Modelling by Patterns for Correct-by-Construction Process

Dominique Méry
Telecom Nancy,Université de Lorraine
dominique.mery@loria.fr

IFIP

Summary

Current Summary

1. Correctness by Construction
2. Distributed Algorithms
3. Discrete Models in Event B
4. The Inductive Paradigm
5. The Call-as-Event Paradigm
6. The Service-as-Event Paradigm
7. The Self-Healing P2P based Protocol
8. Conclusion

Correctness by Construction

- Correctness by Construction is a method of building software -based systems with **demonstrable correctness** for security- and safety-critical applications.
- Correctness by Construction advocates a **step-wise refinement** process from specification to code using tools for checking and transforming models.
- Correctness by Construction is an approach to software/system construction
  - starting with an abstract model of the problem.
  - progressively adding details in a step-wise and checked fashion.
  - each step guarantees and proves the correctness of the new concrete model with respect to requirements

The Cleanroom Method as CbC

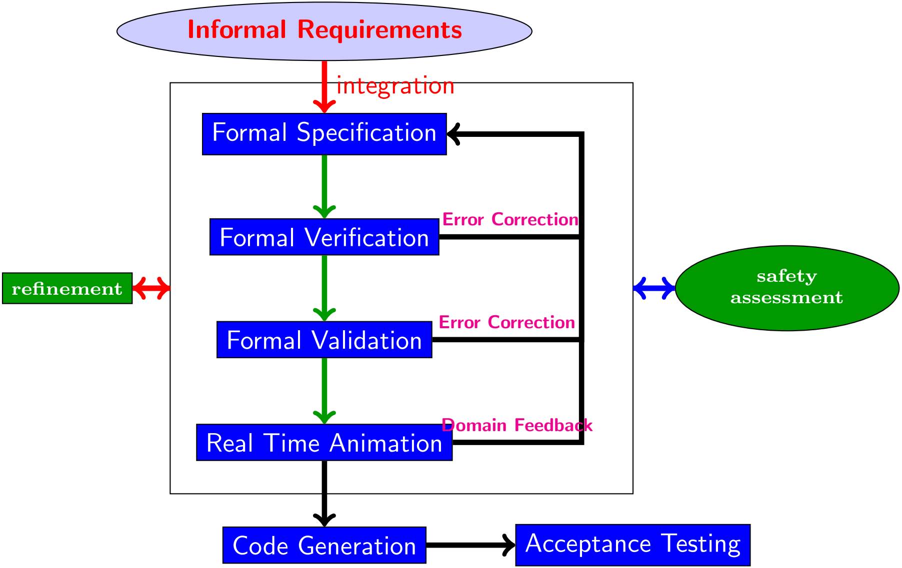
- The **Cleanroom** method, developed by Harlan Mills and his colleagues at IBM and elsewhere, attempts to do for software what cleanroom fabrication does for semiconductors: to achieve quality by keeping defects out during fabrication.
- In semiconductors, **dirt** or **dust** that is allowed to **contaminate** a chip as it is being made cannot possibly be removed later.
- But we try to do the equivalent when we write programs that are full of bugs, and then attempt to remove them all using debugging.

The Cleanroom Method as CbC

The Cleanroom method, then, uses a number of techniques to develop software carefully, in a well-controlled way, so as to avoid or eliminate as many defects as possible before the software is ever executed. Elements of the method are:

- specification of all components of the software at all levels;
- stepwise refinement using constructs called "box structures";
- verification of all components by the development team;
- statistical quality control by independent certification testing;
- no unit testing, no execution at all prior to certification testing.

Critical System Development Life-Cycle Methodology



Overview of Methodology

- Informal Requirements: Restricted form of natural language.
- Formal Specification: Modeling language like Event-B , Z, ASM, VDM, TLA+...
- Formal Verification: Theorem Prover Tools like PVS, Z3, SAT, SMT Solver...
- Formal Validation: Model Checker Tools like ProB, UPPAAL , SPIN, SMV ...
- Real-time Animation: **Our proposed approach ... Real-Time Animator ...**
- Code Generation: **Our proposed approach ... EB2ALL: EB2C, EB2C++, EB2J, EB2C# ...**
- Acceptance Testing: Failure Mode, Effects and Critically analysis(FMEA and FMEA), System Hazard Analyses(SHA)

Case Studies

- *Colin Boyd and Anish Mathuria. Protocols Authentication and Key Establisment. Springer 2003.*
- *C. C. Marquezan and L. Z. Granville. Self-* and P2P for Network Management - Design Principles and Case Studies. Springer Briefs in Computer Science. Springer, 2012.*
- *Pacemaker Challenge Contribution*

Current Summary

Distributed Algorithms: first steps in verification

First steps in a new world by proving the mutual exclusion algorithm of Ricart and Agrawala using a sopund and semantically complete temporal proof system and a graphical notation called proof lattice

Distributed Algorithms: first steps in verification

First steps in a new world by proving the mutual exclusion algorithm of Ricart and Agrawala using a sopund and semantically complete temporal proof system and a graphical notation called proof lattice

Ricart, Glenn; Agrawala, Ashok K. (1 January 1981). "An optimal algorithm for mutual exclusion in computer networks". Communications of the ACM. 24 (1): 917.

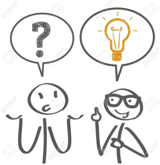
Distributed Algorithms: first steps in verification

First steps in a new world by proving the mutual exclusion algorithm of Ricart and Agrawala using a sopund and semantically complete temporal proof system and a graphical notation called proof lattice

Ricart, Glenn; Agrawala, Ashok K. (1 January 1981). "An optimal algorithm for mutual exclusion in computer networks". Communications of the ACM. 24 (1): 917.

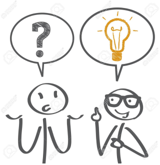
- Definition of sound and semantically complete temporal proof system.
- Annotations and proofs were not machine-assisted.
- How to explain why the algorithm was correct?
- Carvalho and Roucairol published an improvement of the RA algorithm using some kind of abstraction and simplification.

Verifying Distributed Algorithms: lessons learnt



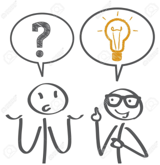
Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:



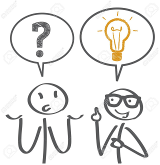
Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:
  - local annotation but global state



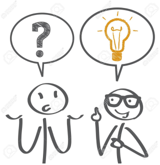
Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...



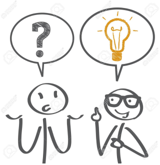
Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)

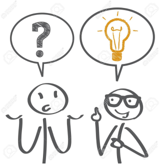


Verifying Distributed Algorithms: lessons learnt

- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions

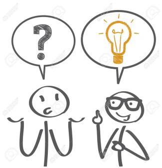


Verifying Distributed Algorithms: lessons learnt



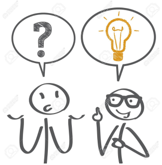
- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process

Verifying Distributed Algorithms: lessons learnt



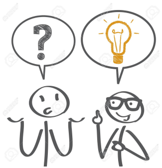
- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, . . .
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process
- Discovering why the distributed system is working

Verifying Distributed Algorithms: lessons learnt



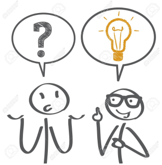
- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process
- Discovering why the distributed system is working
- Explaining in an abstract and simple way why it is working

Verifying Distributed Algorithms: lessons learnt



- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process
- Discovering why the distributed system is working
- Explaining in an abstract and simple way why it is working
- Crocos was an integrated environment for interactive verification of SDL specifications (CAV 1992) using Isabelle and Concerto

Verifying Distributed Algorithms: lessons learnt



- Discovering the correct annotation:
  - local annotation but global state
  - communications: synchronous, asynchronous, coordination, lossy, unsecure, ...
  - Programming model versus execution model (UNITY)
  - Hidden assumptions or implicit assumptions
- Replaying the proof process
- Discovering why the distributed system is working
- Explaining in an abstract and simple way why it is working
- Crocos was an integrated environment for interactive verification of SDL specifications (CAV 1992) using Isabelle and Concerto
- Discovering why the *distributed* process is correct by a simple **abstraction**.

