# Agent Discrete Decision Space

by Sven Nilsen, 2018

*In this paper I introduce a discrete space that enumerates all agent's isolated decisions. This can be used for computational simulations, analytic study, or algorithmic inference in AI safety research.*

A Discrete Space is a parametric algorithm that maps data structures to and from natural numbers. This property makes it possible to count and enumerate (walk through) the space from some dimension parameters. In this paper I use the "discrete" library on crates.io for the Rust programming language.

An Agent Discrete Decision Space is the following type:

```
(Either<DirectedContext, DimensionN>, DimensionN)
```

Using the "discrete" library on crates.io, one can construct the space in Rust by following:

```
let context: (Either<DirectedContext, DimensionN>, DimensionN) = Construct::new();
```

For example code, see "examples/world_agent.rs" in the repository (use the link on crates.io).

The dimension parameters, of this space are the following:

```
((mental, vec![mental[0], mental[1], .., directions]), grid_size)
```

1. `mental`: An N-vector describing the parallel "mental" states of the agent
2. `directions`: A number counting the directions or physical choices the agent can do
3. `grid_size`: An N-vector describing the size of the world (assumed to be an N-dimensional grid)

Explaining the parameters: The agent makes either (`Either`) a mental state change (`DirectedContext`) or a physical change (`DimensionN`) that includes the mental state. For every such state, the agent can be located anywhere in the world (`DimensionN`).

The `DirectionContext` space connects every N-dimensional state to every other N-dimensional state that differs by a value in one dimension. This means that one can adjust the mental dimensions to all parallel mental states.

For example:

> happy, sad, angry, fear, disgust (emotions)
> play, research, work, plan, protect (behavior)

An agent can be happy while working, sad while protecting etc.

Given some configuration state of emotions and behavior, the agent changes to another mental state. This means either changing some emotion or some behavior. E.g. `(happy, play)` might change to `(happy, work)`.

In code, this is represented as numbers ‘(vec![0, 0], 1, 2)`.

```
(vec![<emotion>, <behavior>], <dim>, <new_value>)
```

The `dim` value tells which mental dimension that changes, and `new_value` tells which value it changes into.

One can also think of `DirectedContext` as a set of experiences:

- What would it be like to experience every possible mental state?
- How would it feel to change that mental state into something else?

Despite this simple property, the mathematics behind `DirectedContext` is quite complex. The “discrete” library makes it easy to construct it without making mistakes.

The `DimensionN` space in `Either<DirectedContext, DimensionN>` is a normal N-dimensional space that contains mental dimensions plus a physical direction. This is because when the agent chooses to move, the mental change is not needed. One can reduce the combinatorial complexity quite a bit by assuming a `DimensionN` instead of `DirectedContext`.

The `Either` space selects the first or the second child space. This is either encoded in the unique number for every change, or in enum variants `Select::Fst` or `Select::Snd`. See example code for more information (“examples/world_agent.rs”).

All this is put together in a tuple with the agent’s position:

```
(Either<DirectedContext, DimensionN>, DimensionN)
```

The tuple is also a discrete space that combines its child spaces by multiplying dimensions together.

This space has the following properties:

1. Assumes that the agent either changes mental state or makes a physical move
2. Contains one number representing every change uniquely (isolated decision)
3. Is generic for 1D, 2D, 3D and 4D worlds (higher dimensions are supported as well)
4. Is generic for any N-connected grid (physical moves that the agent can choose between)
5. Is directional
6. Has no redundancy (important for computing upper bounds of complexity)
7. Has an algebraic representation `(Mind + Move)World = MindWorld + MoveWorld`
8. Can be used to construct more complex discrete spaces (e.g. multi-agent environments)
9. Has no data (can be used both to describe perfect and imperfect environments)
10. Addresses all changes uniquely (can be used to log decisions of real world agents)

For example: To test a decision theory, one can enumerate all changes and see how the decision theory behaves. Various heuristics of safety measures can be pre-computed for all states and be used to compare the utilities assigned by the agent to isolated choices. This is more efficient than analyzing safety in context of all other possible choices.

Previously, I proved that safe AI boxing through Safety Interruption Oracle (SIO), is decidable if and only if the world is assumed to consist of finite number of states. The construction of a discrete space for agent's isolated decision is a direct follow-up from that proof.

In principle, it should be possible to derive a perfect SIO for every perfect environment based on some ADDS. While this might be computationally intractable for very large worlds, one might find it useful to derive SIO for small worlds and assume bounded search. This might improve short term safety. It does not guarantee that the agent will behave safely in the real world, but if the agent is not safe in a small and simplified world, it is almost guaranteed to be unsafe in the real world.

One can also use ADDS to extract training data for machine learning. After a decision theory is tested on small worlds, it is trivial to construct a larger world and pick a random number encoding a unique world, run the expensive decision algorithm and use it as label for the machine learning algorithm. Over time, the machine learning algorithm will approximate the decision theory. Machine learning can also be used to predict whether a world is helpful for learning by looking at the number directly.

For example, more objects and multiple agents can be added to the world, without loosing track of the measured complexity. The world can be analyzed directly, or stories might be fabricated using a constraint solver or automated theorem prover.

I previously wrote a paper "Intermediate Decision Theories" where I argue for the idea of customly designed decision theories to go safely from one decision theory to another. However, this approach rlies on making formal assumptions about decision theories.

Since the discrete space corresponds to a formal definition, it is possible to auto-generate a formal assumption from the type of a complex definition of a discrete space. Coupled with a library for common sense about this domain, this might be a powerful tool to reason about safe transitions between multiple decision theories designed for limited environments.