Frege Notes

In order to use many of the Java math functions in Frege, you need to import the corresponding library:

```java
import Prelude.Math
```

If you need to convert a Double to an Int, use `round`:

```java
round 7.6
8
```

To convert an Int to a Double, use `.double`:

```java
x = 5
y.double
5.0
```

Assignment #1 Notes

If you prefer, you can use the Haskell Platform and write Haskell code instead of Frege code. But note that later you will have to write Frege code for the game tournament. You can use as many helper functions as you want for any of the homework questions. You just need to put a file named A1.fr or A1.hs into your CS470/670 directory. No notes or memos are necessary. Please submit your code before Thursday’s class so that we can discuss the solutions in class.

A Trip to Grid-Space World

- Grid-space world is an extremely simple model of our own world.
- It is a three-dimensional space with a floor that is divided into cells by a two-dimensional grid.
- The cells can be empty or contain objects or agents.
- There can be walls between sets of cells.
- The agents are confined to the floor and can move from cell to cell.
- A robot in grid-space world can sense whether neighboring cells are empty or not.

Perception and Action

Organisms in the real world have to do two basic things in order to survive:

- They have to gather information about their environment (perception) and
- based on this information, they have to manipulate their environment (including themselves) in a way that is advantageous to them (action).

The action in turn may cause a change in the organism’s perception, which can lead to a different type of action. We call this the perception-action cycle.

Perception and Action

Complex organisms do not just perceive and act, but they also have an internal state that changes based on the success of previous perception-action cycles. This is the mechanism of learning.

We will first consider a very simple robot that lives in grid-space world and has no internal state.

The grid has no tight spaces, that is, spaces between objects and boundaries that are only one cell wide.
**Perception and Action**

The robot can move to a free adjacent cell in its column or row. Consequently, there are four possible actions that it can take:

- **north:** moves the robot one cell up
- **east:** moves the robot one cell to the right
- **south:** moves the robot one cell down
- **west:** moves the robot one cell to the left

---

**Immediate Perception-Action**

Now that we specified the robot’s capabilities, its environment, and its task, we need to give “life” to the robot. In other words, we have to specify a function that maps sensory inputs to movement actions so that the robot will carry out its task.

Since we do not want the robot to remember or learn anything, one such function would be sufficient. However, it is useful to decompose it in the following way (next slide):

---

**The Robot’s Perception**

For our robot, we define four different features $x_1, \ldots, x_4$ that are important to it. Each feature has value 1 if and only if at least one of the shaded cells is not free:

---

**The Robot’s Action**

To execute action, we define an ordered set of rules:

- If $x_1 = 0$ and $x_2 = 0$ and $x_3 = 0$ and $x_4 = 0$, move north
- If $x_1 = 1$ and $x_2 = 0$, move east
- If $x_2 = 1$ and $x_3 = 0$, move south
- If $x_3 = 1$ and $x_4 = 0$, move west
- If $x_4 = 1$ and $x_1 = 0$, move north
Production Systems

Using Boolean notation, the production system for our boundary-following robot looks like this:

\[ x_4x_1 \rightarrow \text{north} \]
\[ x_3x_4 \rightarrow \text{west} \]
\[ x_2x_3 \rightarrow \text{south} \]
\[ x_1x_2 \rightarrow \text{east} \]
\[ 1 \rightarrow \text{north} \]

Search in State Spaces

- Production systems are a standardized way to represent action functions.
- A production system consists of an ordered list of production rules (productions).
- Each rule is written in the form condition $\rightarrow$ action.
- A production system is therefore written like:
  \[ c_1 \rightarrow a_1 \]
  \[ c_2 \rightarrow a_2 \]
  \[ \ldots \]
  \[ c_m \rightarrow a_m \]
- The action of the first rule whose condition evaluates to 1 is executed.

Search in State Spaces

- Many problems in Artificial Intelligence can be mapped onto searches in particular state spaces.
- This concept is especially useful if the system (our "world") can be defined as having a finite number of states, including an initial state and one or more goal states.
- Optimally, there are a finite number of actions that we can take, and there are well-defined state transitions that only depend on our current state and current action.

Search in State Spaces

- To some extent, it is also possible to account for state changes that the algorithm itself does not initiate.
- For example, a chess playing program can consider its opponent's future moves.
- However, it is necessary that the set of such actions and their consequences are well-defined.
- While computers are able to play chess at a very high level, it is impossible these days to build a robot that, for instance, is capable of reliably carrying out everyday tasks such as going to a supermarket to buy groceries.
Search in State Spaces

Let us consider an easy task in a very simple world with our robot being the only actor in it:
- The world contains a floor and three toy blocks labeled A, B, and C.
- The robot can move a block (with no other block on top of it) onto the floor or on top of another block.
- These actions are modeled by instances of a schema, move(x, y).
- Instances of the schema are called operators.

Search in State Spaces

- The robot's task is to stack the toy blocks so that A is on top of B, B is on top of C, and C is on the floor.
- For us it is clear what steps have to be taken to solve the task.
- The robot has to use its world model to find a solution.
- Let us take a look at the effects that the robot's actions exert on its world.

Search in State Spaces

In order to solve the task efficiently, the robot should "look ahead", that is, simulate possible actions and their outcomes.
- Then, the robot can carry out a sequence of actions that, according to the robot's prediction, solves the problem.
- A useful structure for such a simulation of alternative sequences of action is a directed graph.
- Such a graph is called a state-space graph.

State-Space Graphs

- To solve a particular problem, the robot has to find a path in the graph from a start node (representing the initial state) to a goal node (representing a goal state).
- The resulting path indicates a sequence of actions that solves the problem.
- The sequence of operators along a path to a goal is called a plan.
- Searching for such a sequence is called planning.
- Predicting a sequence of world states from a sequence of actions is called projecting.