Distributed Algorithms: using proof assistant

Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Distributed Algorithms: using proof assistant

Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

- Using the PVS proof assistant

Distributed Algorithms: using proof assistant

Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

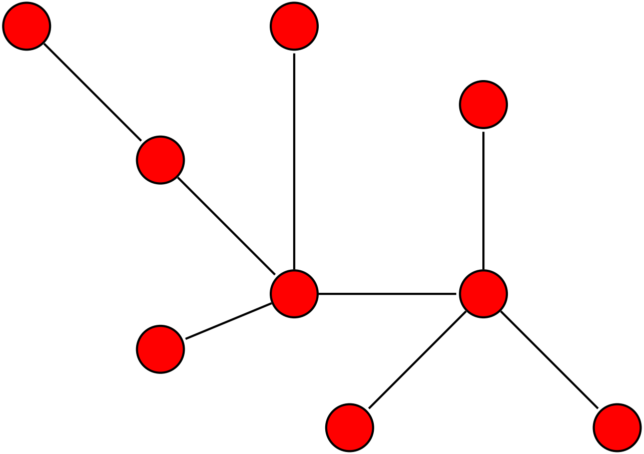
- Using the PVS proof assistant
- Modelling in I/O automata

Distributed Algorithms: using proof assistant

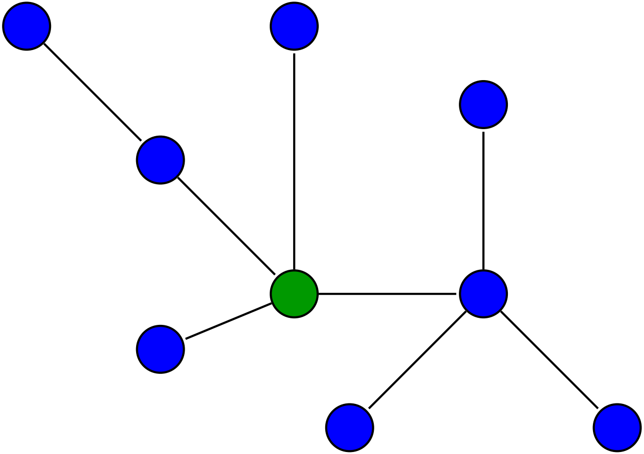
Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

- Using the PVS proof assistant
- Modelling in I/O automata
- Proofs difficult to read.

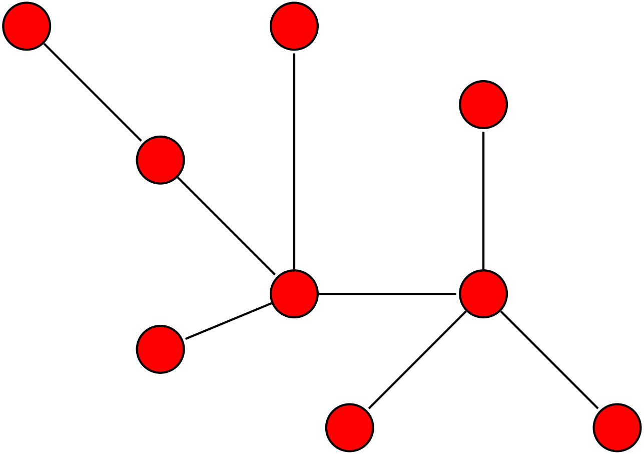
Scenario for the leader election protocol



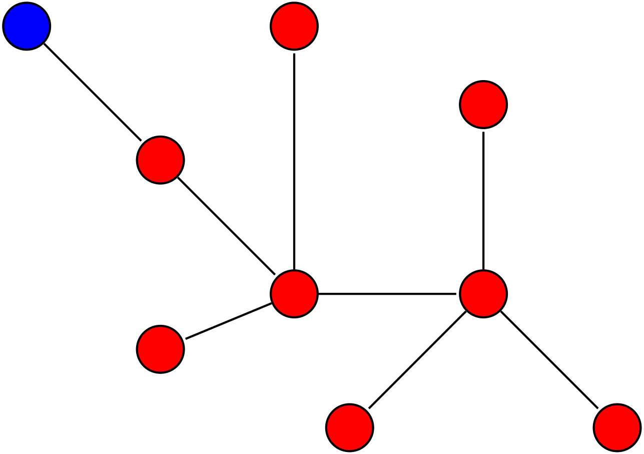
The leader election



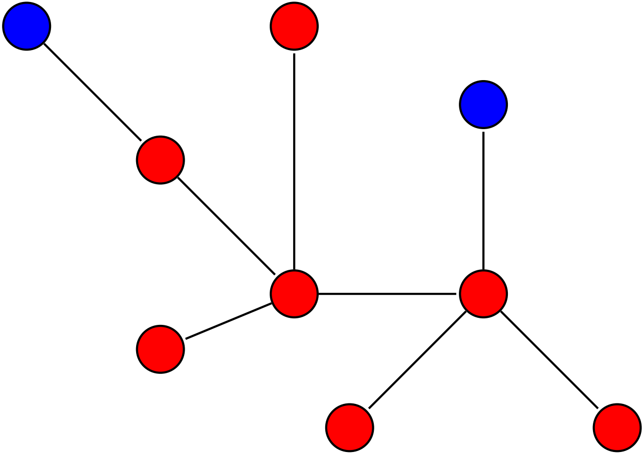
Scenario for the leader election protocol



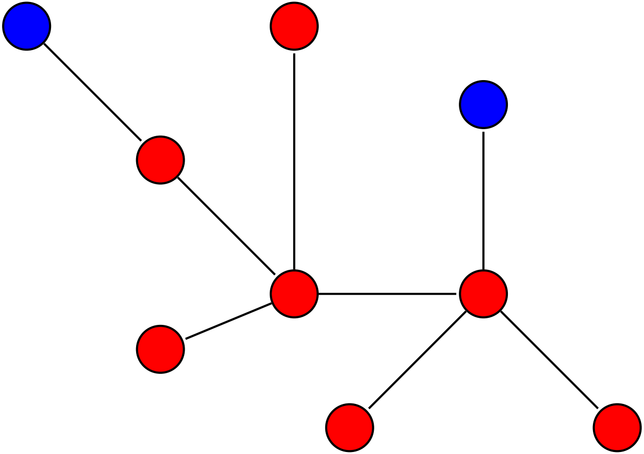
Scenario for the leader election protocol



