Discrete System Models

Jean-Raymond Abrial

2003

This Lecture: Discrete System Models

- Introduction to formal methods and B: EASY :-)

- Presentation of Event-B: A LITTLE LESS EASY :-|

- Application overview: EASY AGAIN :-)

Why Using Formal Methods ?

- When there is nothing better to do.

- When the risk is too high.

- When people have already suffered enough.

- When people question their development process.

- Decision of using FM is always strategic.

Which Formal Method ?

- This is a difficult question.

- Today many formal methods vendors.

- FM has become a meaningless buzz word.

- "Formal" alone does not mean anything.

Questions to be asked to FM Vendors

- Is there a theory behind your FM ?

- What kind of language is your FM using ?

- Do there exists any refinement mechanism ?

- Do you prove anything when using your FM ?

- Have you got an efficient automatic prover ?

Claimed Difficulties in Using FM

- You have to be a mathematician.

- Formalism is hard to master.

- Not visual enough (no boxes, arrows, etc.).

- People will not be able to do formal proofs.

Genuine Difficulties (my own view)

- You have to think a lot before final coding.

- Incorporation in development process.

- Model building is an elaborate activity.

- Prover technology has to improve.

- Making proofs a design criterium.

- Poor quality of requirement documents.

Application Areas

- Train Systems
- Car Systems
- Avionics and Space
- Power Station Control
- Telecom
- Defense
- Complex Data Bases
- Large Business Network
- SmartCard Applications
- . . .

What is B ?

- A Method

- A Theory

- A Language

- A Set of Tools (Atelier B)

The B Method (1)

- To be used at various stages of system development

- Consists in building various mathematical models

- These models are proved to be:

    - Internally consistent

    - Refinements of each others

The B Method (2)

- The development process ends up with either:

  - Some HW/SW architecture

  - Some SW design

  - Some final SW code

  - Some final HW code

The B Theory (references)

- Predicate transformers (Dijkstra, Hoare, . . .)

- Refinement (Back, Hoare, Morgan, Morris, Wirth, Jones, . . .)

- Hiding (Parnas, Hoare, . . .)

- The B-Book (CUP) 1996.

- Event-B (CUP) to be published in 2004

The B Language

- A mathematical language: Set theory

- A structuring mechanism: Abstract “machine”

- A transition formalism: Parallel substitution

The Set of Tools: Atelier B

- Various analysers for the B language.

- A proof obligation generator.

- A prover (automatic and interactive).

- Various translators (towards ADA, $C^{++}, \ldots$)

- Several housekeeping tools.

Some Figures with ATELIER B (Version 3.6)

- Rules of Thumb:

n lines of final code implies $n/3$ proofs
90% of proofs discharged automatically
10% of proofs discharged interactively
400 interactive proofs per man-month

- 60,000 lines of final code $\leadsto$ 2,000 interactive proofs
- 2,000 interactive proofs $\leadsto$ 5 man-months
- Less expensive than heavy testing

Complex Systems (1)

- QUESTION: What is common to

  - an electronic circuit
  - a file transfer protocol
  - an airline booking system
  - a PC operating system
  - a nuclear plant control system
  - a SmartCard electronic purse
  - a launch vehicule flight controller

- ANSWER: They are all complex

Complex Systems (2)

- They are made of many parts
- They interact with a possibly hostile environment
- They involve several executing agents
- They require a high degree of correctness
- There construction spread over several years
- Their specifications are subjected to many changes
- Their construction process requires a talented team

Discrete Systems

- These systems operate in a discrete fashion

- Their dynamical behavior can be abstracted by:
    - A succession of steady states
    - Intermixed with sudden jumps

- The possibility of state changes is enormous

- The change frequency is unthinkable

- Such systems are called transition systems

Reasonings about (discrete) systems

- Two broad categories:

  - Test reasoning (98%)

  - Blue Print reasoning (10%)

Test Reasoning

- Based on laboratory execution

- Obvious incompleteness

- The oracle is usually missing

- Often implies postponing serious thinking

- Re-adapting and re-shaping after testing

- Reveals an immature technology

“Blue Print” Reasoning

- Based on a model: the “blue print”

- Describing the system with the required precision

- Completeness can be approached

- Serious thinking made on the model, not on the final system

- This is validated by proofs

- Reveals a mature technology

Discrete Formal Models (1)

