Formal Software Engineering

GL Formel

J. Christian Attiogbé

Master Alma, Novembre 2018

Plan

About me: Teaching at IUT & UFR Sciences - Research at LS2N

Dpt Info IUT

- Modélisation de données,
- Bases de données,
- Algèbre linéaire
- Modélisation et programmation de systèmes répartis

Dpt Info UFR Sciences

- Formal Software Engineering (*Construction formelle de logiciels*)

LS2N- UMR 6004 / Université de Nantes - CNRS - ECN - IMTA

AeLoS Team LS2N

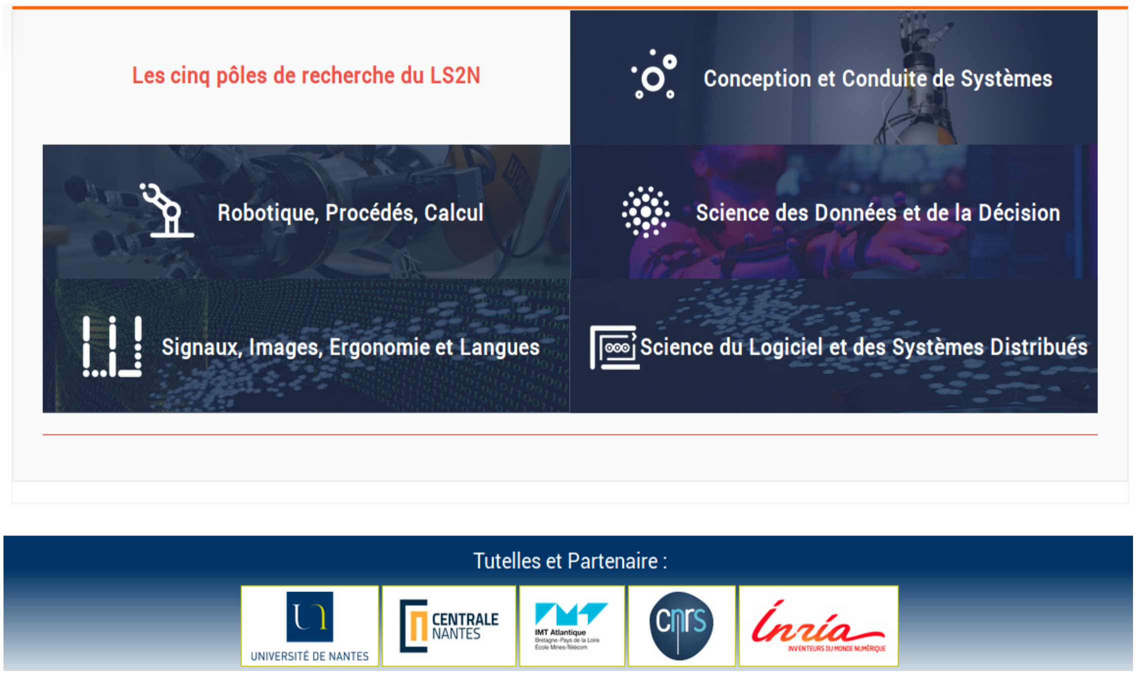

Les cinq pôles de recherche du LS2N
Conception et Conduite de Systèmes
Robotique, Procédés, Calcul
Science des Données et de la Décision
Signaux, Images, Ergonomie et Langues
Science du Logiciel et des Systèmes Distribués
Tutelles et Partenaire :
UNIVERSITÉ DE NANTES
CENTRALE NANTES
IMT Atlantique
cnrs
inria

Figure: Pôles de recherche du LS2N

AeLoS Team - LS2N

Research Topics

- Construction of Correct Architecture and Software
- Modelling, Verification, Refinement, Semantics
- Distributed Systems (services, components, architectures, properties)

Contact the team members for various internship projects, PhD projects, ...

AeLoS Team Members

P. André, G. Ardourel, C. Attiogbé, B. Delahaye, A. Lanoix, M. Oussalah, J. Rocheteau, + PhD Students + Postdocs + internship students (you can join us!)

Presentation of this Course (24h)

Formal modelling and verification of software
(the only way amenable to prove software correctness)

- Part 1 - by Claude Jard (˜ 12 hours)
  Concurrency and Semantic Models ;
  Petri Nets/Romeo (model-checking), Timed Models/Uppaal (model-checking).
- Part 2 - by Christian Attiogbé (˜ 12 hours)
  Correct Construction with B Method, Event-B
  Atelier B/Rodin (theorem-proving).

Presentation of this Course (24h)

Forecast Agenda

Dates	Part 1 C. Jard	Part 2 C. Attiogbé
31/10	CJ	
07/11	CJ	
14/11	CJ	
21/11		CA
28/11		CA
05/12		CA

About you - Motivations for this course

MASTER level ⇒ Managing industrial projects (computer systems)
in various domains,
with variable size (small $\cdots$ big)
Complex CS projects ⇒ Methods, Techniques, Tools

- Analysis Methods,
- Design & Verification Methods,
- Development/Implementation Methods.

You probably already know some programming languages, semi-formal methods, [FM?]

Are you confortable with large CS projects, difficult problems, (what about the future)... ?