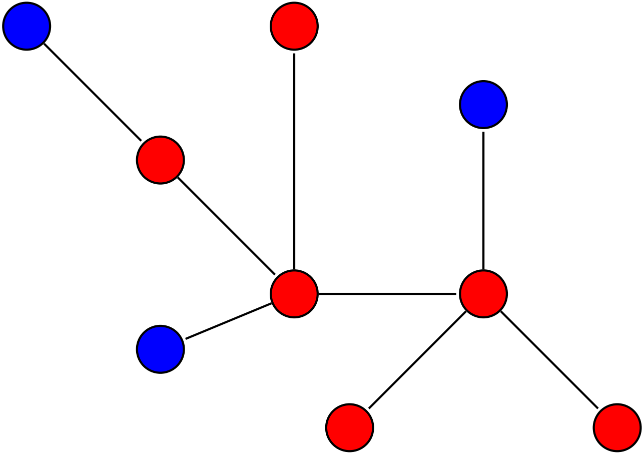
Scenario for the leader election protocol



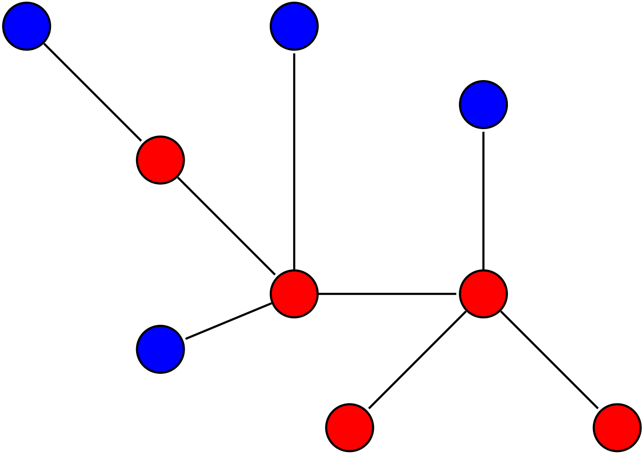
Scenario for the leader election protocol



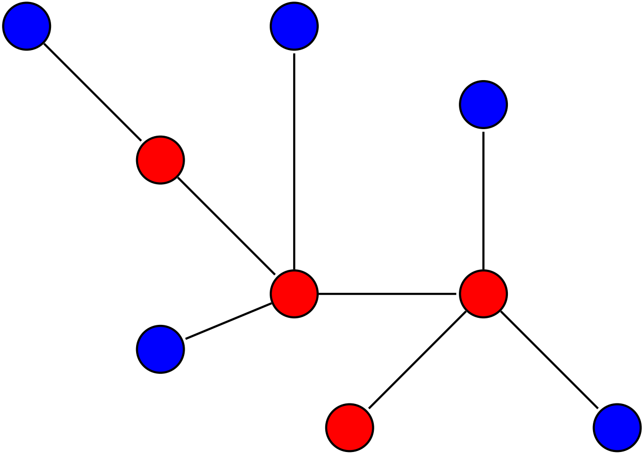
Scenario for the leader election protocol



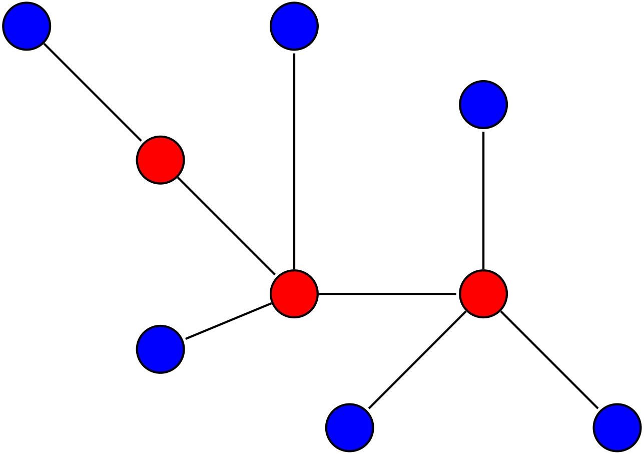
Scenario for the leader election protocol



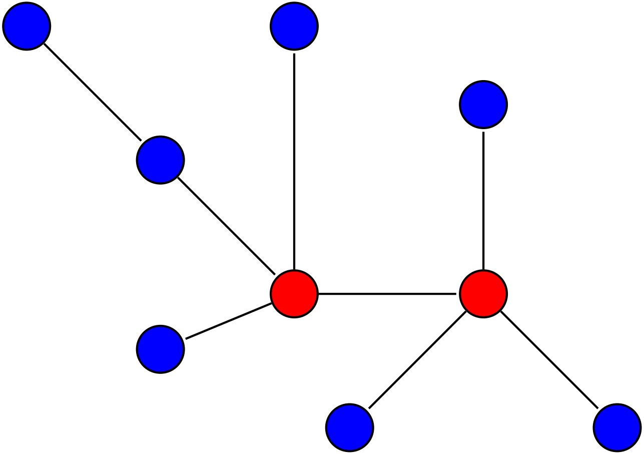
Scenario for the leader election protocol



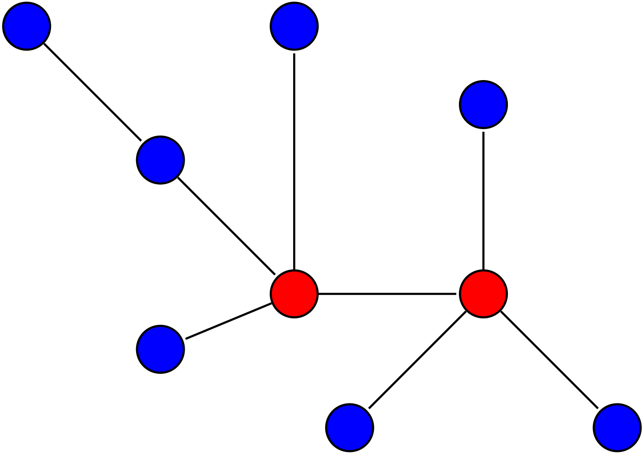
Scenario for the leader election protocol



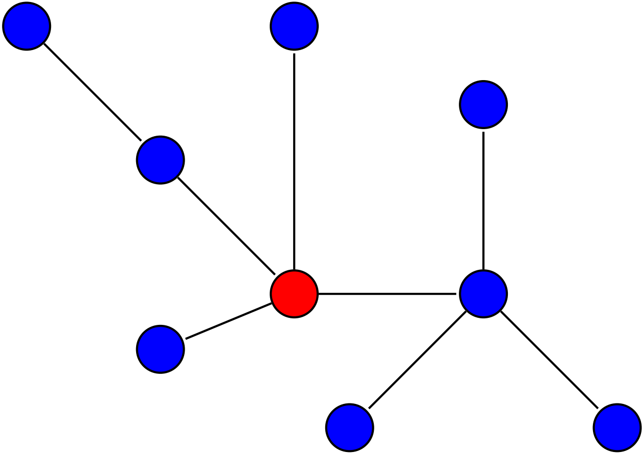
Scenario for the leader election protocol



