A smooth probabilistic extension of concurrent constraint programming

Romain Beauxis
INRIA Futurs and LIX, École Polytechnique
Denotational semantics and probabilities

Probabilistic programs?
- Model probabilistic programs and protocols
- Quantitative reasoning
- Gap of expressivity
- Probabilistic determinism

A denotational semantic?
- Abstract the meaning of a program from its actual implementation
- Modular and compositional representation of programs
- Verify the behaviour of a program without actually running it
- Implies some sort of determinism
Denotational semantics and probabilities

Probabilistic programs?
- Model probabilistic programs and protocols
- Quantitative reasoning
- Gap of expressivity
- Probabilistic determinism

A denotational semantic?
- Abstract the meaning of a program from its actual implementation
- Modular and compositional representation of programs
- Verify the behaviour of a program without actually running it
- Implies some sort of determinism
Denotational semantics and probabilities

**Probabilistic programs?**
- Model probabilistic programs and protocols
- Quantitative reasoning
- Gap of expressivity
- Probabilistic determinism

**A denotational semantic?**
- Abstract the meaning of a program from its actual implementation
- Modular and compositional representation of programs
- Verify the behaviour of a program without actually running it
- Implies some sort of determinism
Denotational semantics and probabilities

**Probabilistic programs?**
- Model probabilistic programs and protocols
- Quantitative reasoning
- Gap of expressivity
- Probabilistic determinism

**A denotational semantic?**
- Abstract the meaning of a program from its actual implementation
- Modular and compositional representation of programs
- Verify the behaviour of a program without actually running it
- Implies some sort of determinism
Denotational semantics and probabilities

**Probabilistic programs?**
- Model probabilistic programs and protocols
- Quantitative reasoning
- Gap of expressivity
- Probabilistic determinism

**A denotational semantic?**
- Abstract the meaning of a program from its actual implementation
- Modular and compositional representation of programs
- Verify the behaviour of a program without actually running it
- Implies some sort of determinism
Denotational semantics and probabilities

Probabilistic programs?
- Model probabilistic programs and protocols
- Quantitative reasoning
- Gap of expressivity
- Probabilistic determinism

A denotational semantic?
- Abstract the meaning of a program from its actual implementation
- Modular and compositional representation of programs
- Verify the behaviour of a program without actually running it
- Implies some sort of determinism
Denotational semantics and probabilities

Probabilistic programs?
- Model probabilistic programs and protocols
- Quantitative reasoning
- Gap of expressivity
- Probabilistic determinism

A denotational semantic?
- Abstract the meaning of a program from it’s actual implementation
- Modular and compositional representation of programs
- Verify the behaviour of a program without actually running it
- Implies some sort of determinism
Denotational semantics and probabilities

Probabilistic programs?

- Model probabilistic programs and protocols
- Quantitative reasoning
- Gap of expressivity
- Probabilistic determinism

A denotational semantic?

- Abstract the meaning of a program from its actual implementation
- Modular and compositional representation of programs
- Verify the behaviour of a program without actually running it
- Implies some sort of determinism
Denotational semantics and probabilities

Probabilistic programs?
- Model probabilistic programs and protocols
- Quantitative reasoning
- Gap of expressivity
- Probabilistic determinism

A denotational semantic?
- Abstract the meaning of a program from its actual implementation
- Modular and compositional representation of programs
- Verify the behaviour of a program without actually running it
- Implies some sort of determinism
Probabilistic Concurrent constraint programming

The Probabilistic Concurrent Constraint Programming (CCP+P) is a language where each process is equipped with a store of constraints to which it talks to. It enjoys the following features:

- Asynchronous language
- Permanently add constraints to the store
- Checks for the validity of a given constraint against current store
- A process may hide variables from other processes
- Confluent probabilistic states
Probabilistic Concurrent constraint programming

The Probabilistic Concurrent Constraint Programming (CCP+P) is a language where each process is equipped with a store of constraints to which it talks to. It enjoys the following features:

- Asynchronous language
- Permanently add constraints to the store
- Checks for the validity of a given constraint against current store
- A process may hide variables from other processes
- Confluent probabilistic states
The Probabilistic Concurrent Constraint Programming (CCP+P) is a language where each process is equipped with a store of constraints to which it talks to. It enjoys the following features:

- Asynchronous language
- Permanently add constraints to the store
- Checks for the validity of a given constraint against current store
- A process may hide variables from other processes
- Confluent probabilistic states
The Probabilistic Concurrent Constraint Programming (CCP+P) is a language where each process is equipped with a store of constraints to which it talks to. It enjoys the following features:

▶ Asynchronous language
▶ Permanently add constraints to the store
▶ Checks for the validity of a given constraint against current store
▶ A process may hide variables from other processes
▶ Confluent probabilistic states
Probabilistic Concurrent constraint programming

The Probabilistic Concurrent Constraint Programming (CCP+P) is a language where each process is equipped with a store of constraints to which it talks to. It enjoys the following features:

- Asynchronous language
- Permanently add constraints to the store
- Checks for the validity of a given constraint against current store
- A process may hide variables from other processes
- Confluent probabilistic states
The Probabilistic Concurrent Constraint Programming (CCP+P) is a language where each process is equipped with a store of constraints to which it talks to. It enjoys the following features:

- Asynchronous language
- Permanently add constraints to the store
- Checks for the validity of a given constraint against current store
- A process may hide variables from other processes
- Confluent probabilistic states
Valuations

\[ \nu_k(\sum_i U_i) = \sum_i \nu_k(U_i) \]

\[ \nu_\infty(\sum_i U_i) \neq \sum_i \nu_\infty(U_i) \]

\[ \nu_k(U \cup V) = \nu_k(U) + \nu_k(V) \]

\[ \nu_\infty(U \cup V) = \nu_\infty(U) + \nu_\infty(V) \]

Defined on open set instead of \( \sigma \)-algebra

Simple valuations:

\[ \sum_i p_i \delta_{x_i}(O) = \sum_{x_i \in O} p_i \]
Valuations

\[ \nu_k(\sum_i U_i) = \sum_i \nu_k(U_i) \]

\[ \nu_{\infty}(\sum_i U_i) \neq \sum_i \nu_{\infty}(U_i) \]

\[ \nu_k(U \cup V) = \nu_k(U) + \nu_k(V) \]

\[ \nu_{\infty}(U \cup V) = \nu_{\infty}(U) + \nu_{\infty}(V) \]

Defined on open set instead of \( \sigma \)-algebra

Simple valuations:

\[ \sum_i p_i \delta_{x_i}(O) = \sum_{x_i \in O} p_i \]
Valuations

\[ \nu_k(\sum_i U_i) = \sum_i \nu_k(U_i) \]

\[ \nu_\infty(\sum_i U_i) \neq \sum_i \nu_\infty(U_i) \]

\[ \nu_k(U \cup V) = \nu_k(U) + \nu_k(V) \]

\[ \nu_\infty(U \cup V) = \nu_\infty(U) + \nu_\infty(V) \]

Defined on open set instead of \( \sigma \)-algebra

Simple valuations:

\[ \sum_i p_i \delta_{x_i}(O) = \sum_{x_i \in O} p_i \]
Valuations

\[ \nu_k(\sum_i U_i) = \sum_i \nu_k(U_i) \]
\[ \nu_\infty(\sum_i U_i) \neq \sum_i \nu_\infty(U_i) \]
\[ \nu_k(U \cup V) = \nu_k(U) + \nu_k(V) \]
\[ \nu_\infty(U \cup V) = \nu_\infty(U) + \nu_\infty(V) \]

Defined on open set instead of \( \sigma \)-algebra

Simple valuations:

\[ \sum_i p_i \delta_{x_i}(O) = \sum_{x_i \in O} p_i \]
Valuations

\[ \nu_k(\sum_i U_i) = \sum_i \nu_k(U_i) \]

\[ \nu_\infty(\sum_i U_i) \neq \sum_i \nu_\infty(U_i) \]

\[ \nu_k(U \cup V) = \nu_k(U) + \nu_k(V) \]

\[ \nu_\infty(U \cup V) = \nu_\infty(U) + \nu_\infty(V) \]