12h! - Rythm

Figure: Drumatic! (big-drum-purdue)

Example: Web services interoperability (WS-AT)

Interoperability of services in distributed applications.
In distributed applications several services cooperate to achieve common goals.
Pbm: How to build such interoperable, distributed applications with coordinated joint works? in an asynchronous context.
Web services tie together a large number of participants (they are services) forming large distributed computational units called activities. These activities are complex due to many parameters: interaction between participants, they can take long time...
To face the complexity, a framework to coordinate the activities is needed (it is the objective of WSCOOR, oasis). It enables participants to reach a consistent agreement on the outcome of distributed activities.
Several protocols have been proposed as basis for the interaction between Web services.
For example WS-Atomic Transaction (WS AT) contains protocols which are mechanisms to create activities, join into them, and reach common agreement on the outcome of joint operations.

Example: Web services interoperability (WS-AT)

Def: An activity is a set of actions spaning multiple services but with a common goal (classical ex: resa).
The activities that require the ACID (atomic, consistent, isolated, and durable) properties of transactions are users of WS-Atomic Transaction.
An initiator creates/initiates an activity, and communicates its context to other applications. The other applications can register to participate in the activity. A coordinator manages all the participants of an activity. The coordinator at some point can decide to abort or to try to commit the transaction. Therefore it initiates (preparation phase) a vote to which all the participants participate. When there is a common positive agreement, it can commit the outcome (commit phase) of the trasaction (all or nothing).

Required Safety Property: to guarantee that the initiator and the participants agree on whether the transaction is committed or aborted.

http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os/wstx-wsat-1.1-spec-os.html

Example 1: a formal specification

```
MACHINE /* Sorting: a set of naturals -> seq. of natural */
  Tri
CONSTANTS
  tride /* defining a function */
PROPERTIES
  tride : FIN(NAT) ---> seq(NAT) &
   (ran(tride(ss)) = ss &
     %(ii,jj).(ii : dom(tride(ss)) & jj : dom(tride(ss)) &
        ii < jj =>
           (tride(ss))(ii) < (tride(ss))(jj) ) ) )
END
```

Emphasize abstraction = what (not how)

Example 2: a formal specification

```
system ProdCons /* Model */
sets
   DATA  ;
   STATE = {empty, full}
variables
   buffer, bufferstate, bufferc
invariant
   bufferstate ∈ STATE
∧ buffer ∈ DATA ∧ bufferc ∈ DATA
initialization
   bufferstate := empty
||  buffer :∈ DATA
||  bufferc :∈ DATA
end
```

Emphasize abstraction

Example 2: (continued)

ProdCons (continued)...

```
events
   produce ≙ /* when buffer empty */
      any dd where
         dd ∈ DATA ∧ bufferstate = empty
      then
         buffer := dd ||
         bufferstate := full
      end ;
   consume ≙ /* when buffer is full */
      select   bufferstate = full
      then
           bufferc := buffer ||
           bufferstate := empty
      end
end
```

Emphasize abstraction = what (not how)

Examples of properties

Always an unique process in CS	$card(activeProc) = 1$
A process cannot be simultaneously active and blocked	$activeProc \cap clockedProc = \emptyset$
...	...

☞ The use of invariant properties

- Safety properties: *nothing bad should happen*
- Liveness properties: *something good eventually happens*
  More generally, one uses Modal Logics.

Categories/Natures of Software Systems

Nature of software systems	what Features?	which Methods?
sequential		
autonomous (transformational)		
centralised		
reactive		
real-time		
parallel		
parallel and concurrent		
distributed		
embedded		
communication protocols ...		

$\Rightarrow$ various types of software systems, various methods

Semi-Formal Methods

Examples of semi-formal methods

- Functional Analysis (SA..., SADT),
- Structured Analysis (SA, SSADM), SA-RT,
- Entity-Relationship (*Entités/Associations*): Merise, Axiale,
- JSD/JSP,
- Object-Oriented Analysis, OMT, UML, Objectory Process (Ericsson, 1987), rational unified process (RUP),
- Software Architecture (System Level ; Top-Down approach),
- etc

Pros and Cons

Need of Formal Methods

Need of rigorous methods for some specific domains:

- Security, Certification, Cost, Maintenance
- ITSEC (Information Technology Security Evaluation Criteria) requires the use of **formal methods**
- Failure of (one flight) of ARIANE!, failure of a Pentium series, etc
- Environments which are dangerous for humans (nuclear, chemistry, marine, etc)
- Embedded Systems (vehicles, home equipments, etc)
- Automata (medical domain, etc)
- etc

Pros and cons

Industry [already] adopts FM!

Difficulties for industries: Market Pressure, High costs, · · ·
BUT, there are numerous success stories

- Proof of a C compiler (Coq, Xavier Leroy, 2011) !!!
- Design of a Real-Time Operating System (TLA+)
  E. Verhulst, R.T. Boute, J.M. Sampaio Faria, B.H.C Sputh, V. Mezhuyev, Formal Development of a Network-Centric RTOS, 2011
- Airbus (Astrée, Scade/Simulink), Aerospace
- NASA (PVS, shuttle, ...), Boeing (...)
- Proof of IEEE 1395 Firewire Protocols (Spin, PVS, B, +++ ; 2004+)
- Proof of control systems (B, Siemens)
- Proof of circuit (STMicroelectronics)
- · · ·

Industry [already] adopts FM!

