Specifying algorithms

Stefan D. Bruda

CS 310, Winter 2025

SPECIFICATIONS

- Specifications describe transformations from input values to output values
  - A specification consists of the following parts: input, output and environment in which the specification/program will work (constraints on variables)
- The formalization of a specification consists of the following:
  - Declarative interface = static properties of the identifiers, including constants
  - Pre-condition: assertion on input values that the program will be given
  - Post-condition: assertion on output values, possibly in relation to input values
- We use logical formulae (assertions) which will not be evaluated during execution, but will hold true if evaluated

```
ASSERT(P)
S
ASSERT(Q)
```

or for short

```
P {S} Q
```

- A specification is a contract: if the pre-conditions hold then the post-conditions must hold, otherwise all the bets are off (the specification is vacuously true)
- Total correctness: if `P` holds when `S` starts then `S` terminates in a state that satisfies `Q`
- Partial correctness: if `P` holds when `S` starts then `S` will not terminate normally in a state that does not satisfy `Q`

Declarative Interface

- Running example: write a code fragment that tests for the presence of a given value `x` in a given array `A[]`
  - The range of subscripts for `A[]`: `0..n`, for a given `n<=max`
  - The type of `x` (and values in `A[]`): some type that can be compared with `==`
  - How will the result be given: by setting the boolean variable `present`
  - Can we assume that the array is sorted: no
  - Can we have duplicates in `A[]`: yes, but duplicates do not matter
  - What happens if `A[]` is empty: set the variable `present` to `false` (not an error)
  - What cannot be modified: `A[0..n]` and `x`
- Declarative interface:

```
const int max;     /* maximum number of entries */
typename Entry;    /* type of entries, use == to compare */
const int n;       /* actual number of entries */
const Entry x;     /* search target */
Entry A[max];      /* A[0..n] are the entries to search */
bool present;      /* search result */
```


Assertions

- Assertions are boolean formulae
- Essentially C expressions of type `bool` with some extensions and shortcuts:
  - Logical implication => and iff <=>
    - If `P=>Q` is true then `P` is said to be stronger than `Q` (and `Q` weaker than `P`)
  - Existential and universal quantifiers: `ForAll(`I`).`P and `Exists(`I`).`P (alternative notation: $\forall I.P$ and $\exists I.P$)
    - The identifier I is said to be bound
  - Natural abbreviations including:
    - `a<=i<b` instead of `a<=i && i<b`
    - `ForAll (i=a; i<b)` P instead of `ForAll (int i) a<=i<b =>` P
    - `Exists (i=a; i<b)` P instead of `Exists (int i) a<=i<b =>` P
    - `x in A[a:b-1]` instead of `Exists (i=a; i<b) x == A[i]`
    - `x < A[a:b-1]` instead of `ForAll (i=a; i<b) x < A[i]`, etc.
  - Assertions are provided as comments (they are not evaluated, but is expected that they would be true if evaluated):

```
#define ASSERT(P)
```

Pre- and Post-Conditions

- A post-condition is expected to be true immediately after the execution of the preceding code
  - Example: `present <=> Exists (k=0; k<n) A[k] == x`
- A pre-condition must be true immediately before the code is executed
  - Example: `0 <= n <= max`
- Some times we need to specify a condition between the initial and the final content of a variable
  - If so we conceptually define a copy of the variable and use that:

    ```
    Entry A[max];       /* A[0..n] are the entries to search */
    /* const Entry A0[max]; */
    ```

  - Strengthen the pre-condition to `0 <= n <= max && A == A0`
  - A possible post-condition is then

    ```
    (present <=> Exists (k=0; k<n) A[k] == x) &&
    (ForAll (i=0; i<n) A[i] == A0[i])
    ```


Array Search

```
const int max;     /* maximum number of entries */
typename Entry;    /* type of entries, use == to compare */
const int n;       /* actual number of entries */
const Entry x;     /* search target */
Entry A[max];      /* A[0..n] are the entries to search */
bool present;      /* search result */

int main () {
    ASSERT(0 <= n <= max && A == A0)

    int i;
    present = false;
    for (i=0; i<n; i++) {
        if (A[i] == x) present = true;
    }

    ASSERT((present <=> Exists (k=0; k<n) A[k] == x) &&
           (ForAll (i=0; i<n) A[i] == A0[i]))
}
```