EECS3342: Specification and Refinement
Bank System to illustrate parameters

JSO, EECS3342

EECS, York University

January 2, 2017

A Structured Requirement Documents

A small bank

A bank provides its customers with cash accounts. Customers can deposit and withdraw cash. Balances may not be less than a global credit limit. Wihdrawals canot be more than available in the cash draw.

REQ 1	The bank maintains a set of accounts.

REQ 2	Customers can deposit and withdraw cash.

- Two separate texts in the same document:
  - explanatory text: the why
  - reference text: the what
- Embedding the reference text within the explanation text
- Convert into atomic statements

Reference Text vs. Explanatory Text

Reference Text

- is inside the boxes,
- forms a contract,
- is concise and self-contained,
- describes the "what"

Explanatory Text

- is outside the boxes,
- is unofficial,
- improves readable,
- explains the "why"

Labels for Reference Text

- Reference text is denoted with an unique label.
- Each label forms by some short text and a number.
- Taxonomy (incomplete):
  - Requirements: REQ
  - Requirements: ENV
  - Assumption: ASM
  - Functional: FUN
  - Environment: EVN
  - Safety: SAF
  - Security: SEC

Labels for Reference Text

- Reference text is denoted with an unique label.
- Each label forms by some short text and a number.
- Taxonomy (incomplete):
  - Requirements: REQ
  - Requirements: ENV
  - Assumption: ASM
  - Functional: FUN
  - Environment: EVN
  - Safety: SAF
  - Security: SEC

Taxonomy

REQ

- aims/goals of the system

REQ 3	All account balances shall exceed a global credit limit.

ENV (environmental constraints/assumptions)

- properties of the system that can be assumed.

ENV 4	Bank tellers do not register withdrawals requiring more than available in the cash draw

env

Requirements Document

Common Problems

- vague and unspecific statements
- over-specification
- irrelevant assumptions
- missing requirements
- missing assumptions

General Guidelines

- Explain ambiguous terms
- Be brief, dry, to the point
- Keep your vocabulary small

Event-B Modelling Method

- A model: discrete transition systems:
  - (infinite) states
  - transitions
- Consists of contexts and machines.

$$\boxed{M} \xrightarrow{\text{sees}} \boxed{C}$$

Figure: The sees relationship

Contexts

- Carrier sets, e.g *ACCOUNT* which is the set of all bank accounts
- Constants, e.g. the credit limit c.
- Axioms / theorems
- Consistency: theorems is derivable from axioms

Describing REQ1: bank accounts; REQ3: credit limit c

carrier sets: $ACCOUNT$

constants: c

axioms:
axm0_1 : $c \in \mathbb{N}1$
thm0_2 : $c > 0$

Machines

- Variables, e.g. b as a map from each account in the system to their corresponding balances
- Invariants / theorems
- Events (next slides)

Variables: cash drawer d and account balances b

variables: d, b

invariants:
inv0_1 : $d \in \mathbb{N}$
inv0_2 : $b \in ACCOUNT \nrightarrow \mathbb{Z}$

```
init
  begin
    d :∈ ℕ
    b :∈ ACCOUNT ⇸ ℕ
  end
```

Account balance

variables: b

invariants:
inv0_2 : $b \in ACCOUNT \nrightarrow \mathbb{Z}$

Example

E.g., $b = \{a1 \mapsto 0, a2 \mapsto 25\}$
$b(a2) =$ balance of account $a2$ is \$25

Events

name
any *parameters* **where**
guard
then
action
end

- An event is enabled: There exists some value for the parameters such that the guard holds.

deposit and withdraw events

deposit
any a, v **where**
$a \in dom(b)$
$v \in \mathbb{N}1$
then
$d := d + v$
$b(a) := b(a) + v$
end

withdraw
any a, v **where**
$a \in dom(b)$
$v \in \mathbb{N}1$
then
$d := d - v$
$b(a) := b(a) - v$
end

Operational Interpretation

- When all events are not enabled, the system stops.
- When some events are enabled
  - One of them is chosen (together with the correct parameter)
  - Its action is carried out.

Consistency 1

1. Theorems.
2. Invariants (taken from atomic requirements)

$$\begin{array}{ll} \textbf{inv0\_1}: & d \in \mathbb{N} \\ \textbf{inv0\_2}: & \forall a. a \in dom(b) \Rightarrow b(a) \geq -c \end{array}$$

- Establishment

init
  begin
    $d :\in \mathbb{N}$
    $b :\in ACCOUNT \nrightarrow \mathbb{N}$
  end

inv0_1: Required to prove that: $d' \in \mathbb{N}$

- Maintenance

withdraw
  any a, v **where**
    $a \in dom(b)$
    $v \in \mathbb{N}1$
  then
    $d := d - v$
    $b(a) := b(a) - v$
  end

inv0_1: Required to prove that $d' \in \mathbb{N}$ where $d' = d - v$. Proof will not work. Why?

New atomic requirement

REQ 5	Withdrawals cannot be for more cash than is in the cash draw.

Describing the cash drawer d

variables: d, b

invariants:
inv0_1 : $d \in NAT$

withdraw
any a, v **where**
$a \in dom(b)$
$b(a) - v \geq -c$
$v \leq d$
then
$b(a) := b(a) - v$
$d := d - v$
end

Consistency 2

1. Theorems.
2. Invariants (taken from atomic requirements)

$$\begin{array}{ll} \textbf{inv0\_1}: & d \in \mathbb{N} \\ \textbf{inv0\_2}: & \forall a . a \in dom(b) \Rightarrow b(a) \geq -c \end{array}$$

- Establishment

```
init
  begin
    d :∈ ℕ
    b :∈ ACCOUNT ⇸ ℕ
  end
```

required to prove that:
$b\prime(a) \geq -c$

- Maintenance

```
withdraw
  any   a, v   where
    a ∈ dom(b)
    v ∈ ℕ1
  then
    d := d − v
    b(a) := b(a) − v
  end
```

required to prove that under the invariant and guard, $b'(a) - v \geq -c$ where b' is the new balance after initialization.