- · · ·
- BOS barrier protecting the harbor of Rotterdam (Z, 2001)
- Proof of microcode and software (Intel)
- Proof of Communication Protocols (IO Automata, 1993+)
- · · ·

The complexity of current computer systems discourages empirical methods.

Why formal methods?

- Some systems need to be correct (=the right behaviours, no bugs)

Examples

Health, Medecine, avionics, transportation, security, army/defense, home automation, wide distributed systems, embedded systems, etc
$\Rightarrow$ all critical systems or complex systems

- Fomal methods are part of a solution
  - Model the system, using mathematics (= formal notations)
  - Reason on the system *à priori*, using mathematics (=rigorous reasoning).
  - to MASTER complexity

Introduction: Prove the correction of a software

Build correctly a software or
Prove the correction of a software S via its model.

- The *model* of the software : M
- The *properties* : P
- $M \models P$
  proof depending on the structure of the model
  ex: prove that P is true in all the (reachable) states of M
  (if M is a state model)

Anyway, you need a formal model; and/or rigorous software dev. methods.
Do you know some?

☞ Learn how to build M, P and how to prove (Modelling + Verification) (using dedicated tools or not).

What to learn?

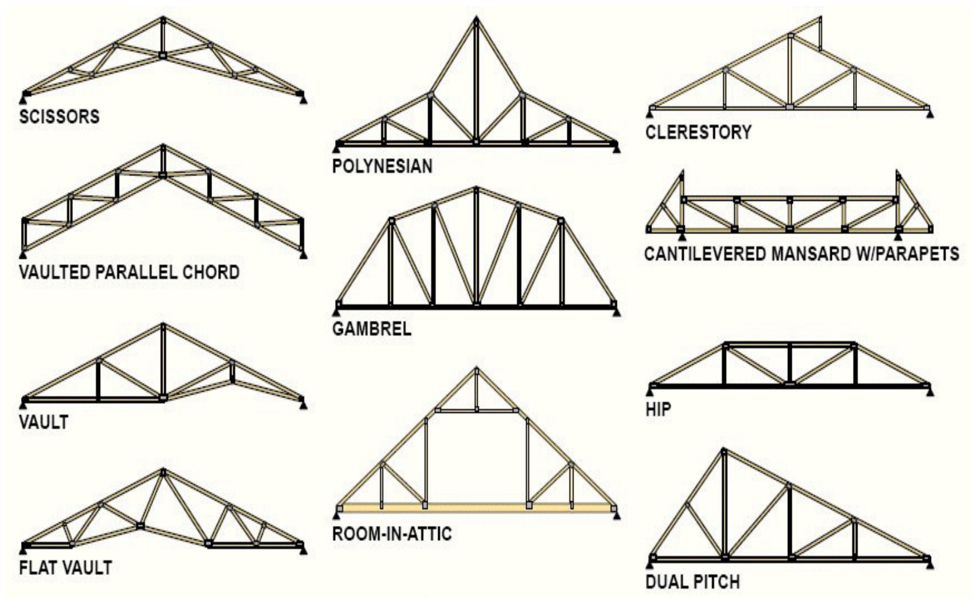


Figure: Truss styles (from caudilltrussandmetal.com)

Overview of formal methods and approaches

- Deductive approaches (logic-based)
  Build a logic model of the system
  Prove the properties of a software/system, from the stated logical specifications = demonstrate a theorem from axioms; theorem proving
  Trends: Correct-by-construction (model, prove, refine until code)
- State exploration approaches (automata-based)
  Build a state-based model of the system
  Check some properties in all the possible states of the software/systems
  model-checking; State-explosion;
  Symbolic model-checking; Statistic mdel-checking; ...
- Static analysis (à posteriori, on the code/abstraction) with abstract interpretation, ...

Examples of models (you already know)

- Logics models (First Order, Higher Order, Modal)
- Axiomatic/algebraic models (equation systems)
- State-based models (Automata, LTS, graphs)
  Finite State Machines (Mealy, Moore, ...), Petri nets, Communicating processes,...
  + various aspects: time, data, signals, probabilities

Various classes of models and systems

- Data-intensive models
- Synchronous models - Asynchronous models
- Timed Systems - Probabilistic (extension of Tansition systems)
- Reactive systems, Embedded systems
- ...

☞ We will learn some aspects in this course.

Features of Formal Methods

Formal methods ⇒ use rigorous approach to

- guaranty of software correction with respect to specifications,
- decrease/remove errors, and disfunctionning,
- make it easy the maintenance and the evolution.

Methods in Engineering

Construction methods of computer systems

A few analogies:

Building Engineering (*Génie civil*)

→ Architecture, schemes/blueprints (design), computings, construction (implementation)

Physics

→ Observations, modelling/study of models, implementation

Computer Science (Informatique)

Requirement Analysis (observations?)

Modelling - study of models,

Design and Implementation of systems.