- A unique paradigm to study ALL such systems: the formal model

- A formal model has a state made of:
  - Constants
  - Variables
  - Invariant

- It also has a number of transitions (called the events) made of:
  - a guard (necessary condition of transition enabling)
  - an action (effect of transition on state)

Discrete Formal Models (2)

- Formal reasoning by means of:
    - Invariant
    - Modalities

- Managing the complexity of formal models by:
    - Refinement
    - Decomposition
    - Generic instanciation

Goal of Event Systems 1

- It helps modelling discrete transition systems

- Such models can be used to develop software systems

- Also any kinds of transition systems, perhaps without informatics

Goal of Event Systems 2

- Modelling helps to formally fix the requirements

- But also to formally develop a design, that is an architecture

- Model of the environment together that of the futur system

- Event systems, quite often, are used to model closed systems

model

$< name >$

sets

$< identifier\ list >$

constants

$< identifier\ list >$

properties

$< predicate\ list >$

variables

$< identifier\ list >$

invariant

$< predicate\ list >$

initialisation

$< action >$

events

$< event\ list >$

end

Constants, Properties, Variables, Invariant

- Constant properties describe the laws the constants must follow

- Invariant predicates describe the laws the variables must follow

- This is done using Predicate Calculus and Set Theory

Initialisation

- The state variables have to be initialised

- This is done by means of generalized substitutions (see below)

Events (transitions)

- An event bears a name

- An event has a guard

    - It is the necessary condition for the event to occur

- An event has an action:

    - A generalized substitution (to be defined below)

    - It states the way the event modifies the variables

Syntax for Events

An event could be either guarded or non-guarded :

Guarded

select
$< conjuncted_list_of_predicates >$
then
$< generalized_substitution >$
end

Non-guarded

begin
$< generalized_substitution >$
end

Generalized Substitution: Multiple Assignment

- An multiple assignment could be deterministic or non-deterministic.

- A deterministic multiple assignment is:

$$< list_of_variables > := < list_of_expressions >$$

- A non-deterministic mutiple assignment is:

any $< list_of_variables >$ **where**
$< conjuncted_list_of_predicates >$
then
$< deterministic_assignment >$
end

Syntactic Facility for Non-deterministic Assignments

$v : P(v_0, v)$	**any** v' **where** $P(v, v')$ **then** $v := v'$ **end**
$v :\in S(v)$	**any** v' **where** $v' \in S(v)$ **then** $v := x$ **end**

Another Syntactic Facility: Parallel Assignment

$v := E(v,w) \quad \parallel$ $w := F(v,w)$	$v, w := E(v,w), F(v,w)$
any x **where** $P(x,v,w)$ **then** $v := E(x,v,w)$ **end** $\parallel$ **any** y **where** $Q(y,v,w)$ **then** $w := F(y,v,w)$ **end**	**any** x, y **where** $P(x,v,w) \;\wedge\; Q(y,v,w)$ **then** $v, w := E(x,v,w), F(y,v,w)$ **end**

Model Consistency Proof for an Event System (1)

- Given a model with state variables v and invariant $I(v)$

- Given an event of the form

$$\textbf{select } P(v) \textbf{ then } v := E(v) \textbf{ end}$$

- One has to prove the invariance of $I(v)$:

$$I(v) \ \wedge \ P(v) \ \Rightarrow \ I(E(v))$$

Notice: $I(E(v))$ is exactly $[v := E(v)]\, I(v)$

Model Consistency Proof for an Event System (2)

- Given a model with state variables v and invariant $I(v)$

- Given a non-deterministic event of the form

> **select** $P(v)$ **then**
> **any** w **where** $Q(v,w)$ **then** $v := E(v,w)$ **end**
> **end**

- One has to prove: (1) feasibility and (2) invariance:

$$(1) \quad I(v) \land P(v) \Rightarrow \exists w \cdot Q(v,w)$$

$$(2) \quad I(v) \land P(v) \land Q(v,w) \Rightarrow I(E(v,w))$$

Refinement

- Refinement is a technique to be used for:

    - taking account of more details

    - expressing some design decisions (data refinement)

Refinement Techniques (1)

- Each abstract event is refined by a concrete event

- A refined event must not contradict its abstraction

- Abstract world deals with state variables x

- Refined world deals with state variables y

- A, so-called, gluing invariant $J(x, y)$ links the two worlds

- Refinement laws to be defined below

Refinement Techniques (2)

- Some new events may appear: they refine “skip”