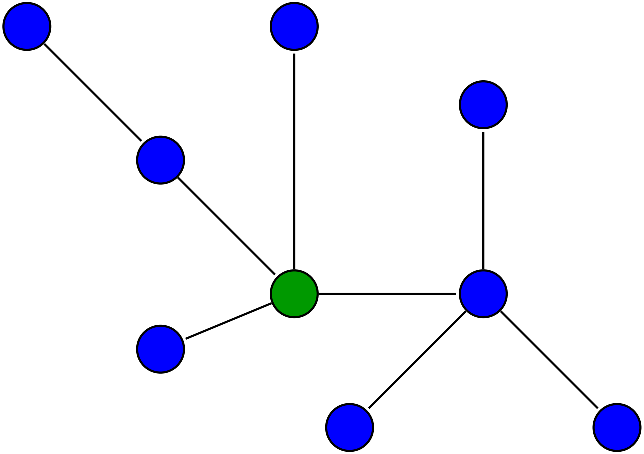
Scenario for the leader election protocol



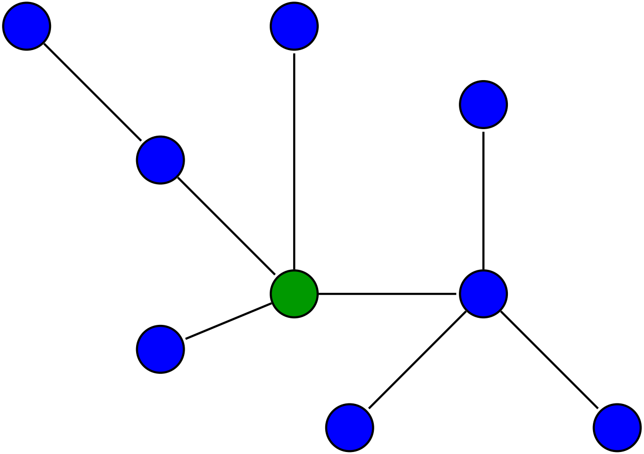
Scenario for the leader election protocol



Scenario for the leader election protocol



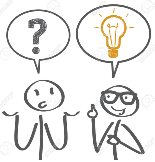
The leader election



Distributed Algorithms: quest for verification

Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

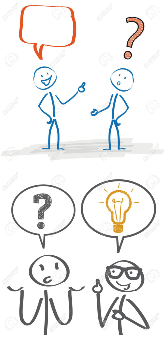
Distributed Algorithms: quest for verification



Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

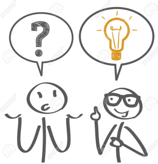
Distributed Algorithms: quest for verification



Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

Distributed Algorithms: quest for verification



Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

- Main inductive property: *a forest is converging to a tree.*
- ...

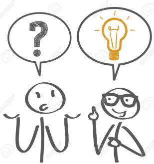
Distributed Algorithms: quest for verification

Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

- Main inductive property: *a forest is converging to a tree.*
- . . . but it should exist eventually a tree.

Distributed Algorithms: quest for verification



Marco Devillers, W. O. David Griffioen, Judi Romijn, Frits W. Vaandrager: Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394. Formal Methods in System Design 16(3): 307-320 (2000)

Jean-Raymond Abrial, Dominique Cansell, Dominique Méry: A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol. Formal Asp. Comput. 14(3): 215-227 (2003)

- Main inductive property: *a forest is converging to a tree.*
- . . . but it should exist eventually a tree.
- Knowledges over graphs should be somewhere modelled.

Guidelines for a Modelling Language

How to Solve It by Pólya

If you can't solve a problem, then there is an easier problem you can solve: find it. or *If you cannot solve the proposed problem, try to solve first some related problem. Could you imagine a more accessible related problem?*.

- *patterns* are a key concept for solving problems;
- Moreover, another key concept is the refinement of models handling the complex nature of such systems: the refinement is used for constructing models or patterns.
- Revisit a list of patterns which can be used for developing programs or systems using the refinement and the proof as a mean to check the whole process.

Paradigms for Modelling Systems

Our aim is to help users, mainly students, to learn how to use the refinement relationship when developing software-based systems.

Paradigm

A paradigm is a distinct set of patterns, including theories, research methods, postulates, and standards for what constitutes legitimate contributions to designing programs.

Pattern

A pattern for modelling in Event-B is a set (project) of contexts and machines that have parameters as sets, constants, variables ...

- The Inductive Paradigm
- The Call-as-Event Paradigm
- The Service -as-Event Paradigm
- The Composition/Decomposition Paradigm

Triptych of Software/System Engineering

Software/System development ideally proceeds in three phases according to Dines Børner::

$$\mathcal{D}, \mathcal{S} \quad \Rightarrow \quad \mathcal{R}$$

- First, a phase of **domain engineering** $\mathcal{D}$: an analysis of the application domain leads to a description of that domain.
- Second, a phase of **requirements engineering** $\mathcal{R}$: an analysis of the domain description leads to a prescription of requirements to software for that domain.
- Third, a phase of **software/system design** $\mathcal{S}$: an analysis of the requirements prescription leads to software for that domain.

Triptych of Software/System Engineering

Software/System development ideally proceeds in three phases according to Dines Børner::

$$\mathcal{D}, \mathcal{S} \quad \Rightarrow \quad \mathcal{R}$$

Pre/Post Specification

- $\mathcal{R}$: pre/post.
- $\mathcal{D}$: integers, reals, ...
- $\mathcal{S}$: algorithm, program, ...

Triptych of Software/System Engineering

Software/System development ideally proceeds in three phases according to Dines Børner::

$$\mathcal{D}, \mathcal{S} \quad \Rightarrow \quad \mathcal{R}$$

Pre/Post Specification

- $\mathcal{R}$: pre/post.
- $\mathcal{D}$: integers, reals, ...
- $\mathcal{S}$: algorithm, program, ...

- Semantical relationship
- Verification by induction principle

Triptych of Software/System Engineering

Software/System development ideally proceeds in three phases according to Dines Børner::

$$\mathcal{D}, \mathcal{S} \quad \Rightarrow \quad \mathcal{R}$$

System Modelling

- $\mathcal{R}$: safety properties in Event-B
- $\mathcal{D}$: theories, context in Event-B
- $\mathcal{S}$: machines for reactive systems

Triptych of Software/System Engineering

Software/System development ideally proceeds in three phases according to Dines Børner::

$$\mathcal{D}, \mathcal{S} \quad \Rightarrow \quad \mathcal{R}$$

System Modelling

- $\mathcal{R}$: safety properties in Event-B
- $\mathcal{D}$: theories, context in Event-B
- $\mathcal{S}$: machines for reactive systems

- Checking proof obligations
- Refinement of models

Triptych of Software/System Engineering

Software/System development ideally proceeds in three phases according to Dines Børner::

$$\mathcal{D}, \mathcal{S} \quad \Rightarrow \quad \mathcal{R}$$

- First, a phase of **domain engineering** $\mathcal{D}$: an analysis of the application domain leads to a description of that domain.
- Second, a phase of **requirements engineering** $\mathcal{R}$: an analysis of the domain description leads to a prescription of requirements to software for that domain.
- Third, a phase of **software/system design** $\mathcal{S}$: an analysis of the requirements prescription leads to software for that domain.

