AlgSDP: Algebra of Sequential Decision Problems
formalised in Agda

Robert Krook **Patrik Jansson**

2020-01-06, WG2.1 #79 in Otterlo, NL

Abstract

Sequential decision problems are a well established concept in decision theory, with the Bellman equation as a popular choice for describing them. Botta, Jansson, Ionescu have formalised the notion of such problems in Idris (presented by Jansson at #75 Uruguay, by Botta at #77 Brandenburg). Here we focus on an Algebra of SDPs (in Agda): combinators for building more complex SDPs from simpler ones.

AlgSDP by example in one slide

A 1D-coord. syst. with $\mathbb{N}$ as state and $+\mathit{1}$, $\mathit{0}$, and $-\mathit{1}$ as actions.

$$p \;:\; \mathit{SDProc}$$

AlgSDP by example in one slide

A 1D-coord. syst. with $\mathbb{N}$ as state and $+\mathit{1}$, $\mathit{0}$, and $-\mathit{1}$ as actions.

$$p \;:\; SDProc$$

We define a product to enable reusing p in a 2D setting:

$$\_\times_{SDP}\_ \;:\; SDProc \to SDProc \to SDProc$$
$$p^2 \;=\; p \;\times_{SDP}\; p$$

AlgSDP by example in one slide

A 1D-coord. syst. with $\mathbb{N}$ as state and $+\mathit{1}$, $\mathit{0}$, and $-\mathit{1}$ as actions.

$$p \; : \; SDProc$$

We define a product to enable reusing p in a 2D setting:

$$\begin{aligned} \_\times_{SDP}\_ \;&:\; SDProc \to SDProc \to SDProc \\ p^2 \;&=\; p \;\times_{SDP}\; p \end{aligned}$$

Both p and p^2 use a fixed state space, but we can also handle time dependent processes (for example p' of type $SDProcT$).

$$\begin{aligned} \_\times^{T}_{SDP}\_ \;&:\; SDProcT \to SDProcT \to SDProcT \\ embed \;&:\; SDProc \to SDProcT \\ p^{2'} \;&=\; p' \;\times^{T}_{SDP}\; (embed\; p) \\ p^3 \;&=\; p^2 \;\times_{SDP}\; p \end{aligned}$$

AlgSDP by example in one slide

A 1D-coord. syst. with $\mathbb{N}$ as state and $+1$, 0, and -1 as actions.

$$p \ : \ SDProc$$

We define a product to enable reusing p in a 2D setting:

$$\begin{aligned} \_\times_{SDP}\_ \ &: \ SDProc \to SDProc \to SDProc \\ p^2 \ &= \ p \ \times_{SDP} \ p \end{aligned}$$

Both p and p^2 use a fixed state space, but we can also handle time dependent processes (for example p' of type $SDProcT$).

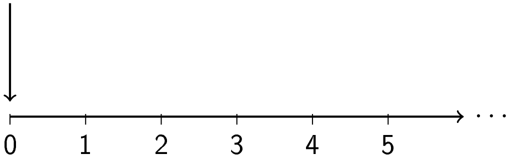
$$\begin{aligned} \_\times^T_{SDP}\_ \ &: \ SDProcT \to SDProcT \to SDProcT \\ embed \ &: \ SDProc \to SDProcT \\ p^{2'} \ &= \ p' \ \times^T_{SDP} \ (embed \ p) \\ p^3 \ &= \ p^2 \ \times_{SDP} \ p \end{aligned}$$

Final example: a process that moves either in 3D or in 2D.

$$\begin{aligned} \_\uplus^T_{SDP}\_ \ &: \ SDProcT \to SDProcT \to SDProcT \\ game \ &= \ p^{2'} \ \uplus^T_{SDP} \ (embed \ p^3) \end{aligned}$$

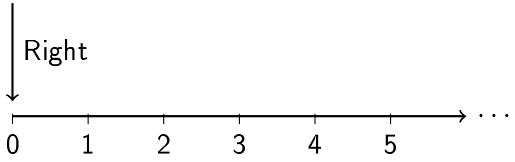
You could think of this as choosing a map in a game.

