Module 2: How to Design Data

CPSC 110

Peyton Seigo

2018-09-19

Module 2: How to Design Data

In summary, a **data definition** describes:

- how to form data of a new type
- how to represent information as data (information into data)
- how to interpret data as information (information from data)
- template for operating on data

Terminology

- cond is a **multi-armed conditional**: can have any number of cases, all at the same level
- **Problem domain**: contains information about a problem (e.g. light is red)
  - every program has a problem domain
- **Program**: uses data to represent information in the problem domain
- **Data definition**: describes how information is represented as data
  - *type comment* defines a new type name
  - body shows how to form data of that type
  - *interpretation* explains how to interpret data of this type as information, thereby establishing the information/data correspondence
  - *template* skeleton for one-argument functions that consume data of this type
    - demonstrates each possible case for the data type
- **Atomic information**: can't be taken apart into pieces AND still be meaningful in the problem domain
  - e.g. the city name "Vancouver" can be broken into V-a-n-c-o-u-v-e-r, but they are not meaningful to a city name (whereas a city itself is meaningful to a province, country, etc.)
  - *"the elapsed time since the start of the animation, the x coordinate of a car or the name of a cat"*
- **Orthogonality**: the HtDF (and HtDW) recipes work will all forms of data. the recipe is mostly orthogonal to the form of data.
- **Interval**: an interval of Numbers, Integers, or Naturals
- **Enumeration**: used when the information in problem domain consists of a fixed number of distinct items
  - *e.g. colours, letter grades, etc.*
  - Each "one-of" is a subclass
  - Do NOT collapse subclasses into a single `cond` case
  - Any data *can* be used, but strings should always be used
  - Interp. is often redundant, examples are nearly always redundant

- **Itemization**: is comprised of 2 or more subclasses, at least one of which is NOT a distinct data item
  - **Rule 1**: If a given subclass is the last subclass of its type, we can reduce the test to just the guard, i.e. `(number? n)`.
  - **Rule 2**: If all remaining subclasses are of the same type, then we can eliminate all of the guards.
  - **WARNING**: in a mixed data itemization template, the type specific predicates (i.e. <=) must be guarded against being called on the wrong type of data.
    * For example, `Integer[1, 10]` should test `(number? n)` if it's the only subclass with numbers, or `(and (number? n)(>= 1 n 10))` if there are multiple subclasses with numbers.
  - **Functions operating on itemizations**: should have at least as many tests as there as cases in the itemization. In the case of adjoining intervals, **it is critical to test the boundaries**.
  - If there are any discrete `string`s as the last subclasses, use an **`else`** instead of `(string =? n "...")`.
  - Always assume the user follows your data definition. Don't do more checks than you need to.

Syntax and structures

- **cond**: expression that has different behaviour based on any number of predicates
- `#;` comments out the entire expression or definition that follows the `#;`
- **`()` and `[]` are equivalent**: `[]` is used with `cond` cases by convention for clarity
- **`Integer[0, 33)`**: a range of `Integer`s from `0` (inclusive) to 33 (non-inclusive)

How to Design Data (HtDD) Recipe

Notes

- Anything to help understand what a data type represents belongs in the interpretation
  - e.g. for movie theatre seats, *"1 and 32 are aisle seats"*

Recipe

The first step of the recipe is to identify the inherent structure of the information.

Once that is done, a data definition consists of four or five elements:

1. A possible structure definition (not until compound data)
2. A type comment that defines a new type name and describes how to form data of that type.

3. An interpretation that describes the correspondence between information and data.
4. One or more examples of the data.
5. A template for a 1 argument function operating on data of this type.

```
;; Data definitions:

(@HtDD SomeType)
;; SomeType is Natural
;; interp. the airspeed velocity of an unladen swallow
(define ST1 24)
(define ST2 10)

(@dd-template-rules atomic-non-distinct)
(define (fn-for-some-type st)
  (... st))

;; Function definitions:

(@HtDF survive?)
(@signature SomeType -> Boolean)
;; produce true if given 24
(check-expect (survive? 24) true)
(check-expect (survive? 0) false)

; (define (survive?) false) ; stub

(@template SomeType) ; copied from data def. & modified in place
(define (survive? st)
  (= st 24))
```