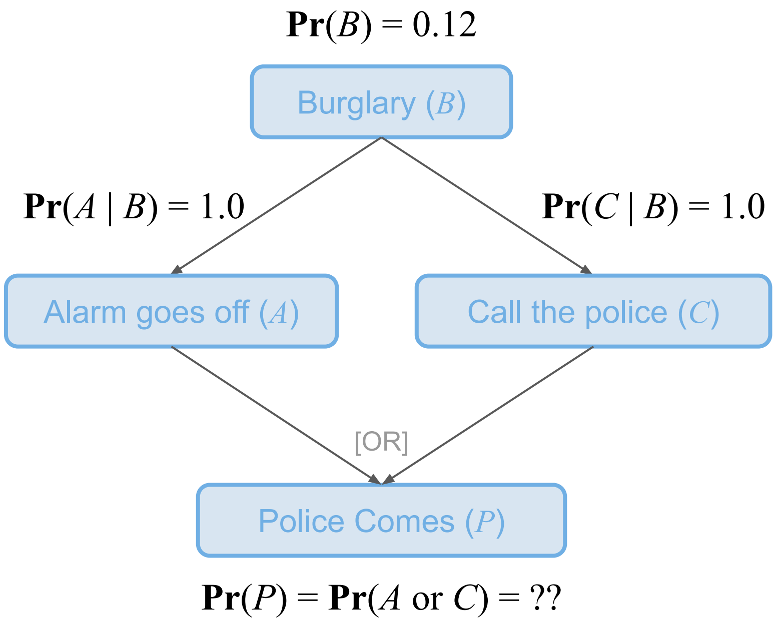
What could be a problem?



What could be a problem?

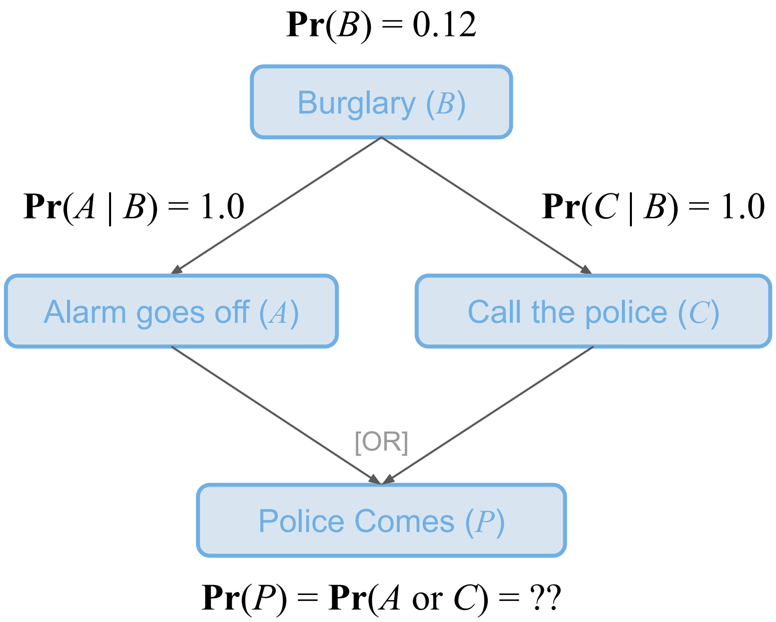

```
rel 0.12::burglary()
rel alarm_goes_off() = burglary()
rel call_the_police() = burglary()
rel police_comes() =
    alarm_goes_off() or call_the_police()
```

What could be a problem?



