CS7480: Topics in Programming Languages: Probabilistic Programming

Lecture 4: `SimPPL` Intro

Instructor: Steven Holtzen

Place: Northeastern University

Term: Fall 2021

Course webpage:
https://www.khoury.northeastern.edu/home/sholtzen/CS7480Fall21/

Overview

- `SimPPL` is online and final
- This week we'll be going over it in detail
- Today:
  1. Language design
  2. Formal `SimPPL` syntax
  3. Formal `SimPPL` semantics
  4. Modeling with `SimPPL`
  5. Comparing languages

The Need for Modeling Languages

- Suppose you are a doctor and you know about the relationship between symptoms and diseases

- There is a 1% chance the average person has a cold, 1% chance of cough, 4% chance of a temperature, 3% chance of runny nose
- If you have a cold:
  - 10% of the time you have a temperature
  - 50% of the time you cough
  - 70% of the time you have a runny nose

- We want to be able to query: what is the probability that the patient has the flu given that they are coughing?

The Need for Modeling Languages

- The doctor *could* write down a joint probability table…

Cold?	Fever?	Temp?	Runny nose?	Cough?	Pr?
N	N	N	N	N	0.91266912
Y	Y	Y	Y	N	…

- … but this is not a convenient representation
  - This is not the way that the doctor understands this information

What makes a *good* modeling language?

- By no means solved, but some things to consider:
  - **Expressive power**: Is the language capable of concisely representing the relevant facts about the world?
    - **Conciseness**: How big do programs need to be?
    - **Completeness**: Can we write down any distribution we want?
  - **Accessibility and interpretability:** is the language accessible to non-expert modelers? Can the model be interpreted by non-experts?
  - **Tractability for inference**: Once a model is created, how efficiently can we perform inference?
- ***These are often in tension!***

Modeling with Programs

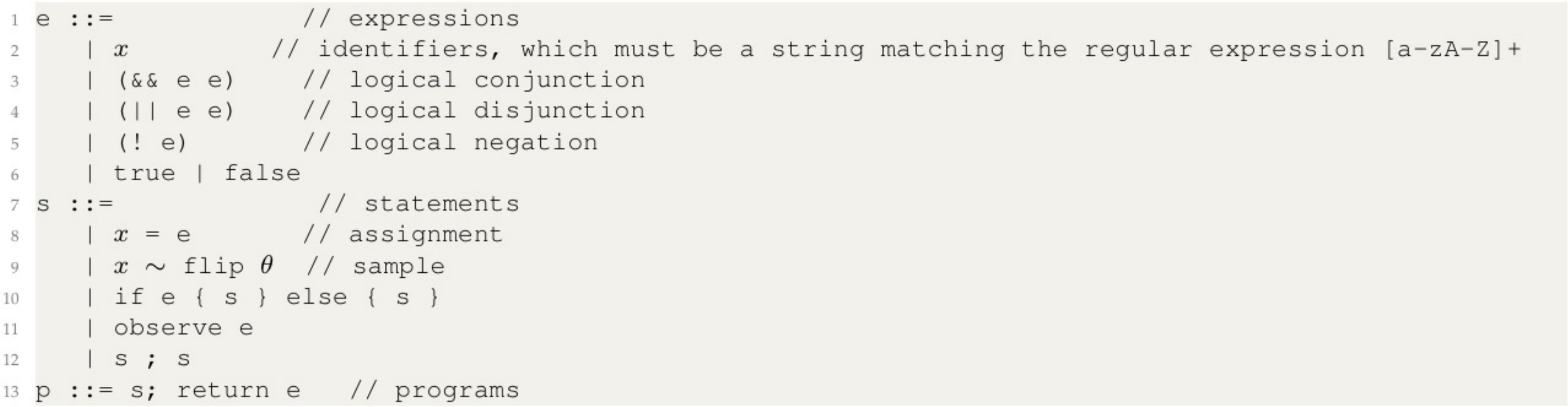
```
cold ~ flip 0.01;
if cold {
   cough ~ flip 0.5;
   temp ~ flip 0.1;
   runnyNose ~ flip 0.07
} else {
    cough ~ flip 0.01;
    temp ~ flip 0.04;
    runnyNose ~ flip 0.03
};
observe cough;
return cold
```

Answer:
0.335570469799

`SimPPL`

- A minimal probabilistic programming language
  - Only `flip` and Boolean values
  - Imperative: has statements and expressions
  - First-class observations: can condition at any point in the program
  - Returns a distribution on a single Boolean value