- They correspond to a finer time granularity

- Special constraints for new events:

  - The new events must not take control for ever

  - Refinement must preserve relative deadlockfreeness

Syntax for Refined Event Model (simplified)

model
$< model_name >$
refines
$< model_name >$
$\ldots$
end

Correct Refinement Proof (1)

- Given an abstraction with variables v and invariant $I(v)$
- Given a refinement with variables w
- Given the abstraction predicate $J(v, w)$
- Given an abstract event and refined event of the form

select $P(v)$ **then** $v := E(v)$ **end**	**select** $Q(w)$ **then** $w := F(w)$ **end**

- One has to prove:

$$I(v) \;\wedge\; J(v, w) \;\wedge\; Q(w) \;\Rightarrow\; P(v) \;\wedge\; J(E(v), F(w))$$

Correct Refinement Proof (2)

- Given abstract and refined events of the following shape:

select
$P(v)$
then
any x **where**
$A(x)$
then
$v := E(x, v)$
end
end

select
$Q(w)$
then
any y **where**
$B(y)$
then
$w := F(y, w)$
end
end

$$I(v) \;\land\; J(v,w) \;\land\; Q(w) \;\land\; B(y)$$
$$\Rightarrow$$
$$P(v) \;\land\; \exists x \cdot (\, A(x) \;\land\; J(E(v,x), F(w,y))\,)$$

Adding New Events in a Refinement

- Each new event must refine skip
- New events must not take control for ever
- For this, they all decrease a variant $V(w)$
- For a new event of the form

select $R(w)$ **then** $w := G(w)$ **end**

- One has then to prove

$$
\begin{array}{l}
I(v) \;\wedge\; R(w) \;\wedge\; J(v,w) \\
\Rightarrow \\
J(v,G(w)) \;\;\wedge\;\; V(G(w)) < V(w)
\end{array}
$$

Last Refinement Proof Obligation: Deadlockfreeness

- The refinement does not deadlock more often than the abstraction
- Given the concrete guards: $P_1(v) \ldots, P_m(v)$
- And the refined guards: $Q_1(w) \ldots, Q_n(w)$
- One has to prove:

$$
\begin{array}{l}
I(v) \;\wedge\; J(v,w) \;\wedge \\
P_1(v) \vee \ldots \vee P_m(v) \\
\Rightarrow \\
Q_1(w) \vee \ldots \vee Q_n(w)
\end{array}
$$

- Concrete deadlock: $\neg\,(Q_1(w) \vee \ldots \vee Q_n(w))$

- Abstract deadlock: $\neg\,(P_1(v) \vee \ldots \vee P_m(v))$

Main References

- Previous rules were elaborated by the Action System Group.

- Among the numerous references, here are two important ones:

- **R.J.R. Back and R. Kurki-Suonio.**
  Decentralization of Process Nets with Centralized Control.
  2nd ACM SIGACT-SIGOPS Symp. on
  Principles of Distributed Computing (1983)

- **M.J. Butler**
  Stepwise Refinement of Communicating Systems.
  Science of Computer Programming (1996)

Formal Development of an Event System

- Starts with a simple state

- And a very few events (quite often just one)

- As development proceeds, one may significantly enlarge the state

- One may also add many events

- At some point each event is working with part of the state only

- The rest of the state is thus just playing a noisy rôle

The Problem of Decomposition

- Given a large model $\mathcal{M}$.

