Abstraction and Specification

Prof. Clarkson
Fall 2018

Today's music: "A Fifth of Beethoven" by Walter Murphy

Please try to sit with your team (and every lecture henceforth).
Review

Previously in 3110:
• Language features for modularity
• Some familiar data structures

Today:
• Abstraction
• Specification of functions
Attendance question

How do you learn libraries?

A. I search StackOverflow for examples then tweak them
B. I read a tutorial on someone's blog
C. I read the library's official documentation
D. I read the library's source code
What if you had to read the implementation?

```ml
let rec sort n l =
  match n, l with
  | 2, x1 :: x2 :: _ ->
    if cmp x1 x2 <= 0 then [x1; x2] else [x2; x1]

  | 3, x1 :: x2 :: x3 :: _ ->
    if cmp x1 x2 <= 0 then begin
      if cmp x2 x3 <= 0 then [x1; x2; x3]
      else if cmp x1 x3 <= 0 then [x1; x3; x2]
      else [x3; x1; x2]
    end else begin
      if cmp x1 x3 <= 0 then [x2; x1; x3]
      else if cmp x2 x3 <= 0 then [x2; x3; x1]
      else [x3; x2; x1]
    end

  | n, _ ->
    let n1 = n asr 1 in
    let n2 = n - n1 in
    let l2 = chop n1 l in
    let s1 = rev_sort n1 l in
    let s2 = rev_sort n2 l2 in
    rev_merge_rev s1 s2 []
```

...
Abstraction

(verb)
Forgetting information, so that different things can be treated as the same

(noun)
Artifacts that result from that process
Specification

(noun)
Intended behavior of abstraction

(verb)
The act of creating such an artifact
Example specification

val sort :
('a -> 'a -> int) -> 'a list -> 'a list

Sort a list in increasing order according to a comparison function. The comparison function must return 0 if its arguments compare as equal, a positive integer if the first is greater, and a negative integer if the first is smaller (see Array.sort for a complete specification). For example, compare is a suitable comparison function. The resulting list is sorted in increasing order. List.sort is guaranteed to run in constant heap space (in addition to the size of the result list) and logarithmic stack space.

Discuss: Identify preconditions and postconditions.
Example specification

val sort : ('a -> 'a -> int) -> 'a list -> 'a list

Sort a list in increasing order according to a comparison function. The comparison function must return 0 if its arguments compare as equal, a positive integer if the first is greater, and a negative integer if the first is smaller (see `Array.sort` for a complete specification). For example, `compare` is a suitable comparison function. The resulting list is sorted in increasing order. `List.sort` is guaranteed to run in constant heap space (in addition to the size of the result list) and logarithmic stack space.
Example specification

```ocaml
val sort : ('a -> 'a -> int) -> 'a list -> 'a list
```

Sort a list in increasing order according to a comparison function. The comparison function must return 0 if its arguments compare as equal, a positive integer if the first is greater, and a negative integer if the first is smaller (see `Array.sort` for a complete specification). For example, `compare` is a suitable comparison function. The resulting list is sorted in increasing order. `List.sort` is guaranteed to run in constant heap space (in addition to the size of the result list) and logarithmic stack space.
Example specification

val sort : ('a -> 'a -> int) -> 'a list -> 'a list

Sort a list in increasing order according to a comparison function. The comparison function must return 0 if its arguments compare as equal, a positive integer if the first is greater, and a negative integer if the first is smaller (see Array.sort for a complete specification). For example, compare is a suitable comparison function. The resulting list is sorted in increasing order. List.sort is guaranteed to run in constant heap space (in addition to the size of the result list) and logarithmic stack space.