`SimPPL` Syntax



```
e ::=               // expressions
   | x          // identifiers, which must be a string matching the regular expression [a-zA-Z]+
   | (&& e e)     // logical conjunction
   | (|| e e)     // logical disjunction
   | (! e)        // logical negation
   | true | false
s ::=                // statements
   | x = e        // assignment
   | x ∼ flip θ   // sample
   | if e { s } else { s }
   | observe e
   | s ; s
p ::= s; return e   // programs
```

`SimPPL` Syntax

- This is a valid `SimPPL` program

```
cold ~ flip 0.01;
if cold {
   cough ~ flip 0.5;
   temp ~ flip 0.1;
   runnyNose ~ flip 0.07
} else {
    cough ~ flip 0.01;
    temp ~ flip 0.04;
    runnyNose ~ flip 0.03
};
observe cough;
return cold
```

`SimPPL` Parser

```
from lark import Lark
simppl_parser = Lark(r"""
   e: NAME
      | "(" "&&" e e ")"
      | "(" "||" e e ")"
      | "(" "!" e ")"
      | "true"
      | "false"

   s: NAME "=" e
     | NAME "~" "flip" SIGNED_NUMBER
     | "if" e "{" s "}" "else" "{" s "}"
     | "observe" e
     | s ";" s

  p: s ";" "return" e


     %import common.SIGNED_NUMBER
     %import common.WS
     %import common.CNAME -> NAME
     %ignore WS

""", start=’p’)
```

`SimPPL` Semantics

- Defined in two parts

1. Expressions: these are like normal non-probabilistic programs

2. Statements: this is where probabilities come in

`SimPPL` Expression Semantics

- Environment $\rho \in Env$ maps variables to Boolean values
  - Example: $\rho = \{X = T, Y = F\}$
  - Lookup: $\rho(X) = T$
- The semantics of expressions have the form

$$\llbracket \mathrm{e} \rrbracket : \texttt{Env} \to \{T, F\}$$

`SimPPL` Expression Semantics

$$[\![x]\!] \rho =$$

$$[\![(!\ e)]\!] \rho =$$

$$[\![(\&\&\ e_1\ e_2)]\!] \rho =$$

`SimPPL` Statement Semantics

- Map environments to *unnormalized distributions on environments*

$$[\![\mathtt{s}]\!] : \mathtt{Env} \to (\mathtt{Env} \to [0, 1])$$

- Interpret it as the unnormalized conditional probability of an output event given an input event

- This is where the probabilities show up

- Sample space: set of *all possible environments*

Assignment

- Syntax: `x = e`
- In ”normal” languages, updates the variable x to have the value denoted by e
- Here, we assign a distribution that gives probability 1 to that case

$$[\![\texttt{x=e}]\!](\rho)(\rho') = \begin{cases} 1 & \text{if } \rho' = \rho[x \mapsto [\![\texttt{e}]\!]\rho] \\ 0 & \text{otherwise} \end{cases}$$

Initial

Final

Assignment

- Syntax: `x = e`
- In ”normal” languages, updates the variable x to have the value denoted by e
- Here, we assign a distribution that gives probability 1 to that case

$$[\![\mathtt{x=e}]\!](\rho)(\rho') = \begin{cases} 1 & \text{if } \rho' \\ 0 & \end{cases}$$

Initial

Final

Could have used the probability notation

$$[\![\mathtt{x=e}]\!](\rho' \mid \rho)$$

Sample

- Syntax: `x ~ flip` θ
- Semantics: assign a probability of θ to the environment where x is true and $1 - \theta$ if it is not true

$$[\![x \sim \texttt{flip}\ \theta]\!](\rho)(\rho') = \begin{cases} \theta & \text{if } \rho' = \rho[x \mapsto T], \\ 1 - \theta & \text{if } \rho' = \rho[x \mapsto F] \\ 0 & \text{otherwise.} \end{cases}$$

Sequence

- Syntax: `s1; s2`
- Semantics $[\![s_1; s_2]\!](\rho)(\rho'')$
  - (Unnormalized) probability of beginning in state ρ and ending in state ρ'' after executing s_1 and then s_2

- Example:

$$[\![x \sim \texttt{flip}\ 0.1; y \sim \texttt{flip}\ 0.4]\!](\emptyset)([x \mapsto T, y \mapsto F]) = 0.1 \times 0.6$$

Sequence

- Breaking the example down more

$$[\![x \sim \texttt{flip}\ 0.1; y \sim \texttt{flip}\ 0.4]\!](\emptyset)([x \mapsto T, y \mapsto F]) = 0.1 \times 0.6$$

- Look at the first flip:

Initial state ρ

$$[\![x \sim \texttt{flip}\ 0.1]\!](\emptyset)([x \mapsto T]) = 0.1$$

- The second flip is *independent*

$$[\![y \sim \texttt{flip}\ 0.4]\!]([x \mapsto T])([x \mapsto T, y \mapsto F]) = 0.6$$