Current Summary

Modelling systems in Event-B

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

Modelling systems in Event-B

MACHINE
  m
SEES
  c
VARIABLES
  x
INVARIANT
  $I(x)$
THEOREMS
  $Q(x)$
INITIALISATION
  $Init(x)$
EVENTS
  $\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e :$

$\Gamma(m) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

Modelling systems in Event-B

MACHINE
  m
SEES
  c
VARIABLES
  x
INVARIANT
  $I(x)$
THEOREMS
  $Q(x)$
INITIALISATION
  $Init(x)$
EVENTS
  $\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e :$

$\Gamma(m) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

$\Gamma(m) \vdash \forall x \in Values : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

Modelling systems in Event-B

MACHINE
  m
SEES
  c
VARIABLES
  x
INVARIANT
  $I(x)$
THEOREMS
  $Q(x)$
INITIALISATION
  $Init(x)$
EVENTS
  $\ldots e$
END

c defines the static environment for the proofs related to m: sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e :$

$\Gamma(m) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

$\Gamma(m) \vdash \forall x \in Values : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

e
any
  u
where
  $G(x, u)$
then
  $x :| (R(u, x, x'))$
end

or e is **observed** $x \xrightarrow{e} x'$

Event B Structure and Proofs

CONTEXT	MACHINE
CONTEXT *ctxt_id_2* **EXTENDS** *ctxt_id_1* **SETS** s **CONSTANTS** c **AXIOMS** $A(s,c)$ **THEOREMS** $T_c(s,c)$ **END**	**MACHINE** *machine_id_2* **REFINES** *machine_id_1* **SEES** *ctxt_id_2* **VARIABLES** v **INVARIANTS** $I(s,c,v)$ **THEOREMS** $T_m(s,c,v)$ **VARIANT** $V(s,c,v)$ **EVENTS** **EVENT** e **any** x **where** $G(s,c,v,x)$ **then** $v :\mid BA(s,c,v,x,v')$ **end** **END**

Invariant preservation	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\wedge BA(s,c,v,x,v')$ $\Rightarrow I(s,c,v')$
Event feasibility	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\Rightarrow \exists v'.BA(s,c,v,x,v')$
Variant modelling progress	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\wedge BA(s,c,v,x,v')$ $\Rightarrow V(s,c,v') < V(s,c,v)$
Theorems	$A(s,c) \Rightarrow T_c(s,c)$ $A(s,c) \wedge I(s,c,v)$ $\Rightarrow T_m(s,c,v)$

Election in One Shot: Building a Spanning Tree

Election in One Shot: Building a Spanning Tree

MACHINE
$ELECTION$
SEES $GRAPH$
VARIABLES rt, ts, ok
INVARIANT

$$\boxed{\begin{array}{l} rt \in ND \\ ts \in ND \leftrightarrow ND \\ ok \in BOOL \\ ok = TRUE \\ \quad \Rightarrow \mathsf{spanning}\,(rt, ts, gr) \end{array}}$$

INITIALISATION $Init(x)$
EVENT election $\widehat{=}$
when
$ok = FALSE$
then
$rt, ts : |(\mathsf{spanning}\,(rt', ts', gr))$
$ok := TRUE$
endEND

Election in One Shot: Building a Spanning Tree

MACHINE
$ELECTION$
SEES $GRAPH$
VARIABLES rt, ts, ok
INVARIANT

$$\boxed{\begin{array}{l} rt \in ND \\ ts \in ND \leftrightarrow ND \\ ok \in BOOL \\ ok = TRUE \\ \quad \Rightarrow \mathsf{spanning}\,(rt, ts, gr) \end{array}}$$

INITIALISATION $Init(x)$
EVENT election $\widehat{=}$
when
$ok = FALSE$
then
$rt, ts : |(\mathsf{spanning}\,(rt', ts', gr))$
$ok := TRUE$
endEND

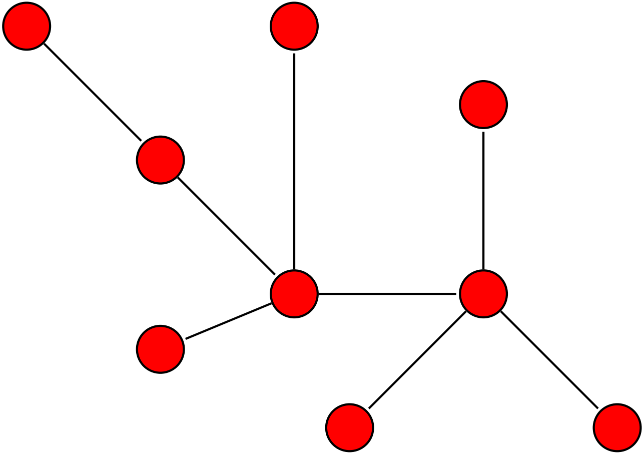
CONTEXT $GRAPH$
$(ax1)\ gr \subseteq ND \times ND$
$(ax2)\ gr = gr^{-1}$
$(ax3)\ \mathsf{dom}\,(gr) = ND$
$(ax4)\ \mathsf{id}\,(ND)\ \cap\ gr\ =\ \varnothing$

$$(ax5)\ \forall p \cdot \left(\begin{array}{l} p \subseteq ND\ \wedge \\ p \subseteq t^{-1}\,[p] \\ \Longrightarrow \\ p = \varnothing \end{array}\right)$$

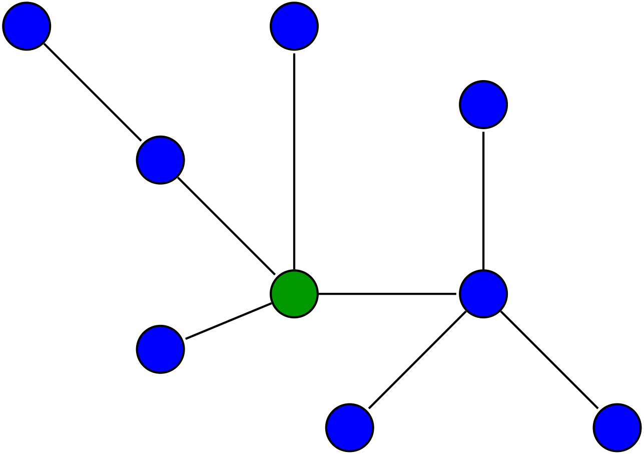
$(Th1) fn \in ND \rightarrow (ND \nrightarrow ND)$
$\forall (r, t) \cdot$

$$\left(\begin{array}{l} r \in ND\ \wedge \\ t \in ND \nrightarrow ND \\ \Longrightarrow \\ (t = fn(r)\ \Leftrightarrow\ \mathsf{spanning}\,(r, t, gr)) \end{array}\right)$$