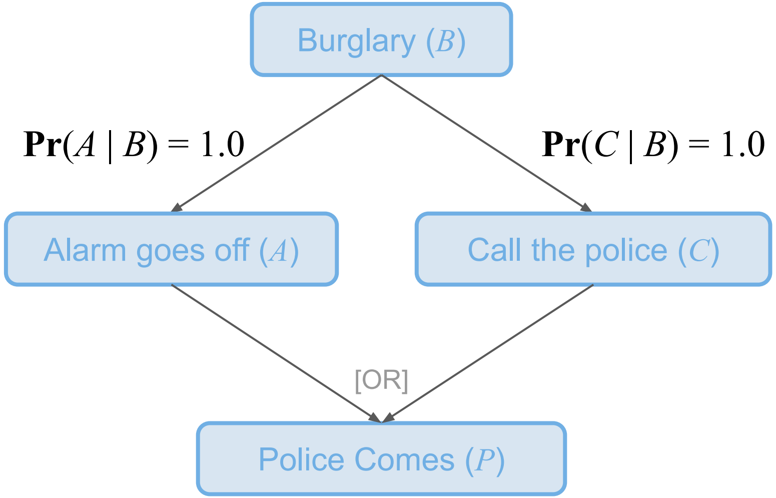

```
rel 0.12::burglary()
rel alarm_goes_off() = burglary()
rel call_the_police() = burglary()
rel police_comes() =
    alarm_goes_off() or call_the_police()
```

What could be a problem?

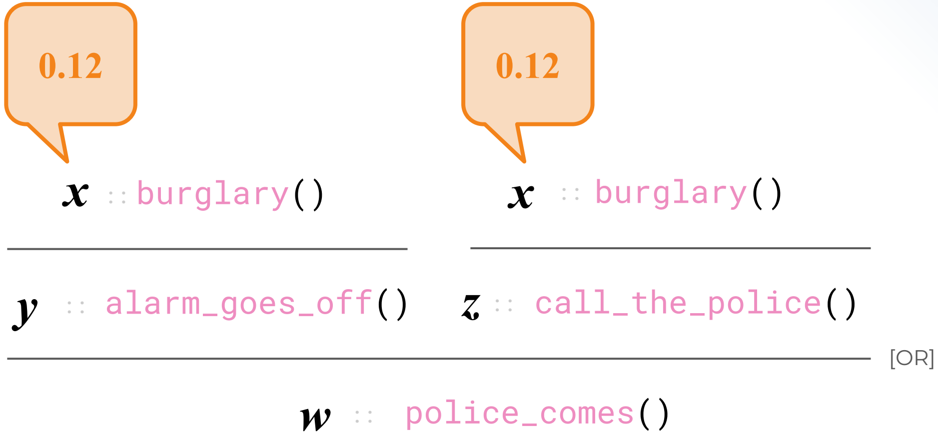


$$\dfrac{\dfrac{x :: \texttt{burglary()}}{y :: \texttt{alarm\_goes\_off()}} \qquad \dfrac{x :: \texttt{burglary()}}{z :: \texttt{call\_the\_police()}}}{w :: \texttt{police\_comes()}}\ \text{[OR]}$$

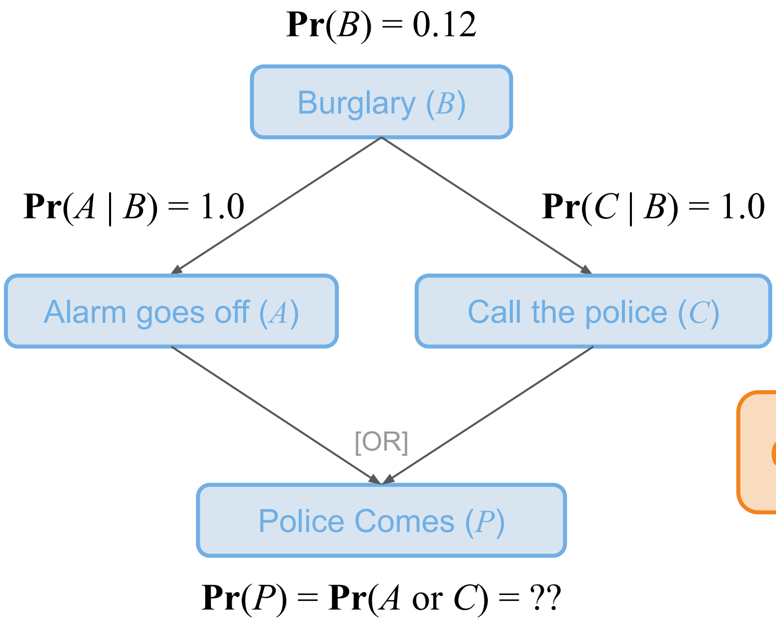
What could be a problem?



