Event-B: Introduction and First Steps[1]

Manuel Carro
manuel.carro@upm.es

Universidad Politécnica de Madrid &
IMDEA Software Institute

[1] Many slides borrowed from J. R. Abrial

Conventions

I will sometimes use boxes with different meanings.

- Quiz to do together during the lecture.

  Q: What happens in this case?

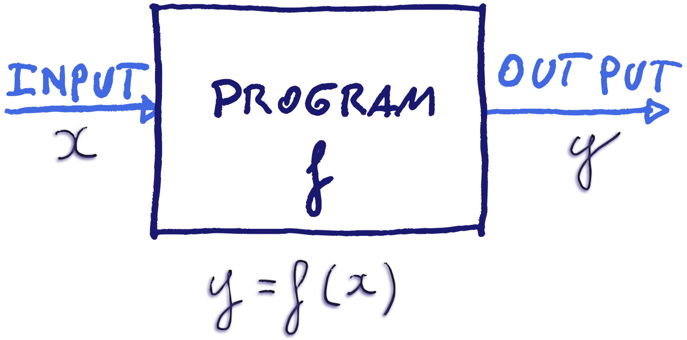
- Material / solutions that I want to develop during the lecture.

  Something to complete here

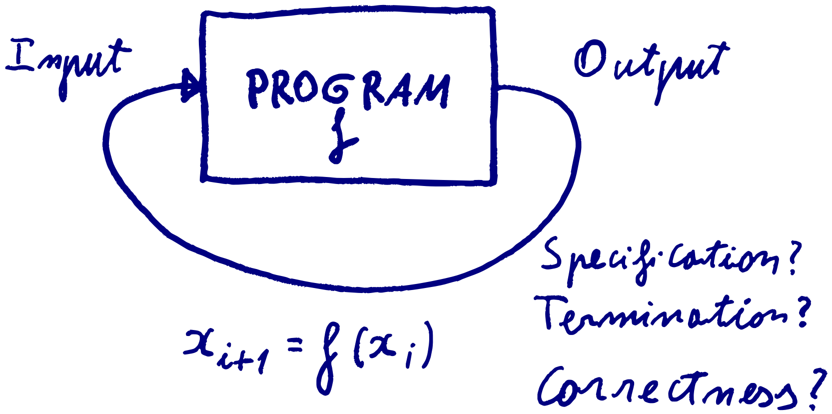
  aaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaa

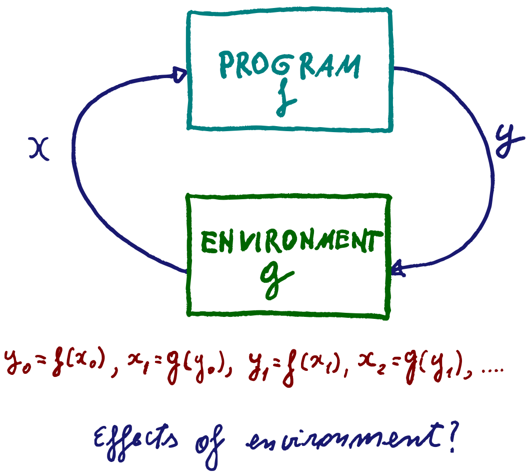
Event B

An industry-oriented method, language, and set of supporting tools to describe systems of interacting, reactive software, hardware components, and their environment, and to reason about them.



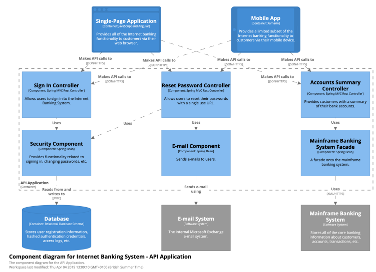
Specification: remember sorting program.





Industrial systems: usual characteristics

- Functionality often not too complex.
  - Algorithms / data structures relatively simple.
  - Underlying maths of reasonable complexity.
- Requirements document usually poor.
- Reactive and concurrent by nature.
  - But often coarse: protecting (large) critical regions often enough.