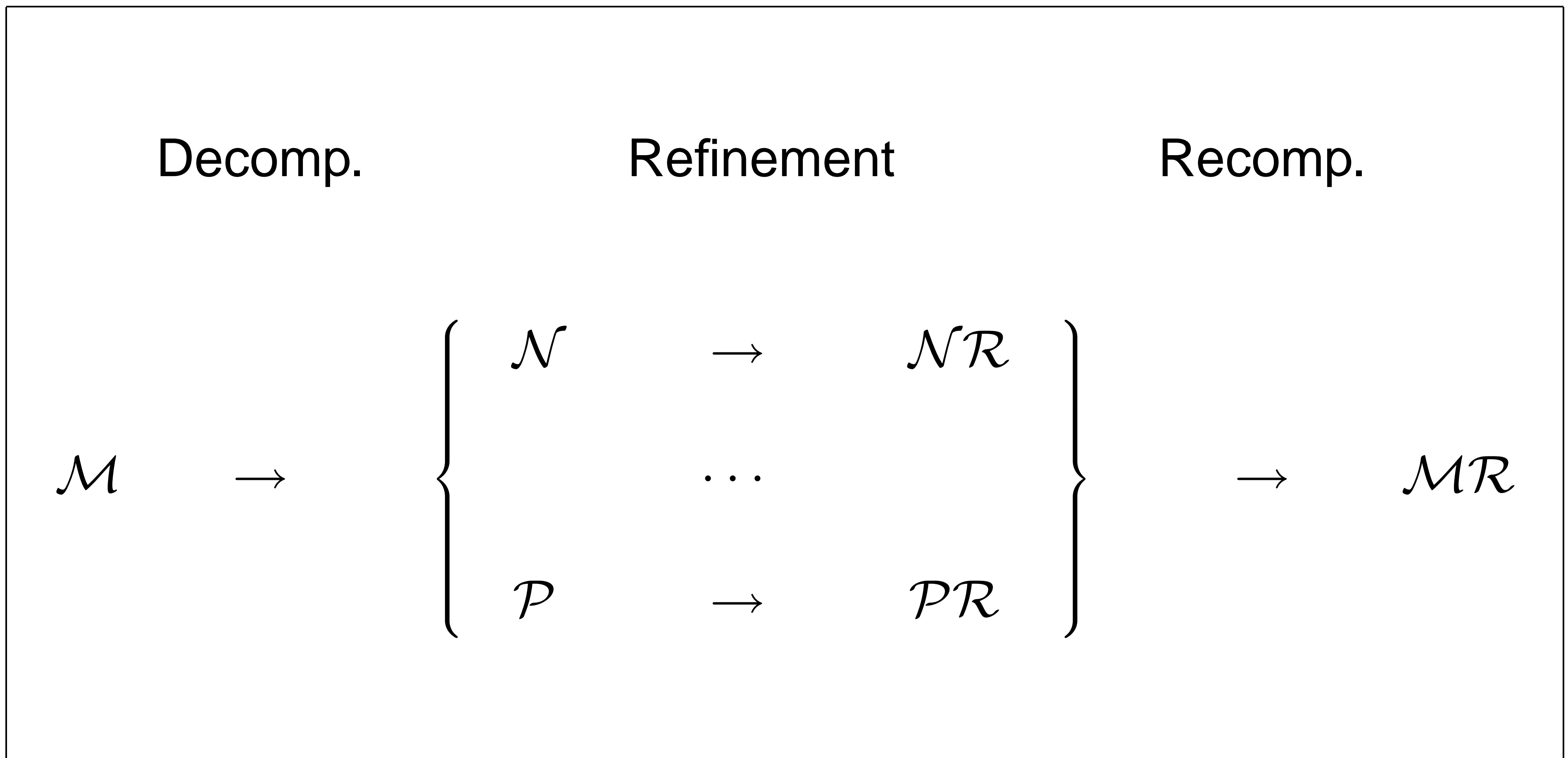
- $\mathcal{M}$ is first splitted into several sub-models, say $\mathcal{N}, \ldots, \mathcal{P}$.

- These sub-models are then refined yielding $\mathcal{NR}, \ldots, \mathcal{PR}$.

- These refined models are eventually put together yielding $\mathcal{MR}$.

- Model $\mathcal{MR}$ must be guaranteed to be a refinement of $\mathcal{M}$.

Decomposition Diagram



Application Overview

- Sequential Program Development
- Distributed Program Development
- Electronic Circuit Development
- System Engineering

Examples of Sequential Program Development

- Array Partitioning

- In Place Reversing of a Linear Chain (pointers)

- Sorting

- Bit Rate Optimization on a Network

- Garbage collection (pointers)

- Marking algorithm (pointers)

Examples of Distributed Program Development

- Transmission Protocol

- Synchronizing Processes on a Tree-shaped Network

- Distributed Mutual Exclusion

- Distributed Termination Detection

- Distributed Routing Algorithm for Mobile Agents

- Leader Election on a Connected Network

Examples of Electronic Circuit Development

- A Single Pulser

- A Binary Arbiter

- A Traffic Control System

Examples of System Engineering

- Car Complete Model

- Location Access Control

- Rail Switching System

- Real Time Process Scheduling

- Analysing Critical Scenarios

Conclusion

- A mathematical framework: discrete transition system

- Together with a development method:
  - refinement,
  - decomposition,
  - generic instantiation

- A number of case studies have been overviewed