Example: 1-dimensional coordinate system



$$\begin{aligned} \textit{1d-state} &: \textit{Set} \\ \textit{1d-state} &= \mathbb{N} \end{aligned}$$

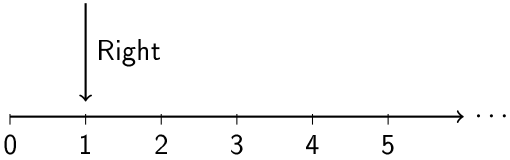
Example: 1-dimensional coordinate system



$$
\begin{aligned}
\textit{1d-state} &: \quad \textit{Set} \\
\textit{1d-state} &= \mathbb{N}
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \textit{1d-control} : \textit{1d-state} \to \textit{Set}\ \textbf{where} \\
&\quad \textit{Right} : \{n : \textit{1d-state}\} \to \textit{1d-control}\ n \\
&\quad \textit{Stay} \;\; : \{n : \textit{1d-state}\} \to \textit{1d-control}\ n \\
&\quad \textit{Left} \;\; : \{n : \textit{1d-state}\} \to \textit{1d-control}\ (\textit{suc}\ n)
\end{aligned}
$$

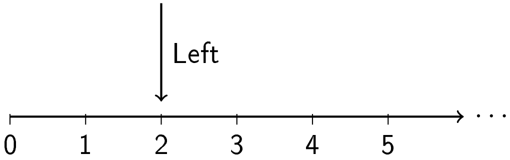
Example: 1-dimensional coordinate system



$$
\begin{aligned}
\mathit{1d\text{-}state} &: \mathit{Set} \\
\mathit{1d\text{-}state} &= \mathbb{N}
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathit{1d\text{-}control} : \mathit{1d\text{-}state} \to \mathit{Set}\ \textbf{where} \\
&\quad \mathit{Right} : \{n : \mathit{1d\text{-}state}\} \to \mathit{1d\text{-}control}\ n \\
&\quad \mathit{Stay} : \{n : \mathit{1d\text{-}state}\} \to \mathit{1d\text{-}control}\ n \\
&\quad \mathit{Left} : \{n : \mathit{1d\text{-}state}\} \to \mathit{1d\text{-}control}\ (\mathit{suc}\ n)
\end{aligned}
$$

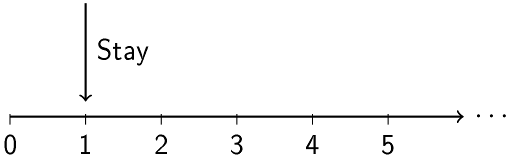
Example: 1-dimensional coordinate system



$$\mathit{1d\text{-}state} : \mathit{Set}$$
$$\mathit{1d\text{-}state} = \mathbb{N}$$

$$
\begin{aligned}
&\textbf{data}\ \mathit{1d\text{-}control} : \mathit{1d\text{-}state} \to \mathit{Set}\ \textbf{where}\\
&\quad \mathit{Right} : \{n : \mathit{1d\text{-}state}\} \to \mathit{1d\text{-}control}\ n\\
&\quad \mathit{Stay} : \{n : \mathit{1d\text{-}state}\} \to \mathit{1d\text{-}control}\ n\\
&\quad \mathit{Left} : \{n : \mathit{1d\text{-}state}\} \to \mathit{1d\text{-}control}\ (\mathit{suc}\ n)
\end{aligned}
$$

Example: 1-dimensional coordinate system



$$\mathit{1d\text{-}state} : \mathit{Set}$$
$$\mathit{1d\text{-}state} = \mathbb{N}$$

data *1d-control* : *1d-state* → *Set* **where**
  Right : {*n* : *1d-state*} → *1d-control n*
  Stay : {*n* : *1d-state*} → *1d-control n*
  Left : {*n* : *1d-state*} → *1d-control* (*suc n*)