Intermediate state ρ'

Final state ρ''

Sequence

- Then, the probability of reaching ρ'' from ρ after executing $s_1; s_2$ is given by summing over all intermediate states ρ'

$$[\![s_1; s_2]\!](\rho)(\rho'') = \sum_{\rho' \in Env} [\![s_1]\!](\rho)(\rho') \times [\![s_2]\!](\rho')(\rho'').$$

If-statements

- We can simply evaluate the guard

$$[\![\texttt{if e } \{\texttt{s}_1\} \texttt{ else } \{\texttt{s}_2\}]\!](\rho)(\rho') = \begin{cases} [\![s_1]\!](\rho)(\rho') & \text{if } [\![e]\!](\rho) = T, \\ [\![s_2]\!](\rho)(\rho') & \text{otherwise.} \end{cases}$$

Semantics of Programs

- The semantics of a program is to map a program to the probability that the program *returns true*

$$\llbracket \texttt{s; return e} \rrbracket : [0, 1]$$

- (Ignoring observations), this can be computed in a manner following the definition of a random variable:

$$\llbracket \texttt{s; return e} \rrbracket = \sum_{\{\rho \in Env \mid \llbracket e \rrbracket \rho = T\}} \llbracket s \rrbracket(\emptyset)(\rho)$$

Observe statements

- Here we go *beyond* the modeling power of tables…

Cold?	**Fever?**	**Temp?**	**Runny nose?**	**Cough?**	**Pr?**
N	N	N	N	N	0.91266912
Y	Y	Y	Y	N	…

- For tables, conditioning is *exogenous to the model*
  - Happens "out of band"; done after modeling

Observe statements

- The syntax $[\![\texttt{observe e}]\!](\rho)(\rho')$ means "the unnormalized probability of transitioning to state ρ' from ρ given that e holds"

$$[\![\texttt{observe e}]\!](\rho)(\rho') = \begin{cases} 1 & \text{if } [\![e]\!](\rho) = T \text{ and } \rho = \rho', \\ 0 & \text{otherwise.} \end{cases}$$

- This is *unnormalized! Example…*

Observe statements

- Note that this is *unnormalized!*

$$[\![\texttt{x} \sim \texttt{flip 0.1; observe x}]\!](\emptyset)([x \mapsto T]) = 0.1$$

$$[\![\texttt{x} \sim \texttt{flip 0.1; observe x}]\!](\emptyset)([x \mapsto F]) = 0$$

Updating Semantics of Programs

- To give a semantics to programs with observations, we need to *renormalize*

Unnormalized probability of outputting true

$$[\![\texttt{x} \sim \texttt{flip 0.1; observe x; return x}]\!](T) = \frac{0.1}{0.1}$$

Normalizing constant

Updating Semantics of Programs

- To give a semantics to programs with observations, we need to *renormalize*

Unnormalized probability of outputting true

$$[\![\texttt{s; return e}]\!](v) = \frac{\sum_{\{\rho' \in Env \mid [\![e]\!](\rho) = T\}} [\![s]\!](\emptyset)(\rho')}{\sum_{\rho' \in Env} [\![s]\!](\emptyset)(\rho')}$$

Normalizing constant

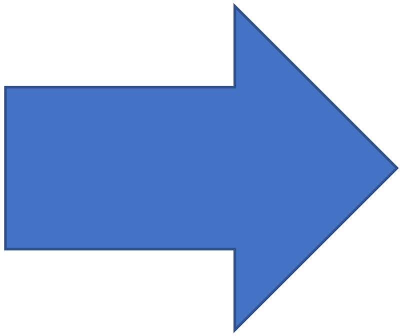
What makes a *good* modeling language?

- Expressive power
  - Conciseness
  - Completeness
- Accessibility and interpretability
- Tractability for inference

Tables Versus Programs

- How do we show one language is *complete* for another?
- Give a reduction: show that any table has a corresponding `SimPPL` statement

A	B	Pr
1	1	0.2
1	0	0.3
0	1	0.4
0	0	0.1



Tables Versus Programs

- How do we show one language is at least as concise as another?
- Give a polynomial-size reduction: show that writing tables as programs does not yield an exponentially large program
- To argue that `SimPPL` programs are more concise, give a small statement that requires an exponentially large table to represent its distribution

What makes a *good* modeling language?

- Expressive power
- Accessibility and interpretability
  - In the eyes of the beholder ☺
- Tractability for inference
  - Next time…

Inference Teaser

- Show that inference for `SimPPL` is NP-hard