Preliminaries

Various **approaches of formal methods**:

- à postériori : First, one implements (programming paradigm) and then one verifies that the produced program is correct
  → proof systems, testing, model-checking
- à priori : One builds correctly the system
  → Development methods (refinement, synthesis), proof systems

Several formal methods (languages, proof systems, methods)

Preliminaries

- Top-down approach: by decomposition
    - Global analysis (system study, system engineering)
    - Software Architecture

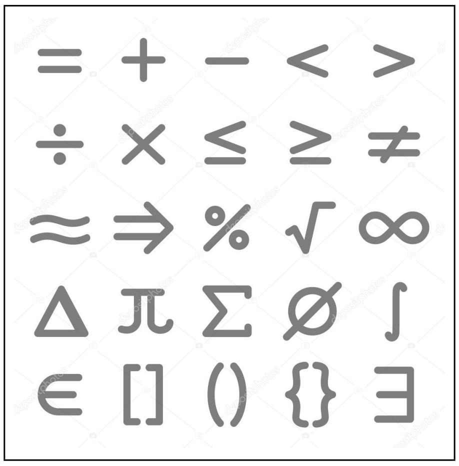
      ↓
    - Implementation of components
        - Direct Programming or
        - Formal Development
- Bottom-up approach: composition of elementary components.
    - Study of available components,
    - Composition, reuse.

In all cases (approaches), make use of formal methods for

- Study of systems
- Study and construction of components/software
- Formal framework for reasoning, analysis, development.

What are inside formal methods?

- Logics
- Algebra
- Discrete Mathematics
- Set Theory
- Automata Theory
- Type Theory
- Refinement Theory
- ...



Examples of a few industrial applications

with the B Method (J-R. Abrial)
GEC ALSTHOM, SNCF and MATRA Transport (now Siemens)

- Railway Speed Control System (KVS for SNCF)
- Line A of the Paris RER - SACEM (signaling, speed control)
- Calcutta Metro (CTDC)
- Montreal Metro (CTDC), Marseille, Bel horizonte
- Météor (line 14, of Paris Metro, without human driver)
- Landing doors (*portes pallières*) in Metro stations
- Old people insurance, in French Sécurité Sociale
- CICS of IBM (major restructuring of a transactional, about 800000 lines of code)
- B and VDM are used in financial domain softwares, BULL UK

Many other systems with PVS, Coq, SPIN, Petri Nets, Lotos, etc

Example of the Railway Speed Control System (Metro)

- Data acquisition (sensors, converters, etc),
- Computation/decision,
- Orders sent to physical devices (speed slowing system, braking system),
- Embedding of the software in the global system of the train.

Other used approaches, along the time

Some of them are equipped with tools and adopted by industries

- LOTOS, SDL (european Standard)
  algebric approach + *communicating processes*
- Isabelle (Germany), PVS (USA)
- MEC/AltaRica (Université de Bordeaux + industries)
- Classical Logics: First order Logic, Hoare Logic, etc
  (Why, Frama-C, Krakatoa (Java), Key,...)
- Non-classical Logics, modal logics
- Coq, High-Order Logics, type theory

Foundation of formal approaches (proof)

Interpretation of the Curry-Howard's Isomorphism:

$$\boxed{\text{Proof} \left| \frac{\mathit{Axioms}}{\mathit{Theorems}} \right.} \quad \text{equiv. to} \quad \boxed{\text{Development} \left| \frac{\mathit{Specifications}}{\mathit{Programs}} \right.}$$