Example: 1-dimensional coordinate system

$$\mathit{1d\text{-}state} \;:\; \mathit{Set}$$
$$\mathit{1d\text{-}state} \;=\; \mathbb{N}$$

$$
\begin{aligned}
&\textbf{data}\ \mathit{1d\text{-}control} : \mathit{1d\text{-}state} \to \mathit{Set}\ \textbf{where}\\
&\quad \mathit{Right} : \{n : \mathit{1d\text{-}state}\} \to \mathit{1d\text{-}control}\ n\\
&\quad \mathit{Stay} : \{n : \mathit{1d\text{-}state}\} \to \mathit{1d\text{-}control}\ n\\
&\quad \mathit{Left} : \{n : \mathit{1d\text{-}state}\} \to \mathit{1d\text{-}control}\ (\mathit{suc}\ n)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{1d\text{-}step} : (x : \mathit{1d\text{-}state}) \to \mathit{1d\text{-}control}\ x \to \mathit{1d\text{-}state}\\
&\mathit{1d\text{-}step}\ x\ \mathit{Right} = \mathit{suc}\ x\\
&\mathit{1d\text{-}step}\ x\ \mathit{Stay} = x\\
&\mathit{1d\text{-}step}\ (\mathit{suc}\ x)\ \mathit{Left} = x
\end{aligned}
$$

Sequential Decision Process

```
record SDProc : Set1 where
  constructor SDP
  field
    State    : Set
```

Sequential Decision Process

```
record SDProc : Set1 where
  constructor SDP
  field
    State   : Set
    Control : State → Set
```

Sequential Decision Process

```
record SDProc : Set1 where
  constructor SDP
  field
    State   : Set
    Control : State → Set
    step    : (x : State) → Control x → State
```

Sequential Decision Process

```
record SDProc : Set1 where
  constructor SDP
  field
    State   : Set
    Control : State → Set
    step    : (x : State) → Control x → State
```

Our example:

```
1d-sys : SDProc
1d-sys = SDP 1d-state 1d-control 1d-step
```

Sequential Decision Problem

In a sequential decision **problem** there is also a fourth field *reward*:

```
record SDProb : Set1 where
  constructor SDP
  field
    State   : Set
    Control : State → Set
    step    : (x : State) → Control x → State
    reward  : (x : State) → Control x → Val
```

(where *Val* is often $\mathbb{R}$).

- The Seq. Dec. Problem is: find a sequence of controls that maximises the sum of rewards.
- Or, in more realistic settings with uncertainty, finding a sequence of *policies* which maximises the *expected* reward.
- Rewards, and problems, are not the focus of this talk but are mentioned for completeness.

Policy

In general:

$$Policy \ : \ (S \ : \ Set) \to ((s \ : \ S) \to Set) \to Set$$
$$Policy \ S \ C \ = \ (s \ : \ S) \to C \ s$$

Policy

In general:

$$\begin{aligned} &\mathit{Policy} : (S : \mathit{Set}) \to ((s : S) \to \mathit{Set}) \to \mathit{Set} \\ &\mathit{Policy}\ S\ C\ = (s : S) \to C\ s \end{aligned}$$

Specialised:

$$\begin{aligned} \mathit{1d\text{-}Policy} &: \ \mathit{Set} \\ \mathit{1d\text{-}Policy} &= \mathit{Policy}\ \mathit{1d\text{-}state}\ \mathit{1d\text{-}control} \\ \text{--}\quad &= (x : \mathit{1d\text{-}state}) \to \mathit{1d\text{-}control}\ x \end{aligned}$$

Policy

Specialised:

$$
\begin{aligned}
\mathit{1d\text{-}Policy} &: \mathit{Set} \\
\mathit{1d\text{-}Policy} &= \mathit{Policy}\ \mathit{1d\text{-}state}\ \mathit{1d\text{-}control} \\
\text{--} &= (x : \mathit{1d\text{-}state}) \to \mathit{1d\text{-}control}\ x
\end{aligned}
$$

Example policies:

$$
\begin{aligned}
&\mathit{right}\ \mathit{stay}\ \mathit{tryleft} : \mathit{1d\text{-}Policy} \\
&\mathit{right}\ \_ = \mathit{Right} \\
&\mathit{stay}\ \_ = \mathit{Stay} \\
&\mathit{tryleft}\ \mathit{zero} = \mathit{Stay} \\
&\mathit{tryleft}\ (\mathit{suc}\ s) = \mathit{Left}
\end{aligned}
$$

Policy

Example policies:

$$
\begin{array}{lcl}
\textit{right stay tryleft} & : & \textit{1d-Policy} \\
\textit{right} \;\; \_ & = & \textit{Right} \\
\textit{stay} \;\; \_ & = & \textit{Stay} \\
\textit{tryleft zero} & = & \textit{Stay} \\
\textit{tryleft}\ (\textit{suc}\ s) & = & \textit{Left}
\end{array}
$$