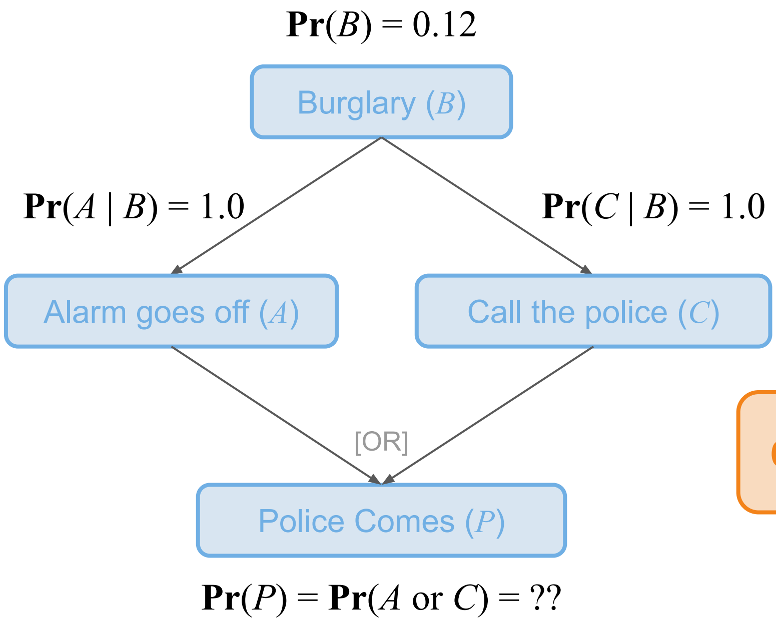
$\mathbf{Pr}(P) = \mathbf{Pr}(A \text{ or } C) = ??$

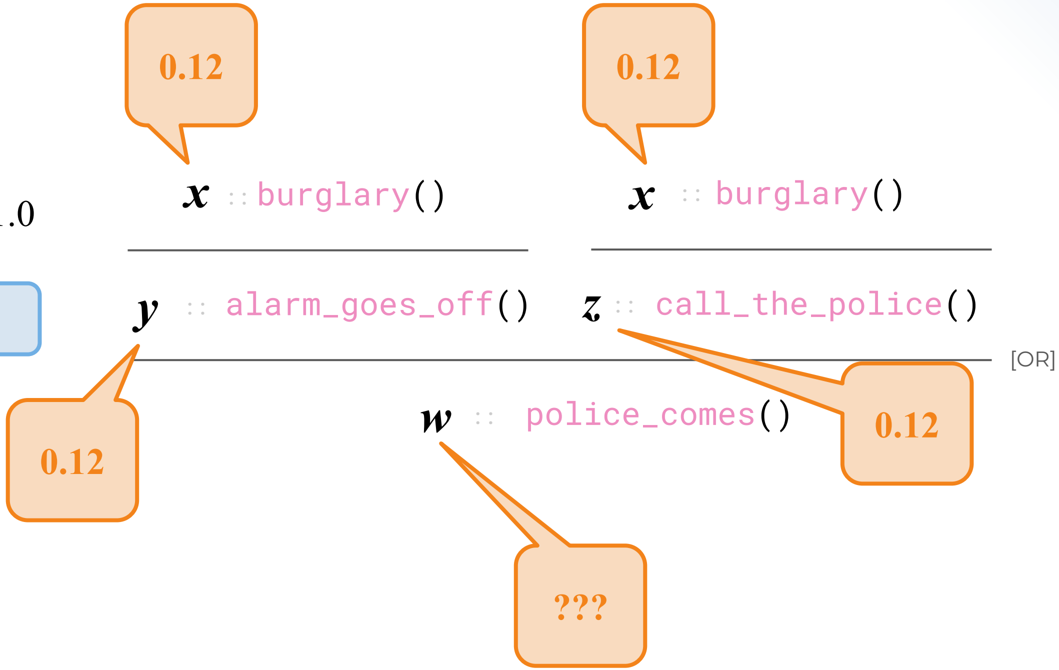


What could be a problem?