♡♡♡

→ Proof assistants needed! not only editors and compilers

Overview of Software Construction

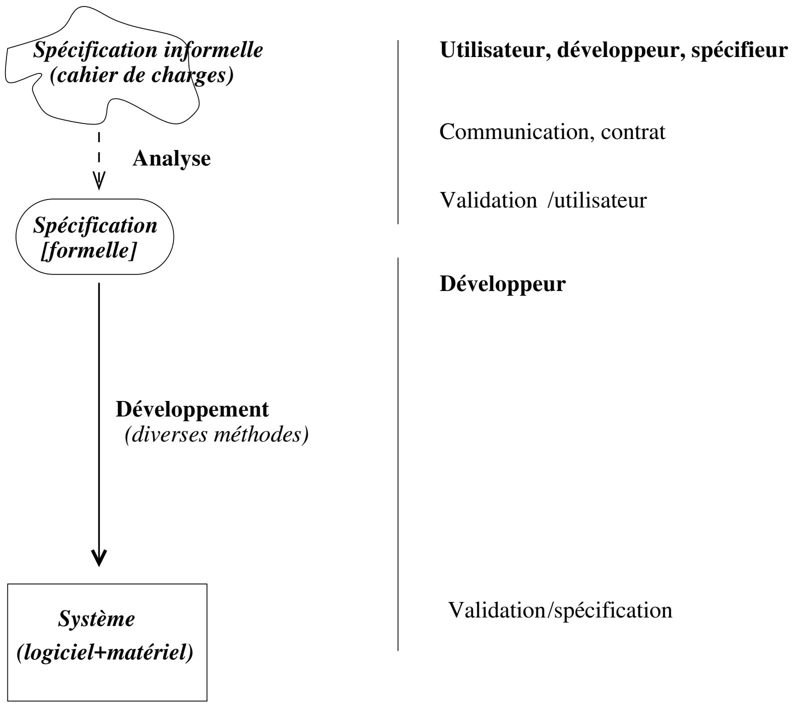


Figure: Issue of system development de system

Overview of Software Construction

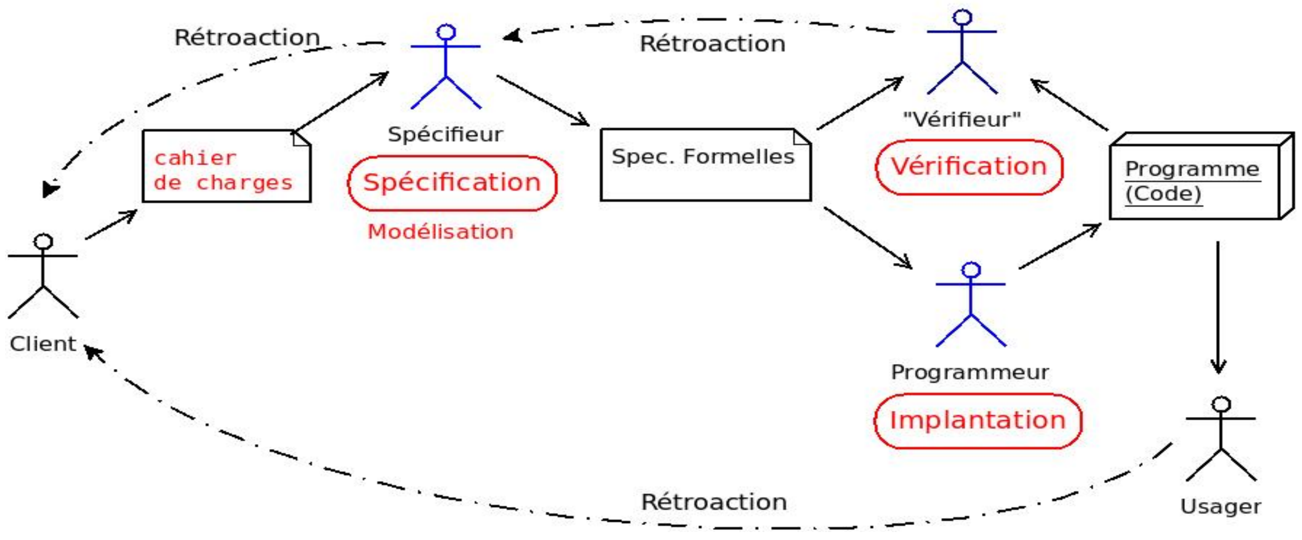


Figure: A life cycle of formal software construction

Use of Formal Methods

→ Not always approriate:

- A hammer to kill a fly
  - depending on the needs
- Professional environment
  - Available experiments?
- Industrial context
  - Delay, costs, productivity
- Certification
  - Requirements of clients.

Figure: Heavy structure (from clevelandbridge.com)

Which approach to use?

→ Several parameters:

- Designer/implementor of big systems,
- Designer/implementor of small (home) systems,
- Features of systems to be implemented,
- Available experiments,
- ...

Categories/Natures of Software Systems

Nature of software/systems	what Features?	which Methods?
sequential		
autonomous (transformational)		
centralised		
reactive		
real-time		
parallel		
parallel and concurrent		
distributed		
embedded		
communication protocols . . .		

$\Rightarrow$ various types of software systems, various methods

A few difficult points

- To describe precisely the intended system specification
- To build correctly the sofware development
- To be sure that the constructed software is correct with respect to the needs
- Maintenance/Evolution of the system.

A few difficult points

- To describe precisely the intended system specification
- To build correctly the sofware development
- To be sure that the constructed software is correct with respect to the needs
- Maintenance/Evolution of the system.

Each project is unique

- Nature of complex systems → multifacets, modular
- Several methods, including:
  - Semi-formal methods
  - Formal Methods (integrated) → to deal with complex systems.

⇒ mastering several methods

A few definitions

Modelling:
Hoare: A scientific theory is formalised as a mathematical model of reality, from which can be deduced or calculated the observable properties and of a well-defined class of processes in the physical world.
There are two main notions of models in computer science.

1. Model = an approximation of the reality by a mathematical structure.
   An object O is a model of a reality R, if O allows one to answer all the questions about R.

In Mathematics, Physics, ... models are built with equation systems using quantities (masses, energy, ...) or hypothetic laws.
⇒ State exploration, simulation

A few definitions (continued)

2. Logics, theory of models
   A model of a theory T is a structure in which the axioms of T are valid.
   A *structure* S is a model of a theory T, or S satisfies T if all formula of T is satisfied in S.
   The reality is a model of a theory!
   First Order Theory = any set of logic formula in a given language (precisely defined).
   Model as an interpretation of a specification - an algebra as a model of an algebraic specification (or an axiomatisation).

⇒ deductive approach, theorem proving

A few definitions (continued)

These two notions of *model* are encountered in the model-oriented (or state-oriented) and property-oriented approaches of Soft. Eng.
In current use,

- *model* = (archetype), what serves or is used for imitation to reproduce orther instances.
- *model* = (paradigm), declination model, conjugation model, etc
- *model* = (reference), . . .

A few definitions (continued)

Semiformal Method =

- Graphical Language [+ formal]
  (precise syntax and unprecise semantics) and
- Various analysis tools.