A family of policies (to move *towards* a particular goal coordinate):

$$
\begin{array}{l}
\textit{towards} : \mathbb{N} \rightarrow \textit{1d-Policy} \\
\textit{towards goal n}\ \textbf{with}\ \textit{compare n goal} \\
\ldots \mid \textit{less}\ \_\ \_ \quad = \textit{Right} \\
\ldots \mid \textit{equal}\ \_ \quad = \textit{Stay} \\
\ldots \mid \textit{greater}\ \_\ \_ = \textit{Left}
\end{array}
$$

Back to Processes - now with abbreviations

$$
\begin{array}{l}
\textbf{record } \mathit{SDProc} \ : \ \mathit{Set1} \textbf{ where} \\
\quad \mathit{constructor\ SDP} \\
\quad \textbf{field } \mathit{State} \ : \ \mathit{Set} \\
\qquad\quad \mathit{Control} \ : \ \mathit{Con\ State} \\
\qquad\quad \mathit{step} \ : \ \mathit{Step\ State\ Control}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Con} \ : \ \mathit{Set} \to \mathit{Set}_1 \\
\mathit{Con}\ S \ = \ S \to \mathit{Set}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Step} \ : \ (S \ : \ \mathit{Set}) \ \to \ \mathit{Con}\ S \ \to \ \mathit{Set} \\
\mathit{Step}\ S\ C \ = \ (s \ : \ S) \ \to \ C\ s \ \to \ S
\end{array}
$$

$$
\begin{array}{l}
\mathit{Policy} \ : \ (S \ : \ \mathit{Set}) \to ((s \ : \ S) \to \mathit{Set}) \to \mathit{Set} \\
\mathit{Policy}\ S\ C \ = \ (s \ : \ S) \to C\ s
\end{array}
$$

Trajectory

Here $\#_{st}$, $\#_{c}$, $\#_{sf}$ extract the different components of an SDP.

$$
\begin{aligned}
&\textit{trajectory} \;:\; (p : \textit{SDProc}) \to \{n : \mathbb{N}\} \to \\
&\qquad \to \textit{Vec}\ (\textit{Policy}\ (\#_{st}\ p)\ (\#_{c}\ p))\ n \\
&\qquad \to \#_{st}\ p \to \textit{Vec}\ (\#_{st}\ p)\ n \\
&\textit{trajectory sys}\ []\ x_0 = [] \\
&\textit{trajectory sys}\ (p :: ps)\ x_0 = x_1 :: \textit{trajectory sys ps}\ x_1 \\
&\quad \textbf{where}\ x_1 : \#_{st}\ \textit{sys} \\
&\qquad\quad x_1 = (\#_{sf}\ \textit{sys})\ x_0\ (p\ x_0)
\end{aligned}
$$

Example:

$$
\begin{aligned}
&\textit{pseq} = \textit{tryleft} :: \textit{tryleft} :: \textit{right} :: \textit{stay} :: \textit{right} :: [] \\
&\textit{test1} = \textit{trajectory 1d-sys pseq 0} \\
&\quad \text{--} = 0 :: 0 :: 1 :: 1 :: 2 :: []
\end{aligned}
$$

In an applied setting many trajectories would be computed to explore the system behaviour.

The Product of SDPs

$$
\begin{aligned}
\_ \times_{SDP} \_ \; &: \; SDProc \to SDProc \to SDProc \\
(SDP\ S_1\ C_1\ sf_1) \; \times_{SDP} \; (SDP\ S_2\ C_2\ sf_2) \\
&= \; SDP\ (S_1 \times S_2)\ (C_1 \; \times_C \; C_2)\ (sf_1 \; \times_{sf} \; sf_2)
\end{aligned}
$$

