Introduction to the Policy Difference Engine - Part 1

Regan Meloche - Nov 2020

I am a developer with Code for Canada, a non-profit organization that works with government partners to deliver better digital services to the public. Every year, Code for Canada selects a cohort of fellows for a 10-month fellowship, lasting from November to August. For the duration of the fellowship, we will be working with **Employment and Social Development Canada (ESDC)**. This article is a brief intro to some of the more technical aspects of the project that have emerged during our introductory research, as well as the concept of 'Rules as Code', which may play an important role in the project.

The Current Landscape

ESDC is a government department that handles many public services such as employment insurance, pensions, and disability benefits. These services are bound by a wide variety of legislations, regulations, and policies, which we will generalize here under the term "rules," which often need to be updated to adjust to new realities. Examples may include changing an age requirement for eligibility of an allowance, changing the dollar amount of a benefit, re-defining what is meant by a "vehicle", etc. With technologies that directly impact the public growing in complexity, and the population experiencing increasingly diverse living situations, it's important to keep the rules up to date. One challenge that quickly emerges is effectively measuring the impact of changing a given rule. This is the challenge that ESDC and Code for Canada will be eagerly addressing. At the time of writing (November 2020), we are at the beginning of the fellowship, which will last until August 2021. We will be working closely with ESDC over the next nine months to make measurable progress on the challenge, and then smoothly hand off our work to ESDC at the end of the fellowship.

Refining the Project

The overarching challenge is to measure the impact of changing a rule. This problem statement is quite general, and can be broken down into different components, and we will focus on two of these. First of all, many rules will be inextricably linked to other rules. When a change is proposed to policy A, and policy A is linked to policy B, C, and D, those three policies must also be investigated to verify if the change will have any unintended side effects. This may further cascade to even more policies, exponentially increasing the amount of investigation. Much of this investigation is manual, so it can be difficult to keep track of all potential side effects.

The other issue is concretely measuring the impacts that a change may have. This includes the impact on the ability for ESDC to effectively deliver services, the impact on the cost of delivering the service, and — perhaps most importantly — the impact on the people that the rule is written for. Let's use an income tax rate as a straightforward example. How does changing an income tax rate by a certain amount affect a family of four with a monthly income of $5,000? How does it affect a single person with a monthly income of less than $1,000? There are many cases of households and individuals that must be considered for any proposed change. If we could run simulations on a model population that approximates the population of the area in question, then we've made progress towards evidence-based policy. The example of changing income tax is fairly straightforward to reason about, since it largely deals with numbers. This is not the case for all legislation. How would we measure the impact of changing some policy surrounding something more complex, such as changing the definition of a "vehicle" in some rules to include the latest and trendiest means of individual transport in a big city (e.g. motorized scooters)? On the surface, this does not appear to lend itself to a simulation in the same way that a number-heavy rule such as income tax might, and this will likely prove to be one of the more challenging aspects of the project.

Rules as Code

In order to run simulations efficiently, the rules must be programmed into a computer, which in turn requires that any ambiguities be removed. Many existing rules were not written with this in mind, so the translation from written rules to machine-consumable logic is a prerequisite for the project. This is the concept of Rules as Code (sometimes known as computational law).

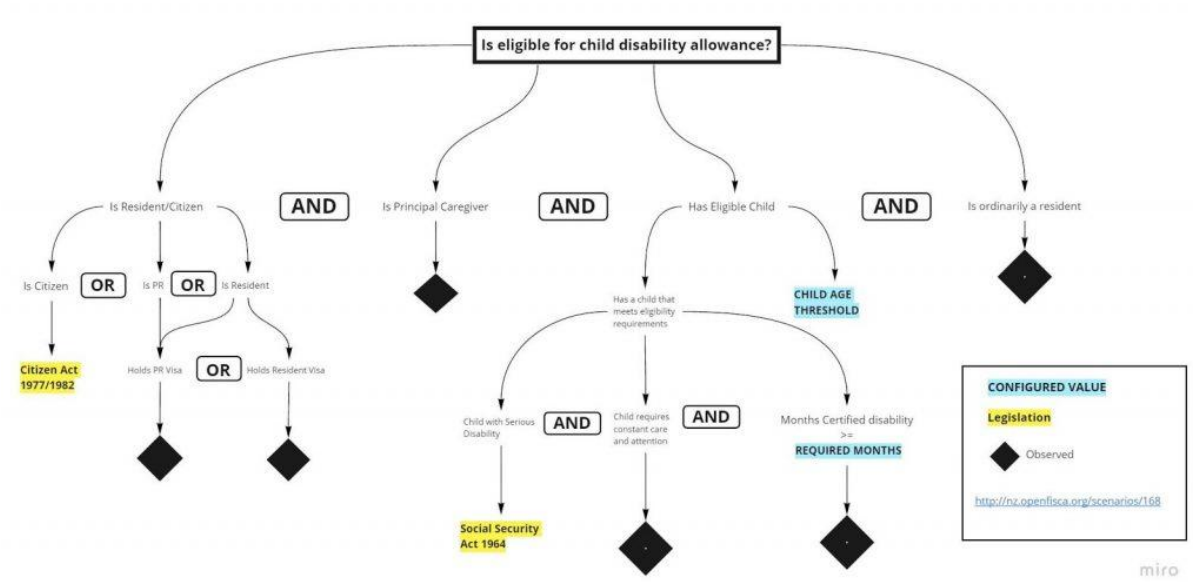
Example: New Zealand Child Disability Allowance