Scenario for the leader election protocol

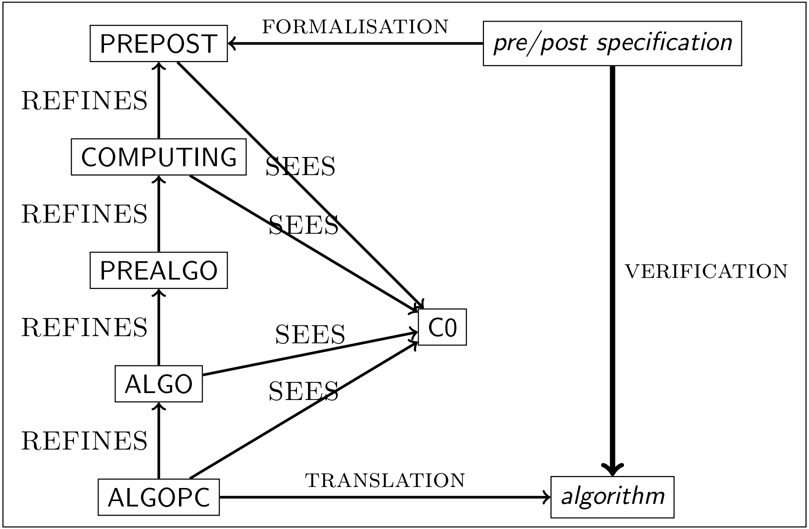


Scenario for the leader election protocol



Current Summary

The Iterative Pattern



General context C0

CONTEXT $C0$

SETS

U

CONSTANTS

$x, v, d0, f, D$

AXIOMS

$axm1 : x \in \mathbb{N}$

$axm25 : D \subseteq U$

$axm24 : f \in D \to D$

$axm23 : d0 \in D$

$axm2 : v \in \mathbb{N} \to D$

$axm3 : v(0) = d0$

$axm4 : \forall n \cdot n \in \mathbb{N} \Rightarrow v(n+1) = f(v(n))$

$th1 : Q(d_0, d) \equiv (d = v(x))$

- the sequence v expresses the post-condition $Q(d_0, d)$ with the precondition $P(d_0)$.
- $Q(d_0, d)$ is equivalent to $d = v(x)$.
- The theorem $th1$ should be proved in the context C0. he

General PREPOST Machine

MACHINE $PREPOST$
SEES $C0$
variables
r
invariants
$inv1 : r \in D$
EVENTS
initialisation
begin
$act1 : r :\in D$
end
EVENT computing
begin
$act1 : r := v(x)$
end
end

- The theorem $th1$ is validating the definition of the result r to compute.
- The event computing is expressing the *contract* of the given problem.
- it by a very simple problem that is the computation of the function n^2 using the addition operator.

First Refinement COMPUTING: Inductive Computation

EVENT INITIALISATION
begin
$act1 : r :\in D$
$act3 : vv := \{0 \mapsto d0\}$
$act5 : k := 0$
end

INITIALISATION is initializing the variables with respect to the initial values of the sequences of the context.

First Refinement COMPUTING: Inductive Computation

EVENT computing
  REFINES computing,
  when
    $grd1 : x \in dom(vv)$
  then
    $act1 : r := vv(x)$
  end
END

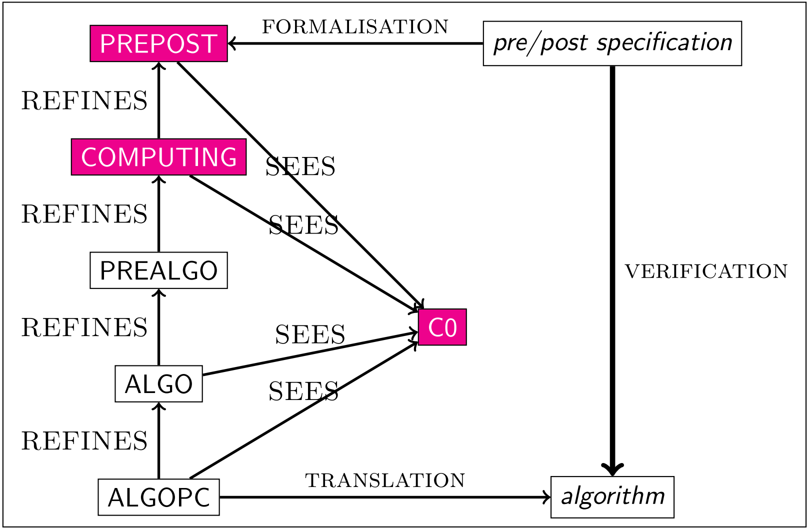
computing is imply observing that the result is computed *simulating* the sequence vv.

First Refinement COMPUTING: Inductive Computation

EVENT step
when
$grd1 : x \notin dom(vv)$
then
$act2 : vv(k+1) := f(vv(k))$
$act4 : k := k+1$
end

step is *simulating* the computation of the values of the sequence vv as a model computation.

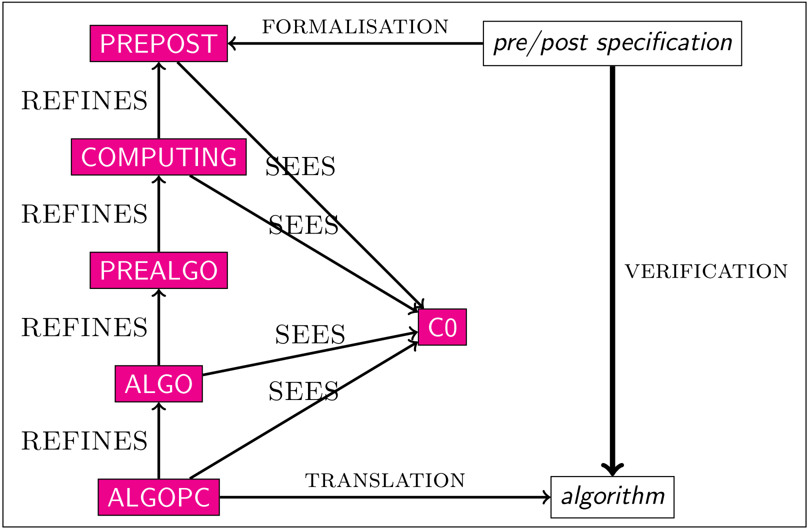
The Iterative Pattern



Completing the machines

- PREALGO: adding new variables for pointing out the necessary values to store cvv
- ALGO: hiding the model variables storing the unnecessary values of sequence vv
- ALGOPC; adding control variable c

The Iterative Pattern



Translation of Event-B Models

Listing 1: Function derived from pattern for the sequence v

```
type(D)   f(int  x)
{int   r,k,cv,or,ok,ocv;
  r=0;k=0;cv=0;or=0;ok=k;ocv=cv;
  while  (k<x)
         {
           ok=k;ocv=cv;
           k=ok+1;
           cv=f(ocv);
         }
  r=cv;   return(r);}
```

Translation of Event-B Models

Comments

- The produced algorithm can be now checked using another proof environment as for instance Frama-C.
- The inductive property of the invariant is clearly verified and is easily derived from the Event-B machines.
- The verification is not required, since the system is correct by construction but it is a checking of the process itself
- the project called ITERATIVE-PATTERN;
- the project is the pattern itself
- The invariants of the Event-B models can be reused in the verification using Frama-C, for instance, and the verification of the resulting algorithm is a confirmation of the translation.