The Product of SDPs

$$
\begin{aligned}
&\_\times_{SDP}\_ \;:\; SDProc \to SDProc \to SDProc\\
&(SDP\ S_1\ C_1\ sf_1)\ \times_{SDP}\ (SDP\ S_2\ C_2\ sf_2)\\
&\quad = SDP\ (S_1 \times S_2)\ (C_1\ \times_C\ C_2)\ (sf_1\ \times_{sf}\ sf_2)
\end{aligned}
$$

$$
\begin{aligned}
&Con \;:\; Set \to Set_1\\
&Con\ S \;=\; S \to Set
\end{aligned}
$$

$$
\begin{aligned}
&\_\times_C\_ \;:\; \{S_1\ S_2 \;:\; Set\} \to\\
&\qquad Con\ S_1 \;\to\; Con\ S_2 \;\to\; Con\ (S_1 \times S_2)\\
&(C_1\ \times_C\ C_2)\ (s_1\ ,\ s_2) \;=\; C_1\ s_1 \times C_2\ s_2
\end{aligned}
$$

The Product of SDPs

$$
\begin{array}{l}
\_\times_{SDP}\_ \;:\; SDProc \to SDProc \to SDProc \\
(SDP\ S_1\ C_1\ sf_1)\ \times_{SDP}\ (SDP\ S_2\ C_2\ sf_2) \\
\quad =\ SDP\ (S_1 \times S_2)\ (C_1\ \times_C\ C_2)\ (sf_1\ \times_{sf}\ sf_2)
\end{array}
$$

$$
\begin{array}{l}
Con \;:\; Set \to Set_1 \\
Con\ S\ =\ S \to Set \\
\_\times_C\_ \;:\; \{S_1\ S_2 \;:\; Set\} \to \\
\qquad\qquad Con\ S_1 \ \to\ Con\ S_2 \ \to\ Con\ (S_1 \times S_2) \\
(C_1\ \times_C\ C_2)\ (s_1\ ,\ s_2)\ =\ C_1\ s_1 \times C_2\ s_2
\end{array}
$$

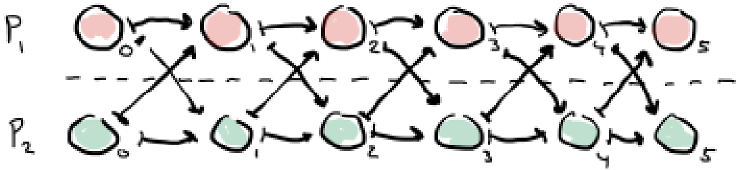
$$
\begin{array}{l}
Step \;:\; (S \;:\; Set) \ \to\ Con\ S \ \to\ Set \\
Step\ S\ C\ =\ (s \;:\; S) \ \to\ C\ s \ \to\ S \\
\_\times_{sf}\_ \;:\; \{S_1\ S_2 \;:\; Set\}\ \{C_1 \;:\; Con\ S_1\}\ \{C_2 \;:\; Con\ S_2\} \\
\qquad\quad \to\ Step\ S_1\ C_1 \ \to\ Step\ S_2\ C_2 \\
\qquad\quad \to\ Step\ (S_1 \times S_2)\ (C_1\ \times_C\ C_2) \\
(sf_1\ \times_{sf}\ sf_2)\ (s_1\ ,\ s_2)\ (c_1\ ,\ c_2)\ =\ (sf_1\ s_1\ c_1\ ,\ sf_2\ s_2\ c_2)
\end{array}
$$

The Product of SDPs

$$
\begin{aligned}
&\_\times_{SDP}\_ \;:\; SDProc \rightarrow SDProc \rightarrow SDProc\\
&(SDP\ S_1\ C_1\ sf_1)\ \times_{SDP}\ (SDP\ S_2\ C_2\ sf_2)\\
&\quad = SDP\ (S_1 \times S_2)\ (C_1\ \times_C\ C_2)\ (sf_1\ \times_{sf}\ sf_2)
\end{aligned}
$$

$$
\begin{aligned}
&\_\times_{sf}\_ \;:\; \{S_1\ S_2\ :\ Set\}\ \{C_1\ :\ Con\ S_1\}\ \{C_2\ :\ Con\ S_2\}\\
&\qquad \rightarrow\ Step\ S_1\ C_1\ \rightarrow\ Step\ S_2\ C_2\\
&\qquad \rightarrow\ Step\ (S_1 \times S_2)\ (C_1\ \times_C\ C_2)\\
&(sf_1\ \times_{sf}\ sf_2)\ (s_1\ ,\ s_2)\ (c_1\ ,\ c_2)\ =\ (sf_1\ s_1\ c_1\ ,\ sf_2\ s_2\ c_2)
\end{aligned}
$$