Postcondition
Example specification

val sort : ('a -> 'a -> int) -> 'a list -> 'a list

Sort a list in increasing order according to a comparison function. The comparison function must return 0 if its arguments compare as equal, a positive integer if the first is greater, and a negative integer if the first is smaller (see \texttt{Array.sort} for a complete specification). For example, \texttt{compare} is a suitable comparison function. The resulting list is sorted in increasing order. \texttt{List.sort} is guaranteed to run in constant heap space (in addition to the size of the result list) and logarithmic stack space.
Specifications are contracts
Benefits

- **Locality:** understand abstraction without needing to read implementation

- **Modifiability:** change implementation without breaking client code

- **Accountability:** clarify who is to blame
Audience of specification

• **Clients**
  - What they must guarantee (preconditions)
  - What they can assume (postconditions)

• **Implementers**
  - What they can assume (preconditions)
  - What they must guarantee (postconditions)
Audience of specification

• Clients
  – What they must guarantee (preconditions)
  – What they can assume (postconditions)

• Implementers
  – What they can assume (preconditions)
  – What they must guarantee (postconditions)
Audience of specification

• Clients
  – What they must guarantee (preconditions)
  – What they can assume (postconditions)

• Implementers
  – What they can assume (preconditions)
  – What they must guarantee (postconditions)
Satisfaction

An implementation satisfies a specification if it provides the described behavior.

Many implementations can satisfy the same specification:
- **Client** has to assume it could be any of them.
- **Implementer** gets to pick one.
SPECIFYING FUNCTIONS
A template for spec. comments

(** [f x] is ...
  Example: ...
  Requires: ...
  Raises: ... *)

val f : t1 ... -> u

Based on Abstraction and Specification in Program Development
(Now Program Development in Java: Abstraction, Specification, and Object-Oriented Design)
By Barbara Liskov and John Guttag
Barbara Liskov

b. 1939

Turing Award Winner 2008

For contributions to practical and theoretical foundations of programming language and system design, especially related to data abstraction, fault tolerance, and distributed computing.
Requires clause

(** [hd lst] is the head of [lst].
    Requires: [lst] is non-empty. *)

val hd : 'a list -> 'a

Precondition: blame client if input is bad
Requires clause

(** [hd lst] is the head of [lst].
Requires: [lst] is non-empty. *)

`val hd : 'a list -> 'a`

Precondition: blame client

Types are part of the source code not the comment.
Returns clause

(** [sort lst] contains the same elements as [lst], but sorted in ascending order. *)

val sort : int list -> int list

Postcondition: blame implementer if output is bad
(unless client violated a precondition)
Example clause

(** Examples:
   - [sort [1;3;2;3]] is [[1;2;3;3]].
   - [sort []] is [[]]. *)

val sort : int list -> int list

Super helpful to clarify spec for humans.
**Raises clause**

(** [hd lst] is the head of [lst].
Requires: [lst] is non-empty.
Raises: [Failure "hd"] if [lst] is empty. *)

**val hd : 'a list -> 'a**

Also a postcondition: behavior implementer must provide
**Total function:**
Well-defined behavior for all inputs. 
*No requires/raises clause needed.*

**Partial function:**
Some inputs lead to unspecified behavior. 
*Requires/raises clause needed.*
Assert the precondition?

Discuss: should you? always? in limited circumstances?

A. Always    B. Sometimes    C. Never
WORKING WITH SPECS
TL;DR: It's hard

Writing good specs is hard:
• the language and compiler do not demand it
• if you're coding only for yourself, neither do you

Reading specs is also hard:
• requires close attention to detail
When to write specifications

• During design:
  – as soon as a design decision is made, document it in a specification
  – posing and answering questions about behavior clarifies what to implement

• During implementation:
  – update specification during code revisions
  – a specification becomes obsolete only when the abstraction becomes obsolete
What if spec is ambiguous?

Ambiguity is a fact of life.
Do the most reasonable thing you can.
   Probably not 🔥

Who wrote it?
• You: improve it
• Client: seek clarification; but if you make 500 requests they probably won't hire you again
Upcoming events

• No A3 GIST tonight

This is hard.

THIS IS 3110