KU LEUVEN

DEPARTMENT OF COMPUTER SCIENCE

Reactive Probabilistic Programming

Pedro Zuidberg Dos Martires, Sebastijan Dumančić
pedro.zuidbergdosmartires@cs.kuleuven.be, sebastijan.dumancic@cs.kuleuven.be

How to Handle Event-Driven Streams of Data within Probabilistic Programming?

Reactive Programming

An ad service provider has to decide which ads to display on a customer's screen. Future ad placements will then depend on a customer's actions on the website. Example code in **ReactiveX** (reactivex.io).

```
var clickResultSets =
    Mouseclicks.
        map(click =>
            getFromDatabase(click)
            takeUntil(mouseclicks)
        ).concatAll();

clickResultSets.forEach(
    ResultSet => updateClickResuls(resultSet);
```

Asynchronous and **external** events at discrete points in time, such as mouse clicks, drive the execution of a program.

Behaviors change continuously over time and are composable first-class citizens in the reactive programming paradigm.

Events refer to streams of value updates to time-dependent variables (behaviors). Events occur at discrete points in time and are composable first-class citizens.

Reactive Probabilistic Programming

Mapping the concepts of reactive programming to probabilistic programming gives us the first two components:

1. **Behaviors are random variables** whose value assignments change with a transition model.
2. **Events are observations** which interact with the random variables through (probabilistic) observations.

Additionally we identify a third component:

3. A **probabilistic planer** that decides which action to take given a probabilistic world state.

We propose a modularized structure:

- A **declarative module**: containing behaviors, events and the planer.
- An **imperative module**: defining the effect of actions.

We stress the need for APIs to probabilistic programming languages!

Proof of concept implemented in the existing probabilistic programming language Distributional Clauses.

https://bitbucket.org/problog/dc_problog
https://github.com/ML-KULeuven/PyDC

Deterministic Imperative Module

```
1  from pydc import DDC
2  #load DDC program and intialize 500 particles
3  ddc = DDC("weather_brussels_hmm.pl", 500)
4  #proceed one time step and query the state
5  ddc.step(observations=
6     "observation(activity(tintin))~=clean")
7  p_hot = ddc.query(
8     "current(temperature(brussels))>20")
9  #take decision
10 if p_hot>0.5: print("wear shorts!")
11 else: print("wear pants!")
```

Probabilistic Declarative Module

```
1  %facts
2  city(brussels) <- true. %true implies the city of Brussels exists in our databse
3  %initial state
4  weather(C):0 ~ finite([0.6:rainy,0.4:sunny]) <- city(C). %we initialize the weather at time step 0
5  %state model
6  temperature(C):t ~ gaussian(10,6) <- weather(C):t ~= rainy. %given rainy weather the mean of th temperature is 10 degrees Celsius
7  temperature(C):t ~ gaussian(24,8) <- weather(C):t ~= sunny. %and 24 for sunny weather
8  %transition model: here we describe how variables at time (t+1) depend on variables at time (t)
9  weather(C):t+1 ~ finite([0.7:rainy,0.3:sunny]) <- weather(C):t ~= rainy.
10 weather(C):t+1 ~ finite([0.4:rainy,0.6:sunny]) <- weather(C):t ~= sunny.
11 activity(tintin):t+1 ~ finite([0.1:walk,0.4:shop,0.5:clean]) <- weather(brussels):t+1 ~= rainy.
12 activity(tintin):t+1 ~ finite([0.6:walk,0.3:shop,0.1:clean]) <- weather(brussels):t+1 ~= sunny.
```

KU Leuven
Department of Computer Science
Celestijnenlaan 200A box 2402
3001 Leuven, Belgium

DTAI Declarative Languages & Artificial Intelligence

https://github.com/ML-KULeuven

chist-era