CookieJar Abstract Data Type

H. Conrad Cunningham

27 April 2022

Contents

Professor of Computer and Information Science
University of Mississippi
214 Weir Hall
P.O. Box 1848
University, MS 38677
(662) 915-7396 (dept. office)

Browser Advisory: The HTML version of this textbook requires a browser that supports the display of MathML. A good choice as of April 2022 is a recent version of Firefox from Mozilla.

CookieJar ADT

Problem Description

The CookieJar problem corresponds (more or less) to exercises 4 and 5 on page 33 of Dale and Walker [1]. The exercise is to define and implement an abstract data type (ADT) for cookie jars, with operations for manipulating the jar and its contents.

As defined in the Dale and Walker textbook, the CookieJar ADT has the following operations:

`Create` to create a new empty cookie jar
`PutIn(CookieJar,CookieType)` to add one cookie of type `cookieType` into the cookieJar
`Eat(CookieJar,CookieType)` to remove one cookie of type `cookieType` from the cookieJar
`IsIn(CookieJar,CookieType)` to determine whether or not the cookieJar contains any cookie of type `cookieType`
`IsEmpty(CookieJar)` to determine whether or not the cookieJar is empty

Note: In a concrete implementation for this ADT, we use names for types, operations (i.e., functions or procedures), and variables that are more idiomatic for the language than the above names.

Semantics Specification

TODO: Give reference/link to the Data Abstraction document.

Axiomatic semantics

In (more or less) the notation of Dale and Walker [1], we can specify an *axiomatic semantics* specification for the CookieJar ADT as follows.

```
structure CookieJar (of CookieType)
    interface
        Create                      -> CookieJar
        IsEmpty(CookieJar)          -> Boolean
        PutIn(CookieJar,CookieType) -> CookieJar
        IsIn(CookieJar,CookieType)  -> Boolean
        Eat(CookieJar,CookieType)   -> CookieJar
    end

    axioms for all Cookie1 and Cookie2 in CookieType,
                   Jar in CookieJar, let

        IsEmpty(Create) = True
```

```
        IsEmpty(PutIn(Jar,Cookie1)) = False

        IsIn(Create,Cookie1) = False

        IsIn(PutIn(Jar,Cookie2),Cookie1) =
            IF Cookie1 = Cookie2
                THEN True
                ELSE IsIn(Jar,Cookie1)
            END IF

        Eat(Create,Cookie1) = Error

        Eat(PutIn(Jar,Cookie2),Cookie1) =
            IF Cookie1 = Cookie2
                THEN Jar
                ELSE PutIn(Eat(Jar,Cookie1),Cookie2)
            END IF

    end
end CookieJar
```

Constructive semantics

The source code files for an implementation of this ADT give a *constructive semantics* for that implementation. We give these semantics specifications as plain text in program comments in the source code files. The following document summarizes this notation:

- **Plain Text Specification Notation (PTSN)** HTML
  PDF

In the case of the CookieJar ADT, it is convenient to use a *bag* to describe the ADT's semantics.

Scala

Immutable Scala CookieJar ADTs

We define the interface to a family of *immutable* CookJar ADTs using a Scala trait. The ADT operations use a *method-chaining functional style*.

We express the constructive semantics in source code comments using the plain-text specification notation described above. We use a bag to represent the abstract model for the ADT's state, invariants to give the details of the data representation, and precondition and postcondition contracts to specify the behavior of the operations.

The trait is defined in the following Scala source file:

- `ICookieJar.scala`

Given this trait, we define three different implementations of the ADT, each of which uses a different representation for CookieJar's "bag".

- Using Scala List
- Using Scala HashMap
- Using Scala List of Tuples

In addition, we give a simple Blackbox test script for the above implementations in the source file:

- `ICookieJarTest.scala`

Mutable Scala CookieJar ADTs

We define the interface to a family of *mutable* CookJar ADTs using a Scala trait. The ADT operations use a *traditional object-oriented style*.

We express the constructive semantics in source code comments using the plain-text specification notation described above. We use a bag to represent the abstract model for the ADT's state, invariants to give the details of the data representation, and precondition and postcondition contracts to specify the behavior of the operations.