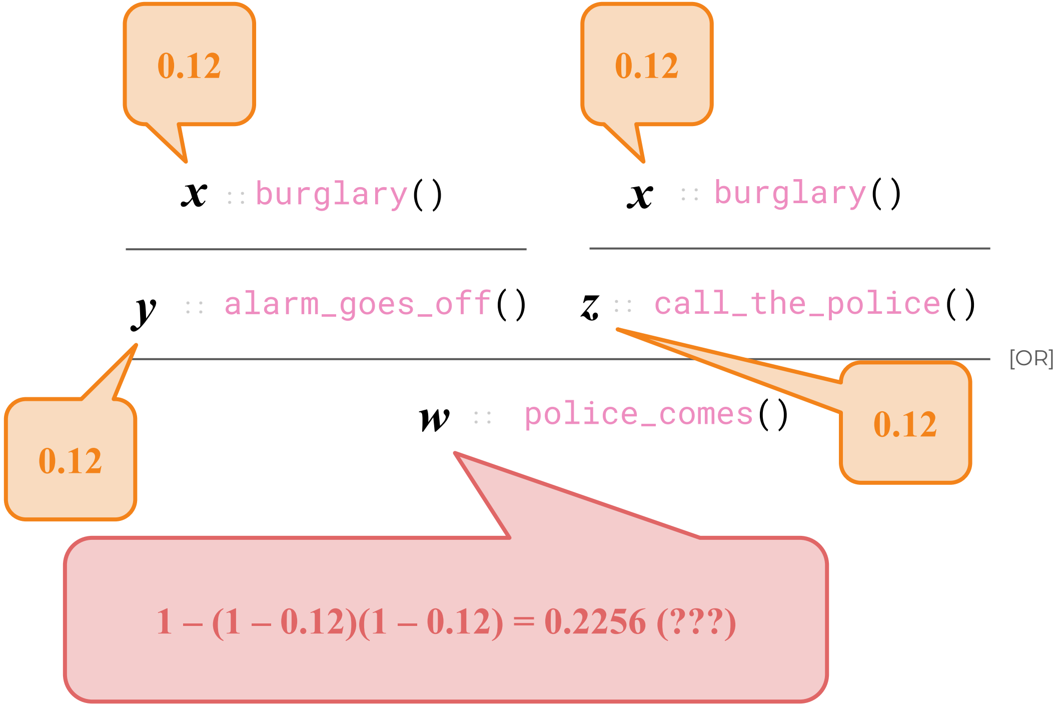


What could be a problem?





What could be the problem?



What could be the problem?

0.12

0.12

$$\cfrac{\cfrac{x :: \texttt{burglary()}}{y :: \texttt{alarm\_goes\_off()}} \qquad \cfrac{x :: \texttt{burglary()}}{z :: \texttt{call\_the\_police()}}}{w :: \texttt{police\_comes()}}\ [\text{OR}]$$

y and *z* are not independent!

The Ultimate Tag: Provenance

Track how a fact is derived!

Context: $B :: \texttt{burglary()}$, $\mathbf{Pr}(B) = 0.12$

Derivation:

$$\dfrac{\dfrac{B :: \texttt{burglary()}}{B :: \texttt{alarm\_goes\_off()}} \qquad \dfrac{B :: \texttt{burglary()}}{B :: \texttt{call\_the\_police()}}}{P = B \text{ or } B = B :: \texttt{police\_comes()}}\ \text{[OR]}$$

Recover: $\mathbf{Pr}(P) = \mathbf{Pr}(B) = 0.12$

Track how a fact is derived!