Preserving the invariants

Required to prove that under the invariant and guard, $b'(a) - v \geq -c$

withdraw
any a, v **where**
$a \in dom(b)$
$v \in \mathbb{N}1$
$b(a) - v \geq -c$
then
$b(a) := b(a) - v$
end

Refinement (1/2)

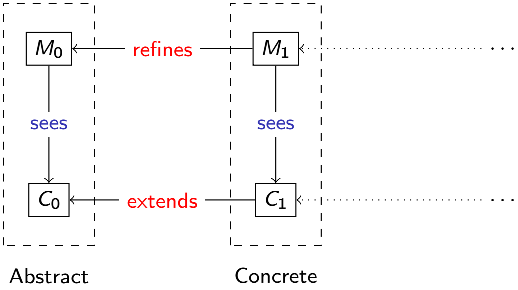

Figure: Refinement

- Contexts can be extended.
- Machines can be refined.
- The abstract model can simulate the concrete model

Refinement (2/2)

- Intuition: refinement reduces behaviours of the model.

Example. Limited deposit

(abs_)deposit
any a, v **where**
$a \in dom(b) \wedge v \in \mathbb{N}$
then
$b(a) := b(a) + v$
end

(cnc_)deposit
any a, v **where**
$a \in dom(b) \wedge v \in \mathbb{N}$
$v \leq L - b(a)$
then
$b(a) := b(a) + v$
end

- Refinement consistency needs to be proved.

Incremental Formal Development

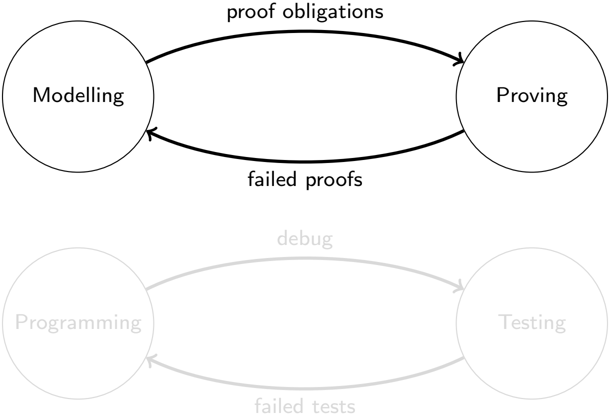

Figure: Incremental development

Incremental Formal Development

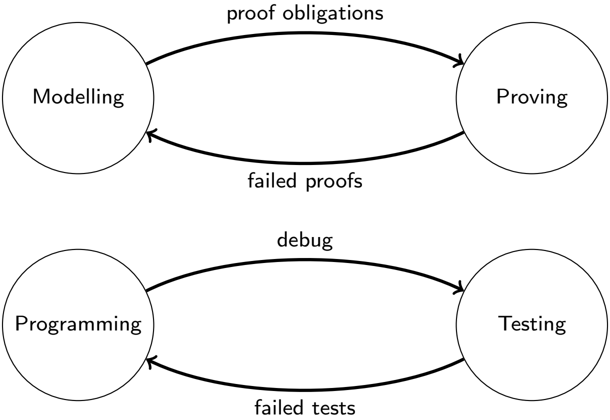

Figure: Incremental development

Summary

- A model is a discrete transition system,
- consists of contexts and machines.
- Consistency:
  - theorems (of contexts and machines),
  - invariants,
  - refinement.

What you need for formal reasoning 1

- Insights about modelling and formal reasoning.
- Systems can be correct-by-construction.
- Modelling can be made practical.

What you need for formal reasoning 2

- Modelling (versus programming)
- Abstraction and refinement
- Some mathematical techniques used for reasoning.
- The practice of incremental development: modelling and proving.
- The usage of some proving tools.

Outline

Theorem Proving: Model Construction

- It concentrates on the construction of models by
  successive refinements
- The properties to be proved are parts of the models:
  invariants and refinement

Why Using Formal Methods with Proofs?

- When there is nothing better available.
- When the risk is too high (e.g. in embedded systems).
- When people have already suffered enough.
- When people question their development process.
- Decision of using formal methods is always strategic.

Which Formal Method with Proofs?

- This is a difficult question.
- Today many formal methods vendors.
- "Formal method" has become a meaningless buzz word.
- "Formal" alone does not mean anything.

Claimed Difficulties in Applying Formal Methods

- They are only used for safety-critical systems.
- You have to be a mathematician.
  - Formalism is hard to master.
  - People will not be able to do formal proofs.
- Formal models are hard to understand
  - Not visual enough (no boxes, arrows, etc.).
- Formal methods lack tools.
- References:
  - Seven Myths of Formal Methods. (1990, by Hall)
    http://www.seas.upenn.edu/~lee/09cis480/papers/7myths1990.pdf
  - Seven More Myths of Formal Methods. (1995, by Bowen&Hinchey)
    http://lml.ls.fi.upm.es/rsd/Papers/seven_more_myths.pdf

Genuine Difficulties in Applying Formal Methods

- Model building is an elaborate activity.
- You have to think a lot before final coding.
- Incorporation in development process.
- Reasoning by means of proof is necessary.
- Making proofs a design criterion.
- Poor quality of requirement documents.