The trait is defined in the following Scala source file:

- `CookieJar.scala`

We define five different implementations, each of which uses a different representation for CookieJar's "bag".

- Using Scala List
- Using Scala HashMap
- Using Scala List of tuples
- Using Scala ArrayBuffer
- Using Scala array

In addition, we give a simple Blackbox test script for the above implementations in the source file:

- `CookieJarTest.scala`

Python

We designed a family of Python ADTs similar to the Scala mutable ADTs above.

Standard Mutable Python CookieJar ADTs

We define the interface to a family of *mutable* CookJar ADTs using a Python 3.7+ abstract class and optional type annotations. The ADT operations use a *traditional object-oriented style*.

We express the constructive semantics in source code comments using the plain-text specification notation described above. We use a bag to represent the abstract model for the ADT's state, invariants to give the details of the data representation, and precondition and postcondition contracts to specify the behavior of the operations.

The abstract class for the "standard" mutable CookieJar ADT is defined in the following Python 3.7+ source file:

- Cookie Jar ADT abstract interface (ABC)

As above, we define two different implementations, each of which uses a different representation for CookieJar's "bag". These extend the abstract class and are implemented using the `@dataclass` decorator. They are also annotated with the optional types.

- Cookie Jar ADT List implementation
- Cookie Jar Dictionary implementation

In addition, we give a simple Blackbox test script for the above implementations in the source file:

- CookieJar ADT Test module

Extended Mutable Python CookieJar ADTs

We define the interface to a family of (an extended) *mutable* CookJar ADTs using a Python 3.7+ abstract class with optional type annotations. The ADT operations use a *traditional object-oriented style*.

We express the constructive semantics in source code comments using the plain-text specification notation described above. We use a bag to represent the abstract model for the ADT's state, invariants to give the details of the data representation, and precondition and postcondition contracts to specify the behavior of the operations.

These ADTs extend the ADTs above by adding two other simple accessor methods. We use these in refining the semantics of other operations.

The abstract class for the "extended" mutable CookieJar ADT is defined in the following Python 3.7+ source file:

- CookieJar ADT abstract interface (ABC)

As with the "standard" ADTs, we define two different implementations, each of which uses a different representation for CookieJar's "bag". These extend the

abstract class and are implemented using the `@dataclass` decorator. They are also annotated with the optional types.

- CookieJar ADT List implementation
- CookieJar Dictionary implementation

In addition, we give a simple Blackbox test script for the above implementations in the source file:

- CookieJar ADT Test module

Ruby

The first implementation I did of this ADT was a simple object-oriented Ruby program in 2006. It implements a mutable ADT using a Ruby dynamic array as its representation.

- Mutable Ruby CookieJar Array implementation
- Answers to exercises

Note: These old programs and solutions follow the Dale and Walker book more closely and may not use the same specification notation as the other solutions.

Acknowledgements

In my graduate Special Topics class on Ruby and Software Development (Engr 692) in Fall 2006, I assigned exercises 4 and 5 on page 33 of the Dale and Walker [1] to my students. In addition to the specification, they were to develop a simple implementation in Ruby. The Ruby implementation is my solution to that exercise.

I developed several solutions in other languages over the years when I have used the program as an example and/or an exercise.

I retired from the full-time faculty in May 2019. As one of my post-retirement projects, I am continuing work on possible textbooks based on the course materials I had developed during my three decades as a faculty member. In January 2022, I began refining the existing content, integrating separately developed materials together, reformatting the documents, constructing a unified bibliography (e.g., using citeproc), and improving my build workflow and use of Pandoc. I adapted this index page from a portion of my Spring 2019 CSci 555 course notes.

I maintain this chapter as text in Pandoc's dialect of Markdown using embedded LaTeX markup for the mathematical formulas and then translate the document to HTML, PDF, and other forms as needed.

References

[1] Nell Dale and Henry M. Walker. 1996. *Abstract data types: Specifications, implementations, and applications.* D. C. Heath, Lexington, Massachusetts, USA.