Example: $P_1\ \times_{SDP}\ P_2$



Product example

Example:

$$\mathit{2d\text{-}system} = \mathit{1d\text{-}sys} \times_{SDP} \mathit{1d\text{-}sys}$$

Now *2d-system* is a process of two dimensions rather than one:

$$\begin{aligned}
\mathit{pseq} &= \mathit{tryleft} :: \mathit{tryleft} :: \mathit{right} :: \mathit{stay} :: \mathit{right} :: [] \\
\mathit{2d\text{-}pseq} &= \mathit{zipWith}\ _\times_P_\ \mathit{pseq}\ \mathit{pseq} \\
\mathit{test2} &= \mathit{trajectory}\ \mathit{2d\text{-}system}\ \mathit{2d\text{-}pseq}\ (0\ ,\ 5) \\
\text{--} &= (0\ ,\ 4) :: (0\ ,\ 3) :: (1\ ,\ 4) :: (1\ ,\ 4) :: (2\ ,\ 5) :: []
\end{aligned}$$

where $\_\times_P\_$ is a combinator for policies.

Zero and One

$$
\begin{aligned}
&\mathit{zero} \;:\; \mathit{SDProc} \\
&\mathit{zero} \;=\; \textbf{record}\;\{ \\
&\quad \mathit{State} \;\;=\; \bot; \\
&\quad \mathit{Control} \;=\; \lambda\, \mathit{state} \;\to\; \bot; \\
&\quad \mathit{step} \;\;=\; \lambda\, \mathit{state} \;\to\; \lambda\, \mathit{control} \;\to\; \mathit{state}\,\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{unit} \;:\; \mathit{SDProc} \\
&\mathit{unit} \;=\; \textbf{record}\;\{ \\
&\quad \mathit{State} \;\;=\; \top; \\
&\quad \mathit{Control} \;=\; \lambda\, \mathit{state} \;\to\; \top; \\
&\quad \mathit{step} \;\;=\; \lambda\, \mathit{state} \;\to\; \lambda\, \mathit{control} \;\to\; \mathit{tt}\,\}
\end{aligned}
$$

Coproduct combinator

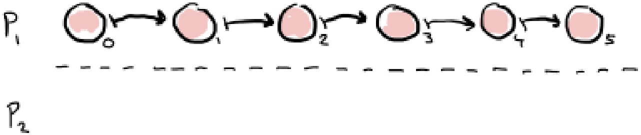
$$
\begin{aligned}
&\_\uplus_{SDP}\_ \ : \ SDProc \to SDProc \to SDProc\\
&SDP\ S_1\ C_1\ sf_1\ \uplus_{SDP}\ SDP\ S_2\ C_2\ sf_2\\
&\quad = \ SDP\ (S_1 \uplus S_2)\ (C_1\ \uplus_C\ C_2)\ (sf_1\ \uplus_{sf}\ sf_2)
\end{aligned}
$$

$$
\begin{aligned}
&\_\uplus_C\_ \ : \ \{S_1\ S_2\ :\ Set\}\\
&\qquad \to Con\ S_1 \to Con\ S_2 \to Con\ (S_1 \uplus S_2)\\
&(C_1\ \uplus_C\ C_2)\ (inj_1\ s_1) \ = \ C_1\ s_1\\
&(C_1\ \uplus_C\ C_2)\ (inj_2\ s_2) \ = \ C_2\ s_2
\end{aligned}
$$

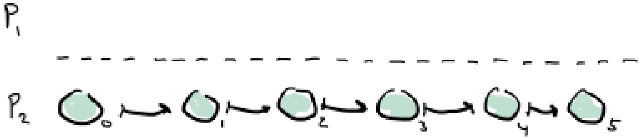
$$
\begin{aligned}
&\_\uplus_{sf}\_ \ : \ \{S_1\ S_2\ :\ Set\}\\
&\qquad \to \{C_1\ :\ Con\ S_1\} \to \{C_2\ :\ Con\ S_2\}\\
&\qquad \to Step\ S_1\ C_1 \to Step\ S_2\ C_2\\
&\qquad \to Step\ (S_1 \uplus S_2)\ (C_1\ \uplus_C\ C_2)\\
&(sf_1\ \uplus_{sf}\ sf_2)\ (inj_1\ s_1)\ c_1 \ = \ inj_1\ (sf_1\ s_1\ c_1)\\
&(sf_1\ \uplus_{sf}\ sf_2)\ (inj_2\ s_2)\ c_2 \ = \ inj_2\ (sf_2\ s_2\ c_2)
\end{aligned}
$$