Context:

E :: `earthquake()` , $\mathbf{Pr}(E) = 0.01$

B :: `burglary()` , $\mathbf{Pr}(B) = 0.12$

Derivation:

$$\frac{E \text{ :: } \texttt{earthquake()} \qquad\qquad B \text{ :: } \texttt{burglary()}}{A = E \text{ or } B \text{ :: } \texttt{alarm()}} \text{ [OR]}$$

Recover:

$\mathbf{Pr}(A) = \mathbf{Pr}(E \text{ or } B) = 0.1288$

Boolean Formula as Tag!

- Before running the program, we create boolean variables for each input fact with probability, and store the variables and their probabilities inside a context
- The tag for each input fact would be the boolean variable of itself
- During execution, we use “AND”, “OR”, and “NOT” to combine these variables into complex boolean formula
- At the end of execution, we recover the probability of the derived fact by performing “Weighted Model Counting (WMC)” on the boolean formula associated with that fact

Boolean Formula Tag, Formally

$$
\begin{array}{lrl}
\text{(Boolean Variables)} & \boldsymbol{V} & \\
\text{(Boolean Formula)} & \boldsymbol{F} ::= & \boldsymbol{V} \\
 & & \mid \text{or}(\boldsymbol{F}, \boldsymbol{F}) \\
 & & \mid \text{and}(\boldsymbol{F}, \boldsymbol{F}) \\
 & & \mid \text{not}(\boldsymbol{F}) \\
\text{(Context)} & \mathbf{C} :: & \boldsymbol{V} \rightarrow [0, 1] \\
\text{(WMC)} & \text{wmc} :: & (\boldsymbol{F}, \mathbf{C}) \rightarrow [0, 1]
\end{array}
$$

Boolean Formula Tag, Formally

Each input fact with probability will be assigned a boolean variable in $\boldsymbol{V}$

$$
\begin{array}{lrl}
\text{(Boolean Variables)} & \boldsymbol{V} & \\
\text{(Boolean Formula)} & \boldsymbol{F} ::= & \boldsymbol{V} \\
 & & \mid \text{or}(\boldsymbol{F}, \boldsymbol{F}) \\
 & & \mid \text{and}(\boldsymbol{F}, \boldsymbol{F}) \\
 & & \mid \text{not}(\boldsymbol{F}) \\
\text{(Context)} & \mathbf{C} :: & \boldsymbol{V} \rightarrow [0, 1] \\
\text{(WMC)} & \text{wmc} :: & (\boldsymbol{F}, \mathbf{C}) \rightarrow [0, 1]
\end{array}
$$

Boolean Formula Tag, Formally

Every input fact and derived fact will be tagged with a boolean formula

$$
\begin{array}{lrl}
\text{(Boolean Variables)} & \boldsymbol{V} & \\
\text{(Boolean Formula)} & \boldsymbol{F} ::= & \boldsymbol{V} \\
 & & \mid \text{or}(\boldsymbol{F}, \boldsymbol{F}) \\
 & & \mid \text{and}(\boldsymbol{F}, \boldsymbol{F}) \\
 & & \mid \text{not}(\boldsymbol{F}) \\
\text{(Context)} & \mathbf{C} :: & \boldsymbol{V} \rightarrow [0, 1] \\
\text{(WMC)} & \text{wmc} :: & (\boldsymbol{F}, \mathbf{C}) \rightarrow [0, 1]
\end{array}
$$