Translation of Event-B Models

Listing 2: Function derived from pattern power3

```
#include <limits.h>
/*@  requires  0 <= x;
     requires x*x*x <= INT_MAX  ;
     ensures \result ==x*x*x;
*/
int power3(int x)
{int  r,ocz,cz,cv,cu,ocv,cw,ocw,ct,oct,ocu,k,ok;
  cz=0;cv=0;cw=1;ct=3;cu=0;  ocw=cw;ocz=cz;
  oct=ct;ocv=cv;ocu=cu;k=0;ok=k;
      /*@ loop invariant cz == k*k*k;
        @ loop invariant cu  == k;
        @ loop invariant cv+ct==3*(cu+1)*(cu+1);
        @ loop invariant cz+cv+cw==3*(cu+1)*(cu+1)*(cu+1);
        @ loop invariant cv== 3*cu*cu;
        @ loop invariant cw == 3*cu+1;
        @ loop invariant k  <= x;
        @ loop assigns ct,oct,cu,ocu,cz,ocz,k,cv,cw,r,ok;
        @ loop assigns ocv,ocw;*/
  while (k<x)
        {
          ocz=cz;ok=k;ocv=cv;ocw=cw;oct=ct;ocu=cu;
          cz=ocz+ocv+ocw;
          cv=ocv+oct;
          ct=oct+6;
          cw=ocw+3;
          cu=ocu+1;
          k=ok+1;}
  r=cz;return(r);}
```

Summary for proof obligations

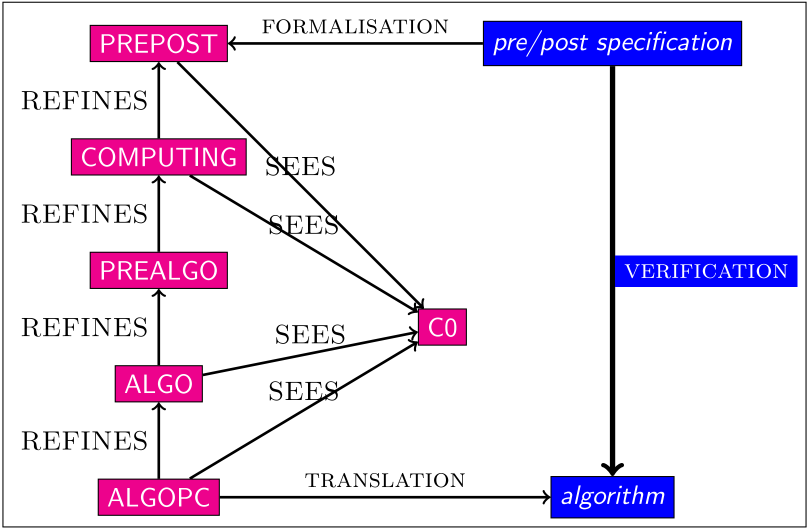
Name	Total	Automatic	Interactive
ex-induction	40	36	4
C0	2	0	2
PREPOST	4	4	0
COMPUTING	16	14	2
PREALGO	9	9	0
ALGO	6	6	0
ALGOPC	3	3	0

Translation of Event-B Models

Summary

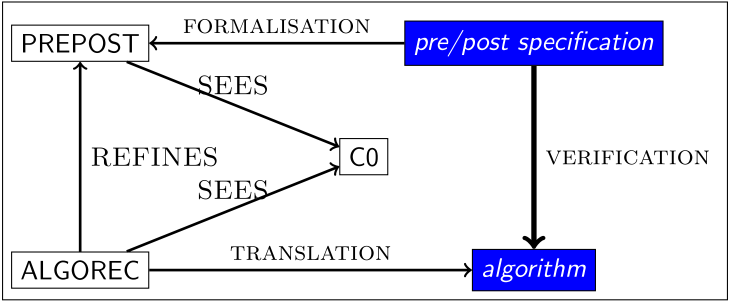
- The loop invariant is inductive but Frama-C does not prove it completely.
- Not the case with the RODIN platform which is able to discharge the whole set of proof obligations.
- However, the Event-B model is using auxiliary knowledge over sequences used for defining the computing process.
- The most difficult theorem is to prove that $\forall n \in \mathbb{N} : z_n = n * n * n$.

The Iterative Pattern

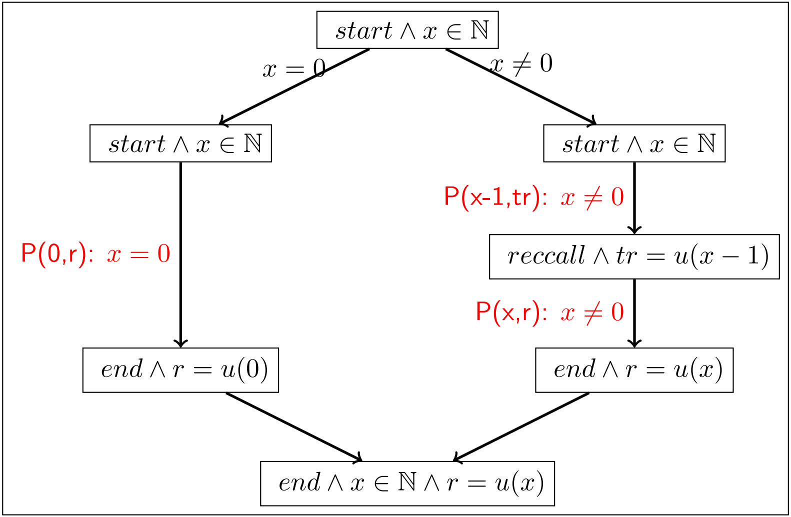


Current Summary

The recursive pattern



The refinement diagram



Ideas for producing algorithms

- **EVENT** rec%PROC(h(x),y)%P(y) is simply simulating the recursive call of the same function.
- The invariant is defined in a simpler way by analysing the inductive structure and a control variable is introduced for structuring the inductive computation.

```
EVENT
  e
where
  ℓ = ℓ1
  g_{ℓ1,ℓ2}(x)
then
  ℓ := ℓ2
  x := f_{ℓ1,ℓ2}(x)
end
```

```
EVENT
  rec%PROC(h(x),y)%P(y)
any y
where
  ℓ = ℓ1
  g_{ℓ1,ℓ2}(x, y)
then
  ℓ := ℓ2
  x := f_{ℓ1,ℓ2}(x, y)
end
```

```
EVENT
  call%APROC(h(x),y)%P(y)
any y
where
  ℓ = ℓ1
  g_{ℓ1,ℓ2}(x, y)
then
  ℓ := ℓ2
  x := f_{ℓ1,ℓ2}(x, y)
end
```

The computation of the function x^2

variables

r, c, tr

invariants

$inv1 : r \in \mathbb{N}$

$inv2 : c = end \Rightarrow r = n * n$

$inv3 : c = callrec \Rightarrow n \neq 0$

$inv4 : c = callrec \Rightarrow tr = (n-1) * (n-1)$

$inv5 : c \in C$

$inv6 : tr \in \mathbb{N}$

$inv7 : c = end \Rightarrow r = n * n$

$inv8 : c = end \wedge n \neq 0$

$\Rightarrow tr = (n-1) * (n-1) \wedge r = tr + 2 * (n-1) + 1$

$inv9 : c = callrec \Rightarrow n * n = tr + 2 * (n-1) + 1$

The computation of the function x^2

EVENT INITIALISATION
  begin
    $act1 : r := 0$
    $act2 : c := start$
    $act3 : tr :\in \mathbb{N}$
  end
EVENT square0
REFINES square(n;r)
  when
    $grd1 : c = start$
    $grd2 : n = 0$
  then
    $act1 : c := end$
    $act2 : r := 0$
  end

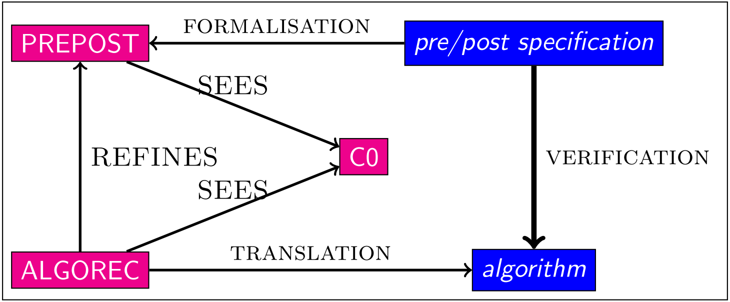
EVENT squaren
REFINES square(n;r)
  when
    $grd1 : c = callrec$
  then
    $act1 : r := tr + 2 * (n - 1) + 1$
    $act2 : c := end$
  end
EVENT rec%square(n-1;tr)
  when
    $grd1 : c = start$
    $grd2 : n \neq 0$
  then
    $act1 : c := callrec$
    $act2 : tr := (n - 1) * (n - 1)$
  end

Summary for proof obligations

Summary for proof obligations

Name	Total	Automatic	Interactive
cae-square	34	32	2
square0	3	2	1
specquare	2	2	0
square	29	28	1

The recursive pattern



Comments

- Proofs are easier and simpler
- Invariant is simple to find
- Translation is automatic
- Students are not happy with it and tools are not always set for this verification.