- Many special cases.
- Communication with hardware / environment involved.
- Many details (≈ properties to ensure) to be taken into account.
- Large (in terms of LOCs).

Producing correct (software) systems hard — but not necessarily from a theoretical point of view.

Typical approaches and problems

Usual approach

- Choose a platform.
- Write software specifications (which often neglect or under-represent the environment).
- Design by cutting in small pieces with well-defined communication.
- Code and test / verify units.
- Integrate and test.

Typical approaches and problems

Usual approach

- Choose a platform.
- Write software specifications (which often neglect or under-represent the environment).
- Design by cutting in small pieces with well-defined communication.
- Code and test / verify units.
- Integrate and test.

Pitfalls

- Often too many details / interactions / properties to take into account.
- Cutting in pieces: poor job in taming complexity.
  - Small pieces: easy to prove them right.
  - Additional relationships created!
  - Overall complexity not reduced.
- Modeling environment?
  E.g., we expect a car driver to stop at a red light.
- Result: system as a whole seldom verified.

The Event B approach

Complexity: Model Refinement

- System built incrementally, monotonically.
  - Take into account subset of requirements at each step.
  - Build model of a *partial* system.
  - Prove its correctness.
- **Add** requirements to the model, ensure correctness:
  - The requirements correctly captured by the new model.
  - New model preserves properties of previous model.

Details: Tool Support

- Tool to edit Event B models (Rodin).
- Generates *proof obligations*: theorems to be proved to ensure correctness.
- Interfaced with (interactive) theorem provers.
- Extensible.

Basic ideas

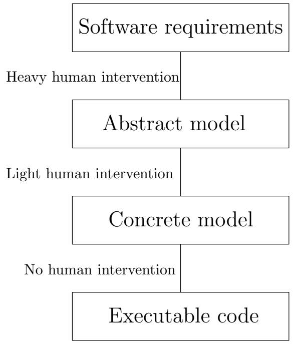
- Model: formal description of a discrete system.
  - Formal: sound mechanism to decide whether some properties hold
  - Discrete: can be represented as a transition system

Basic ideas

- Model: formal description of a discrete system.
    - Formal: sound mechanism to decide whether some properties hold
    - Discrete: can be represented as a transition system
- Formalization contains models of:
    - The future software components
    - The future equipments surrounding these components

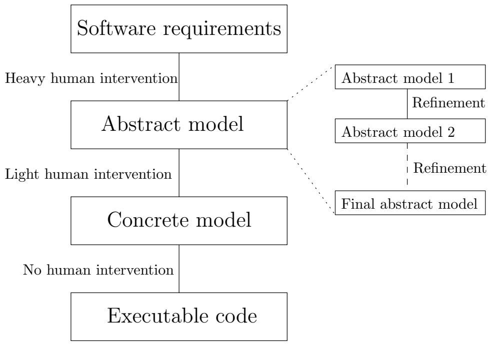
Refinement

- Refinement allows us to build a model gradually.
- Ordered sequence of more precise partial models.
- Each model is a refinement of the one preceding it.
- Each model is proven:
    - Correct.
    - Respecting the boundaries of the previous one.



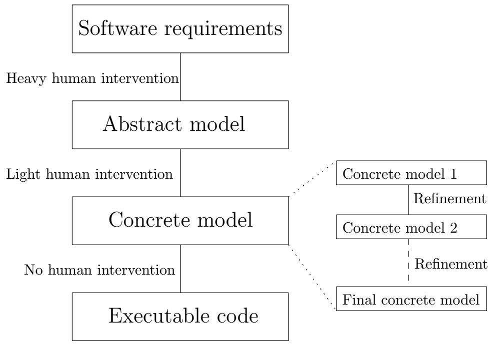
Refinement

- Refinement allows us to build a model gradually.
- Ordered sequence of more precise partial models.
- Each model is a refinement of the one preceding it.
- Each model is proven:
    - Correct.
    - Respecting the boundaries of the previous one.