Boolean Formula Tag, Formally

(Boolean Variables) $\boldsymbol{V}$

(Boolean Formula) $\boldsymbol{F} ::= \boldsymbol{V}$

$\quad\mid \text{or}(\boldsymbol{F}, \boldsymbol{F})$

$\quad\mid \text{and}(\boldsymbol{F}, \boldsymbol{F})$

$\quad\mid \text{not}(\boldsymbol{F})$

(Context) $\mathbf{C} :: \boldsymbol{V} \rightarrow [0, 1]$

(WMC) $\text{wmc} :: (\boldsymbol{F}, \mathbf{C}) \rightarrow [0, 1]$

Boolean formula can be propagated with “or”, “and”, and “not” operations

Boolean Formula Tag, Formally

$$
\begin{array}{lr}
\text{(Boolean Variables)} & \boldsymbol{V} \\
\text{(Boolean Formula)} & \boldsymbol{F} ::= \boldsymbol{V} \\
 & \mid \text{or}(\boldsymbol{F}, \boldsymbol{F}) \\
 & \mid \text{and}(\boldsymbol{F}, \boldsymbol{F}) \\
 & \mid \text{not}(\boldsymbol{F}) \\
\text{(Context)} & \mathbf{C} :: \boldsymbol{V} \rightarrow [0, 1] \\
\text{(WMC)} & \text{wmc} :: (\boldsymbol{F}, \mathbf{C}) \rightarrow [0, 1]
\end{array}
$$

A context is a mapping from each boolean variable to its probability

Boolean Formula Tag, Formally

$$
\begin{array}{lrl}
\text{(Boolean Variables)} & \boldsymbol{V} & \\
\text{(Boolean Formula)} & \boldsymbol{F} ::= & \boldsymbol{V} \\
 & & \mid \text{or}(\boldsymbol{F}, \boldsymbol{F}) \\
 & & \mid \text{and}(\boldsymbol{F}, \boldsymbol{F}) \\
 & & \mid \text{not}(\boldsymbol{F}) \\
\text{(Context)} & \mathbf{C} :: & \boldsymbol{V} \to [0, 1] \\
\text{(WMC)} & \text{wmc} :: & (\boldsymbol{F}, \mathbf{C}) \to [0, 1]
\end{array}
$$

Weighted Model Counting is a method that can compute the probability of a boolean formula being true, based on the context

Example: The sum of two probabilistic digits

```
rel digit_1 = {0.01::0; 0.01::1; 0.98::2}
rel digit_2 = {0.02::0; 0.97::1; 0.01::2}
rel sum_of_digits(x + y) = digit_1(x) and digit_2(y)
```

Example: The sum of two probabilistic digits

```
rel digit_1 = {0.01::0; 0.01::1; 0.98::2}
rel digit_2 = {0.02::0; 0.97::1; 0.01::2}
rel sum_of_digits(x + y) = digit_1(x) and digit_2(y)
```

```
sum_of_digits(0)

sum_of_digits(1)

sum_of_digits(2)
```

Example: The sum of two probabilistic digits

```
rel digit_1 = {0.01::0; 0.01::1; 0.98::2}
rel digit_2 = {0.02::0; 0.97::1; 0.01::2}
rel sum_of_digits(x + y) = digit_1(x) and digit_2(y)
```

```
(digit_1(0) and digit_2(0))    :: sum_of_digits(0)

(digit_1(0) and digit_2(1)) or
(digit_1(1) and digit_2(0))    :: sum_of_digits(1)

(digit_1(0) and digit_2(2)) or
(digit_1(2) and digit_2(0)) or
(digit_1(1) and digit_2(1))    :: sum_of_digits(2)
```

Example: The sum of two probabilistic digits

```
rel digit_1 = {0.01::0; 0.01::1; 0.98::2}
rel digit_2 = {0.02::0; 0.97::1; 0.01::2}
rel sum_of_digits(x + y) = digit_1(x) and digit_2(y)
```

```
sum_of_digits: {
  0.0002::(0),
  0.00989806::(1),
  0.029206969011999998::(2),
  0.950604939999998::(3),
  0.0098::(4)
}
```

Provenance Framework

Full Boolean Formula is Exact and Perfect, but...