Current Summary

Conclusion

- Paradigm for planning refinements:
  - The Service -as-Event Paradigm: the distributed pattern
  - The Composition/Decomposition Paradigm: mechanisms-based pattern.
- Graphical notation: the refinement diagram
- Possibly combining the Visidia model and the refinement process (http://rimel.loria.fr and http://visidia.labri.fr)

Conclusion

- Paradigm for planning refinements:
  - The Service -as-Event Paradigm: the distributed pattern
  - The Composition/Decomposition Paradigm: mechanisms-based pattern.
- Graphical notation: the refinement diagram
- Possibly combining the Visidia model and the refinement process (http://rimel.loria.fr and http://visidia.labri.fr)
- Teaching why and how distributed algorithms are working: the parachutist process

Conclusion

- Paradigm for planning refinements:
  - The Service -as-Event Paradigm: the distributed pattern
  - The Composition/Decomposition Paradigm: mechanisms-based pattern.
- Graphical notation: the refinement diagram
- Possibly combining the Visidia model and the refinement process (http://rimel.loria.fr and http://visidia.labri.fr)
- Teaching why and how distributed algorithms are working: the parachutist process

Next

- Transformation of Event-B models into *programming*, Dynamic networks

Conclusion

- Paradigm for planning refinements:
  - The Service -as-Event Paradigm: the distributed pattern
  - The Composition/Decomposition Paradigm: mechanisms-based pattern.
- Graphical notation: the refinement diagram
- Possibly combining the Visidia model and the refinement process (http://rimel.loria.fr and http://visidia.labri.fr)
- Teaching why and how distributed algorithms are working: the parachutist process

Next

- Transformation of Event-B models into *programming*, Dynamic networks
- Probabilistic assumptions

Conclusion

- Paradigm for planning refinements:
  - The Service -as-Event Paradigm: the distributed pattern
  - The Composition/Decomposition Paradigm: mechanisms-based pattern.
- Graphical notation: the refinement diagram
- Possibly combining the Visidia model and the refinement process (http://rimel.loria.fr and http://visidia.labri.fr)
- Teaching why and how distributed algorithms are working: the parachutist process

Next

- Transformation of Event-B models into *programming*, Dynamic networks
- Probabilistic assumptions
- Atlas of *correct-by-construction* distributed algorithms

Conclusion

- Paradigm for planning refinements:
  - The Service -as-Event Paradigm: the distributed pattern
  - The Composition/Decomposition Paradigm: mechanisms-based pattern.
- Graphical notation: the refinement diagram
- Possibly combining the Visidia model and the refinement process (http://rimel.loria.fr and http://visidia.labri.fr)
- Teaching why and how distributed algorithms are working: the parachutist process

Next

- Transformation of Event-B models into *programming*, Dynamic networks
- Probabilistic assumptions
- Atlas of *correct-by-construction* distributed algorithms
- Hybrid modelling for CPS

Case studies

Case studies

- Sequential algorithms using the iterative pattern

Case studies

- Sequential algorithms using the iterative pattern
- Recursive algorithms using the recursive pattern

Case studies

- Sequential algorithms using the iterative pattern
- Recursive algorithms using the recursive pattern
- Distributed algorithms: protocols, leader election, self-stability, spanning tree, snapshot, mutual exclusion, cryptographic protocols, ...

Case studies

- Sequential algorithms using the iterative pattern
- Recursive algorithms using the recursive pattern
- Distributed algorithms: protocols, leader election, self-stability, spanning tree, snapshot, mutual exclusion, cryptographic protocols, ...
- *C. C. Marquezan and L. Z. Granville. Self-* and P2P for Network Management - Design Principles and Case Studies. Springer Briefs in Computer Science. Springer, 2012.*

Case studies

- Sequential algorithms using the iterative pattern
- Recursive algorithms using the recursive pattern
- Distributed algorithms: protocols, leader election, self-stability, spanning tree, snapshot, mutual exclusion, cryptographic protocols, ...
- *C. C. Marquezan and L. Z. Granville. Self-* and P2P for Network Management - Design Principles and Case Studies. Springer Briefs in Computer Science. Springer, 2012.*
- Distributed algorithms in the local computation model with LABRI in Visidia: naming, spanning, election ... (visidia.labri.fr)

Case studies

- Sequential algorithms using the iterative pattern
- Recursive algorithms using the recursive pattern
- Distributed algorithms: protocols, leader election, self-stability, spanning tree, snapshot, mutual exclusion, cryptographic protocols, ...
- *C. C. Marquezan and L. Z. Granville. Self-* and P2P for Network Management - Design Principles and Case Studies. Springer Briefs in Computer Science. Springer, 2012.*
- Distributed algorithms in the local computation model with LABRI in Visidia: naming, spanning, election ... (visidia.labri.fr)
- http://eb2all.loria.fr

Publications for the talk

- Dominique Méry: Refinement-Based Guidelines for Algorithmic Systems. Int. J. Software and Informatics 3(2-3): 197-239 (2009)
- Nazim Benassa, Dominique Méry: Cryptographic Protocols Analysis in Event B. Ershov Memorial Conference 2009: 282-293
- Nazim Benassa, Dominique Méry: Proof-Based Design of Security Protocols. CSR 2010: 25-36
- Dominique Méry, Neeraj Kumar Singh: A generic framework: from modeling to code. ISSE 7(4): 227-235 (2011)
- Dominique Méry, Neeraj Kumar Singh: Formal Specification of Medical Systems by Proof-Based Refinement. ACM Trans. Embedded Comput. Syst. 12(1): 15:1-15:25 (2013)
- Manamiary Bruno Andriamiarina, Dominique Méry, Neeraj Kumar Singh: Revisiting snapshot algorithms by refinement-based techniques. Comput. Sci. Inf. Syst. 11(1): 251-270 (2014)
- Manamiary Bruno Andriamiarina, Dominique Méry, Neeraj Kumar Singh: Analysis of Self- and P2P Systems Using Refinement. ABZ 2014: 117-123
- Yamine At Ameur, Dominique Méry: Making explicit domain knowledge in formal system development. Sci. Comput. Program. 121: 100-127 (2016)
- Dominique Méry: Playing with state-based models for designing better algorithms. Future Generation Comp. Syst. 68: 445-455 (2017)
- Dominique Méry, Michael Poppleton: Towards an integrated formal method for verification of liveness properties in distributed systems: with application to population protocols. Software and System Modeling 16(4): 1083-1115 (2017)
- Dominique Méry: Modelling by Patterns for Correct-by-Construction Process. ISoLA (1) 2018: 399-423


Questions
???