$\longrightarrow$ Combination of languages/methods/techniques that do not all have a precise semantics.
Examples : JSD, OMT, OOX, UML, Unified Process, RUP

Interest and Limitations of Semi-formal Methods

- SADT, SA-RT, SSADM, ...
- JSD-JSP,
- Merise, Axial, ...
- OOA, OMT, UML, Unified Process, RUP
- ...

The problem analysis is performed.
It is a positive contribution, although insufficient.
But, the problem is sided.
$\longrightarrow$ impossible to reason formally on the intended system.
$\longrightarrow$ there can be ambiguities and errors.

A few definitions (continued)

Formal Method =

- Formal Language (precise syntax and precise semantics) and
- Proof or formal reasoning system.

Examples: FSM, Petri, Z, CCS, CSP, HOL, Coq, PVS, B, ...

Formal Development =

- systematic transformation of specifications into programs using predefined laws/rules.
  Synthesis, Refinement
  Need Provers/assistants : Isabelle, Why, Coq, ...

Examples: B Method, Perfect, Escher C,...

A few definitions (continued)

Verification: to show that a system (S) is correct with respect to some properties (P)

$$S \models P$$

Validation: to show that a system (S) is correct with respect to some informal properties (the needs)

$$S \sim S_{informal}$$

Formal reasoning : Consists in **applying a formal system to a specification**.

Examples of theory

Set theory: it is based on a set of axioms (Bourbaki, Cantor, Zermelo, ...).
The objects of this theory are called sets.
The classe of the sets is called the universe.
The axioms of the set theory (of Zermelo+Fraenkel) are the following:

Examples of theory (continued)

- Axiom of the empty set: *there exists a set which does not contain any element: it is the empty set.*
- Extensionality Axiom: *two sets are equal if and only if they contain exactly the same elements.*
- Union Axiom: *the union of sets is a set.*
- Axiom of the set of parts: *given a set E, there exists a set P suchat that a set F is member of P if F is a part of E.*
- Axiom of replacing/substitution schema(Fraenkel, 1922) : *When one defines a function with the formula of the set theory, the elements for which this function verifies a given property are also a set.*

Moreover, to these axioms is added, the axiom of infinite: *there exists an infinte ordinal.*
ZFC = ZF + axiom of choice

- Axiom of choice: *Given a family of disjoint sets, if we consider one element of each set of the family, then one builds another set.*

Some references

- Jan Van Leeuwen, *Handbooks of Formal Models and Semantics*, 1990
- J. Wing, *A Case Study in Model Checking Software Systems*, SCP, 1997
- Mana & Pnueli; de Roever et Al.;
- E. Clarke, J. Wing, *Formal Methods: State of the Art and Future Directions*, CMU, 2006
- L. Lamport, numerous documents!
- André Arnod, *sémantique des processus communicants*
- J-F. Monin, *Introduction aux méthodes formelles*. Hermès, 2000
- *Success Stories*
  `www.fm4industry.org/index.php/DEPLOY_Success_Stories`,
  `www.fm4industry.org/index.php/Deploying_Event-B_in_an_Industrial_Microprocessor_Development`
- and Dijkstra, Hoare,...

Wing, Hehner, Monin, Holloway, $\cdots$

Quick itroduction with formal modelling examples.

Logics: Modelling and reasoning

10 min.

Structure of the brain: logical part

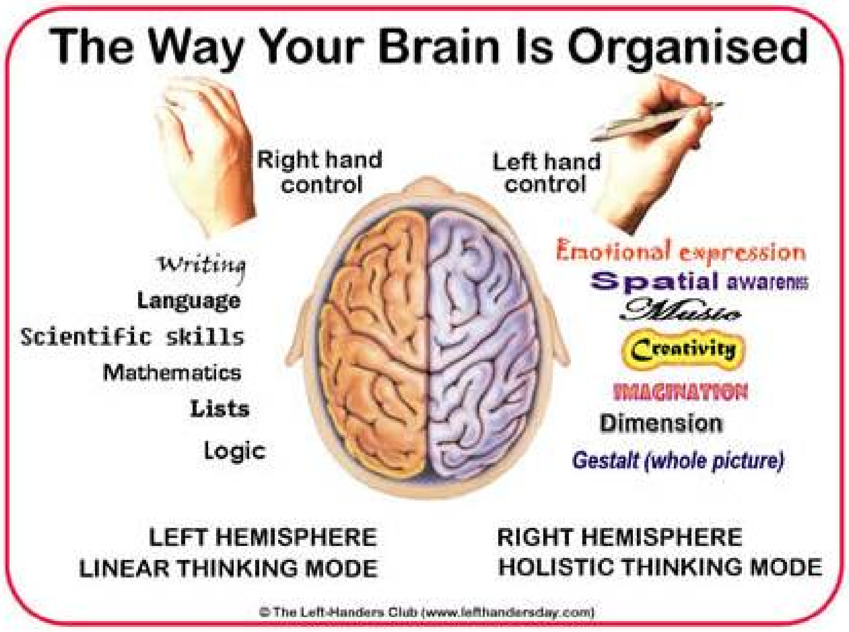


Figure: Brain organisation (from www.mindfulnet.org/page8.htm)

First Order Logic

A proposition is a sentence named P, Q, E... with a value TRUE or FALSE; the construction of a proposition is made with the following grammar:

$$\begin{array}{lll} prop & ::= & P, Q, E, \ldots \\ & | & prop \wedge prop \\ & | & \neg\ prop \\ & | & prop \Rightarrow prop \end{array}$$

Parentheses can be used if necessary.
Other operators (logical connectors) : $\vee$, $\equiv$
The semantics of a proposition (with the connectors) is given by a truth table (Exercice).