- The Weighted Model Counting is a very time consuming task!
  - Model: Solution to a propositional boolean formula
  - Model Counting: Number of solutions to a propositional boolean formula
    - #SAT: #P-Complete Problem
  - Weighted Model Counting: “number of solutions” with weights assigned to each boolean variable
- Scallop, paired with a deep neural network, needs to reason about hundreds of facts for each data-point, while there could be thousands of data-points per dataset
  - Performance is a huge issue!

The need for a generalized framework

- We cannot afford to perform the exact probabilistic reasoning with full provenance
- However, there are many approximation algorithms that can also facilitate differentiable and probabilistic reasoning, and can yield good learning outcome
- Therefore, we want to design a generalized framework for instrumenting arbitrary tag for differentiable and probabilistic reasoning
- And we get customizability and scalability!

The need for a generalized framework

- We cannot afford to perform the exact probabilistic reasoning with full provenance
- However, there are many approximation algorithms that can also facilitate differentiable and probabilistic reasoning, and can yield good learning outcome
- Therefore, we want to design a generalized framework for instrumenting arbitrary tag for differentiable and probabilistic reasoning
- And we get customizability and scalability!

(Extended) **Provenance Semiring Framework**

Add Tags to the Facts

We let the tags associated with the facts $x, y, z \in T$, where T is a set of all possible tags, which we call *Tag Space*

$$\frac{x :: \texttt{earthquake()} \qquad y :: \texttt{burglary()}}{z :: \texttt{alarm()}} \; [\text{OR}]$$

Propagation of Tags through OR

For this Tag Space $\boldsymbol{T}$, we define a binary operation $\oplus$ to combine two tags when they are "OR"-ed together:

$$\frac{x :: \texttt{earthquake()} \qquad\qquad y :: \texttt{burglary()}}{x \oplus y :: \texttt{alarm()}}\ \text{[OR]}$$

Propagation of Tags through AND

For this Tag Space $\boldsymbol{T}$, we define a binary operation $\otimes$ to combine two tags when they are "AND"-ed together:

$$\frac{\boldsymbol{x} :: \texttt{father("John", "Alice")} \qquad \boldsymbol{y} :: \texttt{mother("Alice", "Bob")}}{\boldsymbol{x} \otimes \boldsymbol{y} :: \texttt{grandfather("John", "Bob")}}\ \text{[AND]}$$

Propagation of Tag through Negation

For this Tag Space $\boldsymbol{T}$, we define a unary operation $\ominus$ to negate a tag

$$\frac{\boldsymbol{x} :: \texttt{color(0, "red")}}{\ominus \boldsymbol{x} :: \texttt{\textasciitilde color(0, "red")}}\ \text{[NOT]}$$

True and False Tag

For this Tag Space $\boldsymbol{T}$, we need additionally two elements that serves as "True" ($\mathbf{1}$) and "False" ($\mathbf{0}$):

They should have the following properties:

- $\oplus(1, X) = 1$
- $\oplus(0, X) = X$
- $\otimes(1, X) = X$
- $\otimes(0, X) = 0$

Provenance Framework for Probabilistic Reasoning

(Tag Space)	$\boldsymbol{T}$
(Additive Identity/False)	$\mathbf{0} :: \boldsymbol{T}$
(Multiplicative Identity/True)	$\mathbf{1} :: \boldsymbol{T}$
(Addition/Or)	$\oplus(\boldsymbol{T}, \boldsymbol{T}) \rightarrow \boldsymbol{T}$
(Multiplication/And)	$\otimes(\boldsymbol{T}, \boldsymbol{T}) \rightarrow \boldsymbol{T}$
(Negation/Not)	$\ominus(\boldsymbol{T}) \rightarrow \boldsymbol{T}$
(Recover Function)	$\rho(\boldsymbol{T}) \rightarrow [0, 1]$

Example: Add/Mult Probability

Definition:

$\boldsymbol{T} ::= [0, 1]$
$\mathbf{0} ::= 0$
$\mathbf{1} ::= 1$
$\oplus(x, y) = min(1, x + y)$
$\otimes(x, y) = x \times y$
$\ominus(x) = 1 - x$