View the requirements for the New Zealand Child Disability Allowance.

Let's take an example that comes from a rule in New Zealand concerning the eligibility to receive a child disability allowance. The requirements can be broken down into the following components:

- You are a NZ citizen or resident
- You are the principal caregiver
- The child meets the eligibility criteria
- Both you and the child should ordinarily reside in New Zealand

Each of these components can in turn be broken down into even simpler requirements. For example, what does it mean for the child to meet the eligibility criteria? First of all, they must meet the age requirement of being under 18. On top of that, they must meet the definition of disability, which upon further digging, is outlined in the Social Security Act of 1964. The child must also require constant care and attention. This particular piece is not rooted in any sort of threshold number or existing act, but is simply an observed fact. Finally, they must also meet the requirement that they are expected to have the disability for a minimum of 12 months. This is another threshold value. If we flesh out all of these requirements, we can build a tree-like structure that breaks each rule down into its atomic components:



A flow diagram that shows how eligibility is determined for child disability allowance. This diagram demonstrates that residential status (resident/citizen), caregiver role/guardianship, as well as the child's needs and months certified with disability status can all impact eligibility for this particular benefit.

So we can see that ultimately these can break down into observed true/false facts (requires constant care and attention), facts based in prior legislation (definition of disability), or defined threshold values (age of requirement). This breakdown is starting to look much more machine-friendly and we could very easily capture these rules in computer code, thus giving us the concept of 'Rules as Code' (RaC). Again, note that this will not be the case with just any rule. Part of the challenge will be to identify rules that are suitable for this translation process.

Using the Transformed Rule

Once a given rule has been converted into RaC, we open up all the possibilities that come with any coded system. We can easily check for eligibility, we can make illustrative visualizations, and most applicable to us — we can run simulations on a model population. To this end, we can create a list of test cases or personas. This may be a collection of households with different configurations of ages, children, incomes, disability statuses, etc. Once these

test cases are in the system, we can represent a rule change by toggling values in the coded rules, and quantify the impact on the different personas.

Ethical and Legal Considerations

At this point it is worth pointing out an important ethical consideration. When modelling humans as data, we must always be careful about the data we are considering and the data we are not considering. A simulation on a model may show a net positive effect on the population in terms of overall cost, but if for example, we leave out key pieces of data (race, gender, age), and it turns out that a policy change actually negatively affects one group, then the simulation is largely incomplete. For this article, we will leave aside the discussion on whether *any* model may be better than no model at all, but it is an important consideration, and the discussion of any important omissions and oversights must remain part of the conversation.

One more issue with translating written rules to RaC is the preservation of legal intent. Again, most rules were not written with automation in mind, and automation demands clear rules without ambiguities. An ambiguous rule cannot be encoded into a rule without some bias on the translator's part, so an awareness of this is important. On a positive note, the act of translating may in fact bring to light these ambiguities and encourage an official clarification of the rule.

Next Steps

We've defined the high-level goal of the solution as measuring the impact of changing rules. We can break this down into two more concrete subgoals:

- Finding relevant connections between different rules
- Running valid simulations on proposed rule changes

With these defined, the next step will be to outline a general solution, which will

be part of what we call a Policy Difference Engine (PDE). This proposed tool (or set of tools) may be primarily used by someone responsible for proposing changes to existing legislation, regulations, or policies. In a future article, we hope to present this outline, and explore some technologies that may help serve the needs of the PDE.