Examples of Proposition

A cat with a hat is a Lion
Peter rides a bycicle
$0 > 3$

Predicates

Propositional calculus deals with : absolute truth.
Predicate calculus deals with : relative truth,
it is an extension of propositional calculus.

$$x > 2$$

$$x \in I\!N \Rightarrow x \geq 0$$

Two kinds of variables are used in predicates: free variables and bound variables which are introduced with quantifiers.

How to use predicates

- Substitution

$$[x := 5](x \in I\!N \Rightarrow x \geq 0)$$

$$(5 \in I\!N \Rightarrow 5 \geq 0)$$

$$[x := elephant](BigEars(x) \Rightarrow African(x))$$

- Quantification

$$\forall x.BigEars(x) \Longrightarrow African(x),$$

$$\forall x.(Animal(x) \wedge BigEars(x)) \Longrightarrow African(x)$$

Construction of predicates

$$
\begin{array}{lcl}
Predicat & ::= & Predicat \Rightarrow Predicat \\
 & | & Predicat \wedge Predicat \\
 & | & \neg\ Predicat \\
 & | & \forall\, Variable.Predicat \\
 & | & [\,Variable := Expression]Predicat \\
 & | & Expression = Expression \\
Expression & ::= & Variable \\
 & | & [\,Variable\ := \ Expression]Expression \\
Variable & ::= & Identifier
\end{array}
$$

Usage of Logics

- for modelling : *predicates*

  predicate = formula to be proved

$$P \ \wedge \ Q$$

$$P \ \Rightarrow \ Q$$

$$0 \ < 3$$

$$\{0,3\} \subset \{0,4,8,3\}$$

- for reasoning : *sequents*

$$H \vdash P$$

$$\left.\begin{array}{r} H : Hypotheses \\ P : conjecture \end{array}\right\} predicates$$

Inference rules of propositional calculus

$\wedge\ intr$	$\dfrac{HYP \vdash P \qquad HYP \vdash Q}{HYP \vdash P \wedge Q}$	use backward to decompose into simple subgoals with the same hypotheses
$\wedge\ elim$	$\dfrac{HYP \vdash P \wedge Q}{HYP \vdash P \qquad HYP \vdash Q}$	
$\Rightarrow\ intr$	$\dfrac{HYP, P \vdash Q}{HYP \vdash P \Rightarrow Q}$	deduction rule
$\Rightarrow\ elim$	$\dfrac{HYP \vdash P \Rightarrow Q}{HYP, P \vdash Q}$	anti-deduction

Modus Ponens	$\dfrac{HYP \vdash P \qquad HYP \vdash P \Rightarrow Q}{HYP \vdash Q}$	
Contradiction	$\dfrac{HYP, \neg Q \vdash P \qquad HYP, \neg Q \vdash \neg P}{HYP \vdash Q}$	first rule for ¬
	$\dfrac{HYP, Q \vdash P \qquad HYP, Q \vdash \neg P}{HYP \vdash \neg Q}$	second rule for ¬

Reasoning

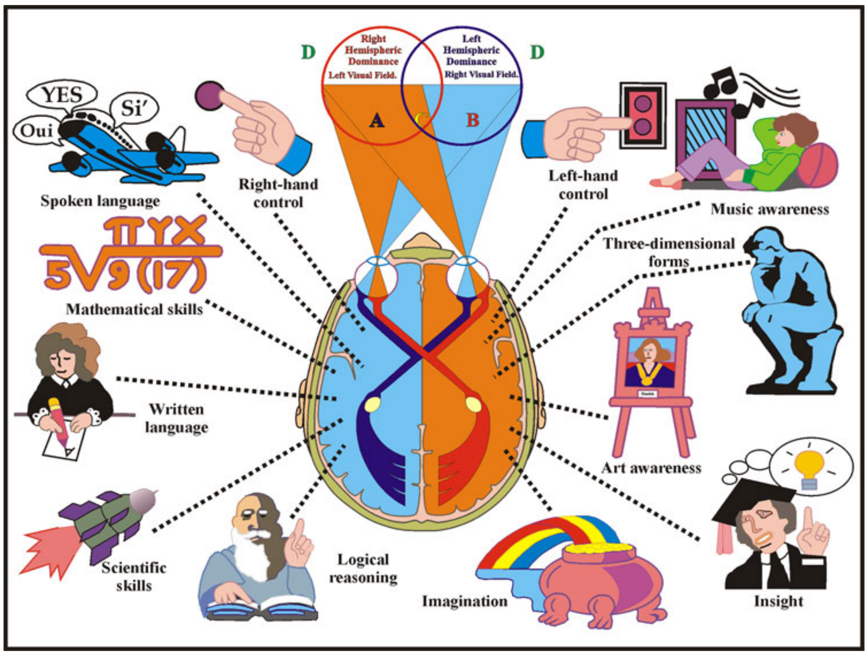


Figure: To mimic brain complexity (from ehealing.us)

Reasoning

with a meta-language

- Inference rules

  An inference rule links sequents and its defines a valid step of a proof.
  An inference rule has the following shape:

$$\frac{\sum_1, \sum_2, \ldots, \sum_n}{\sum}$$

  The sequents $\sum_1, \sum_2, \ldots, \sum_n$ are called *antecedents*, and the sequent $\sum$ is called *consequent*.