Example:

0.01 :: `earthquake()` $\quad$ 0.12 :: `burglary()`

-- [OR]

$\oplus(0.01, 0.12) = min(1, 0.01 + 0.12) = 0.13$ :: `alarm()`

Example: Min/Max Probability

Definition:

$\boldsymbol{T}$ ::= [0, 1]
$\mathbf{0}$ ::= 0
$\mathbf{1}$ ::= 1
$\boldsymbol{\oplus}$ ::= $\oplus(x, y) = max(x, y)$
$\boldsymbol{\otimes}$ ::= $\otimes(x, y) = min(x, y)$
$\boldsymbol{\ominus}$ ::= $\ominus(x) = 1 - x$

Example:

$$\frac{0.01 :: \texttt{earthquake()} \qquad 0.12 :: \texttt{burglary()}}{\oplus(0.01, 0.12) = max(0.01, 0.12) = 0.12 :: \texttt{alarm()}} \text{[OR]}$$

Example: Proofs (DNF Formula)

Definition:

$\boldsymbol{T} ::= \mathrm{P}(\mathrm{P}(\boldsymbol{V}))$
$\mathbf{0} ::= \{\}$
$\mathbf{1} ::= \{\{\}\}$
$\oplus(X,\ Y) = X \cup\ Y$
$\otimes(X,\ Y) = \{P \cup\ Q \text{ for } P \in X, Q \in\ Y\}$
$\ominus$(Undefined)

Example:

```
rel digit_1 = {0.01::0; 0.01::1; 0.98::2}
rel digit_2 = {0.02::0; 0.97::1; 0.01::2}
rel sum_of_digits(x + y) = digit_1(x) and digit_2(y)
```

```
sum_of_digits(1):
  -   Formula: (d1(0) and d2(1)) or (d1(1) and d2(0))
  -   Tag: {{d1(0), d2(1)}, {d1(1), d2(0)}}
```

Example: Proofs (DNF Formula)

Definition:

$\boldsymbol{T} ::= \mathrm{P}(\mathrm{P}(\boldsymbol{V}))$
$\mathbf{0} ::= \{\}$
$\mathbf{1} ::= \{\{\}\}$
$\oplus(X, Y) = X \cup Y$
$\otimes(X, Y) = \{P \cup Q \text{ for } P \in X, Q \in Y\}$
$\ominus$(Undefined)

This provenance is only defined on positive Scallop program

Example:

```
rel digit_1 = {0.01::0; 0.01::1; 0.98::2}
rel digit_2 = {0.02::0; 0.97::1; 0.01::2}
rel sum_of_digits(x + y) = digit_1(x) and digit_2(y)
```

```
sum_of_digits(1):
  -   Formula: (d1(0) and d2(1)) or (d1(1) and d2(0))
  -   Tag: {{d1(0), d2(1)}, {d1(1), d2(0)}}
```

Example: Top-k Proofs (DNF Formula)

Definition:

$\boldsymbol{T} ::= \mathrm{P}(\mathrm{P}(\boldsymbol{V}))$
$\mathbf{0} ::= \{\}$
$\mathbf{1} ::= \{\{\}\}$
$\oplus(X,\ Y) = \text{top-k}(X \cup\ Y)$
$\otimes(X,\ Y) = \text{top-k}(\{P \cup\ Q \text{ for } P \in X, Q \in Y\})$
$\ominus(\text{Undefined})$

Example:

```
rel digit_1 = {0.01::0; 0.01::1; 0.98::2}
rel digit_2 = {0.02::0; 0.97::1; 0.01::2}
rel sum_of_digits(x + y) = digit_1(x) and digit_2(y)
```

```
sum_of_digits(3):
  -   Formula: (d1(1) and d2(2)) or (d1(2) and d2(1))
  -   Tag: top-1{{d1(1), d2(2)}, {d1(2), d2(1)}}
  -                ------P1------  ------P2------
  -     =>        {{d1(2), d2(1)}}
```