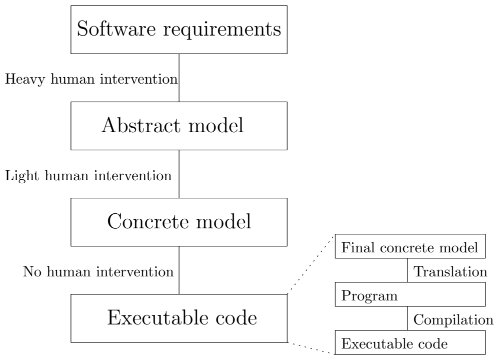
Refinement

- Refinement allows us to build a model gradually.
- Ordered sequence of more precise partial models.
- Each model is a refinement of the one preceding it.
- Each model is proven:
    - Correct.
    - Respecting the boundaries of the previous one.



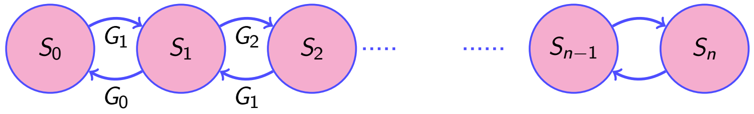
Refinement

- Refinement allows us to build a model gradually.
- Ordered sequence of more precise partial models.
- Each model is a refinement of the one preceding it.
- Each model is proven:
  - Correct.
  - Respecting the boundaries of the previous one.



Models and states

A discrete model is made of states



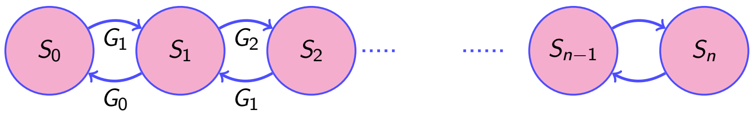
- States are represented by constants, variables, and their relationships

$$S_i = \langle c_1, \ldots, c_n, v_1, \ldots, v_m \rangle$$

- Relationships among constants and variables written using set-theoretic expressions

Models and states

A discrete model is made of states



- States are represented by constants, variables, and their relationships

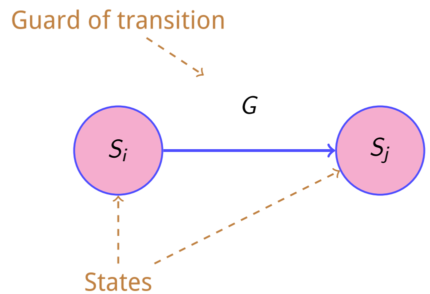
$$S_i = \langle c_1, \ldots, c_n, v_1, \ldots, v_m \rangle$$

- Relationships among constants and variables written using set-theoretic expressions

What is its relationship with a regular program?

States and transitions

- Transitions between states: triggered by events
- Events: guards and actions
  - Guard (G_i) denote enabling conditions of events
  - Actions denote how states are modified by events
- Guards and actions written with set-theoretic expressions (e.g., first-order, classical logic).
- Event B based on set theory.



Examples:

$S_i \equiv x = 0 \land y = 7$

$S_i \equiv x, y \in \mathbb{N} \land x < 4 \land y < 5 \land x + y < 7$

Write extensional definition for the latter

A simple example – informal introduction!

Search for element k in array f of length n, assuming k is in f.

Constants / Axioms

$$\text{CONST } n \in \mathbb{N}$$
$$\text{CONST } f \in 1..n \longrightarrow \mathbb{N}$$
$$\text{CONST } k \in \text{ran}(f)$$

Variables / Invariants

$$\text{VARIABLE } i \in 1..n$$

```
Event Search
  when
    i < n ∧ f(i) ≠ k
  then
    i := i + 1
  end
```

```
Event Found
  when
    f(i) = k
  then
    skip
  end
```

(initialization of i not shown for brevity)

Events

```
Event EventName
  when
    guard:  G(v, c)
  then
    action:  v := E(v, c)
  end
```

- Executing an event (normally) changes the system state.
- An event can[2] fire when its guard evaluates to true.
- G(v, c) predicate that enables EventName
- v := E(v, c) is a state transformer.

[2] Not "must"!