Defined on open set instead of \( \sigma \)-algebra

Simple valuations:

\[ \sum_i p_i \delta_{x_i}(O) = \sum_{x_i \in O} p_i \]
Valuations

\[ \nu_k(\sum_i U_i) = \sum_i \nu_k(U_i) \]

\[ \nu_\infty(\sum_i U_i) \neq \sum_i \nu_\infty(U_i) \]

\[ \nu_k(U \cup V) = \nu_k(U) + \nu_k(V) \]

\[ \nu_\infty(U \cup V) = \nu_\infty(U) + \nu_\infty(V) \]

Defined on open set instead of \( \sigma \)-algebra

Simple valuations:

\[ \sum_i p_i \delta_{x_i}(O) = \sum_{x_i \in O} p_i \]
Valuations

\[ \nu_k(\sum_i U_i) = \sum_i \nu_k(U_i) \]

\[ \nu_\infty(\sum_i U_i) \neq \sum_i \nu_\infty(U_i) \]

\[ \nu_k(U \cup V) = \nu_k(U) + \nu_k(V) \]

\[ \nu_\infty(U \cup V) = \nu_\infty(U) + \nu_\infty(V) \]

Defined on open set instead of \( \sigma \)-algebra

Simple valuations:

\[ \sum_i p_i \delta_{x_i}(O) = \sum_{x_i \in O} p_i \]
Vector space of valuations

We lift the original constraint lattice to the vector space of valuations.

- Constraints $c \longrightarrow$ simple valuations $\sum_i x_i \delta_{c_i}$
- Closure operator $\longrightarrow$ linear closure operator
- Set of fixed points $\longrightarrow$ vector space of fixed points
- Directed sequence of constraints $\longrightarrow$ directed sequence of valuations
Vector space of valuations

We lift the original constraint lattice to the vector space of valuations.

- Constraints $c \rightarrow$ simple valuations $\sum_i x_i \delta_{c_i}$
- Closure operator $\rightarrow$ linear closure operator
- Set of fixed points $\rightarrow$ vector space of fixed points
- Directed sequence of constraints $\rightarrow$ directed sequence of valuations
Vector space of valuations

We lift the original constraint lattice to the vector space of valuations.

- Constraints $c \rightarrow$ simple valuations $\sum_i x_i \delta_{c_i}$
- Closure operator $\rightarrow$ linear closure operator
- Set of fixed points $\rightarrow$ vector space of fixed points
- Directed sequence of constraints $\rightarrow$ directed sequence of valuations
Vector space of valuations

We lift the original constraint lattice to the vector space of valuations.

- **Constraints** $c \rightarrow$ simple valuations $\sum_i x_i \delta_{c_i}$
- **Closure operator** $\rightarrow$ linear closure operator
- **Set of fixed points** $\rightarrow$ vector space of fixed points
- **Directed sequence of constraints** $\rightarrow$ directed sequence of valuations
Vector space of valuations

We lift the original constraint lattice to the vector space of valuations.

- Constraints $c \rightarrow$ simple valuations $\sum_i x_i \delta c_i$
- Closure operator $\rightarrow$ linear closure operator
- Set of fixed points $\rightarrow$ vector space of fixed points
- Directed sequence of constraints $\rightarrow$ directed sequence of valuations
Future works

▶ Applications to various protocols
  ▶ Finite runs. Ex.: Dinning cryptographers
  ▶ Infinite runs. Ex.: Onion routing
▶ Consider unfair executions
▶ Similar extensions for other languages
Future works

- Applications to various protocols
  - Finite runs. Ex.: Dinning cryptographers
  - Infinite runs. Ex.: Onion routing
- Consider unfair executions
- Similar extensions for other languages
Future works

- Applications to various protocols
  - Finite runs. Ex.: Dinning cryptographers
  - Infinite runs. Ex.: Onion routing
- Consider unfair executions
- Similar extensions for other languages
Future works

- Applications to various protocols
  - Finite runs. Ex.: Dinning cryptographers
  - Infinite runs. Ex.: Onion routing
- Consider unfair executions
  - Similar extensions for other languages
Future works

- Applications to various protocols
  - Finite runs. Ex.: Dinning cryptographers
  - Infinite runs. Ex.: Onion routing
- Consider unfair executions
- Similar extensions for other languages