Coproduct combinator example

Left injection:

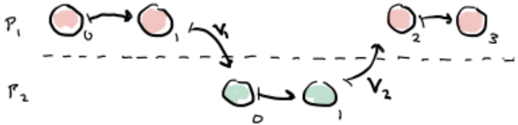


Right injection:



Yielding coproduct example

Illustration: It is capable of switching between the two processes, as illustrated by the calls to *v1* and *v2*.



With a combinator such as this one could you model e.g a two player game.
The processes would be the players and the combined process allows each to take turns making their next move.

Yielding coproduct code

$$
\begin{aligned}
\_\uplus^m_C\_ \ &: \ \{S_1\ S_2 \ : \ Set\} \\
&\quad \rightarrow Con\ S_1 \rightarrow Con\ S_2 \rightarrow Con\ (S_1 \uplus S_2) \\
(C_1\ \ \uplus^m_C\ \ C_2)\ (inj_1\ s_1) \ &= \ Maybe\ (C_1\ s_1) \\
(C_1\ \ \uplus^m_C\ \ C_2)\ (inj_2\ s_2) \ &= \ Maybe\ (C_2\ s_2)
\end{aligned}
$$

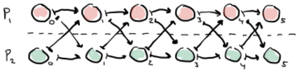
$$
\begin{aligned}
\_\rightleftarrows\_ \ &: \ (S_1\ S_2 \ : \ Set) \rightarrow Set \\
s_1\ \rightleftarrows\ s_2 \ &= \ (s_1\ \rightarrow\ s_2) \times (s_2\ \rightarrow\ s_1)
\end{aligned}
$$

$$
\begin{aligned}
&\uplus^m_{sf}\ \ : \ \{S_1\ S_2 \ : \ Set\}\ \{C_1 \ : \ Con\ S_1\}\ \{C_2 \ : \ Con\ S_2\} \\
&\qquad \rightarrow (S_1\ \rightleftarrows\ S_2) \\
&\qquad \rightarrow Step\ S_1\ \ C_1 \rightarrow Step\ S_2\ \ C_2 \\
&\qquad \rightarrow Step\ (S_1 \uplus S_2)\ (C_1\ \ \uplus^m_C\ \ C_2) \\
&\uplus^m_{sf}\ \ \_ \qquad\quad sf_1\ sf_2\ (inj_1\ s_1)\ (just\ c) \ = \ inj_1\ (sf_1\ s_1\ c) \\
&\uplus^m_{sf}\ \ \_ \qquad\quad sf_1\ sf_2\ (inj_2\ s_2)\ (just\ c) \ = \ inj_2\ (sf_2\ s_2\ c) \\
&\uplus^m_{sf}\ \ (v_1\ ,\ \_)\ sf_1\ sf_2\ (inj_1\ s_1)\ nothing \ = \ inj_2\ (v_1\ s_1) \\
&\uplus^m_{sf}\ \ (\_\ ,\ v_2)\ sf_1\ sf_2\ (inj_2\ s_2)\ nothing \ = \ inj_1\ (v_2\ s_2)
\end{aligned}
$$

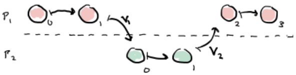
$$
syntax\ \ \uplus^m_{sf}\ \ r\ sf_1\ sf_2 \ = \ sf_1\ \langle\ r\ \rangle\ sf_2
$$

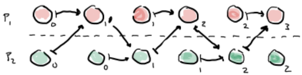
Summary

- It is possible to implement an algebra of SDPs
- Products are immediately useful



- Plain coproducts — not so much
- Many variants possible: yielding coproducts, interleaving product, etc.





Time-depedent, monadic cases left as exercises for the audience;-)