Reasoning (continued)

- Proof principle
  To prove a sequent, one uses the inference rules
  - as derivation rules : forward rule application,
  - as reduction rules : backward rule application.

Implementation

- Theorem to prove / Inference
  To prove a theorem

$$P \vdash Q$$

one transforms it into inference rule

$$\frac{H \vdash P}{H \vdash Q}$$

- Proofs : forward or backward - tactics

SETS: Modelling and reasoning

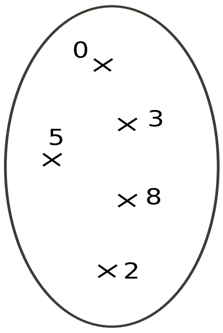
30 min.

Structuring

Figure: Amazing steel structure (from clsteel)

Sets and typing

- Predefined Sets (work as types)
  BOOL, CHAR,
  INTEGER ($\mathbb{Z}$), NAT ($\mathbb{N}$), NAT1 ($\mathbb{N}^*$) ,
  STRING



- Cartesian Product E × F

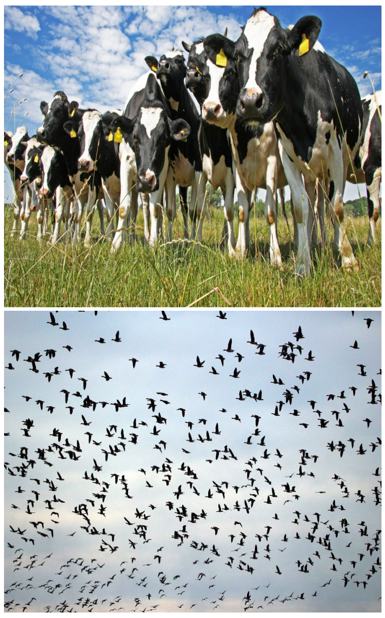


Figure: Sets of cows, birds

Sets and typing

- The set of subsets (powerset) of E $\mathcal{P}(E)$ written POW(E)
- user defined
  abstract sets,
  enumerated sets

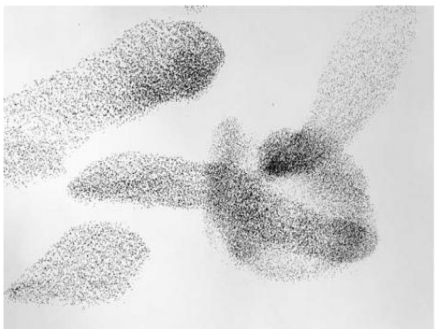


Figure: Set of sets of birds

Set Theory Language

The standard set operators

E, F and T are sets, x a member of F

Description	Notation	Ascii
union	$E \cup F$	`E \/ F`
intersection	$E \cap F$	`E /\ F`
membership	$x \in F$	`x : F`
difference	$E \setminus F$	`E - F`
inclusion	$E \subseteq F$	`E <: F`

+ generalised Union and intersection
+ quantified Union et intersection

Set Theory Language

In ascii notation, the negation is written with /.

Description	Notation	Ascii
not member	$x \notin F$	`x /: F`
non inclusion	$E \not\subseteq F$	`E /<: F`
non equality	$E \neq F$	`E /= F`

Generalised Union (à la B)

an operator to achieve the generalised union of well-formed *set expressions*.

$S \in \mathcal{P}(\mathcal{P}(T))$

$\Rightarrow$

$union(S) = \{x \mid x \in T \wedge \exists\, u.(u \in S \wedge x \in u)\}$

Example

$$union(\{\{aa, ee, ff\}, \{bb, cc, gg\}, \{dd, ee, uu, cc\}\})$$
$$= \{aa, ee, ff, bb, cc, gg, dd, uu\}$$

Quantified Union

an operator to achieve the quantified union of well-formed *set expressions*.

$\forall\ x.(x \in S \Rightarrow E \subseteq T)$

$\Rightarrow$

$$\bigcup x.(x \in S \mid E) = \{y \mid y \in T \wedge \exists\, x.(x \in S \wedge y \in E)\}$$

Exemple

$$UNION(x).(x \in \{1,\ 2,\ 3\} \mid \{y \mid y \in NAT \wedge y = x * x\})$$

$$= \{1\} \cup \{4\} \cup \{9\} = \{1,\ 4,\ 9\}$$

Generalised Intersection (à la B)

an operator to achieve the generalised intersection of of well-formed *set expressions*.

$S \in \mathcal{P}(\mathcal{P}(T))$

$\Rightarrow$

$inter(S) = \{x \mid x \in T \wedge \forall\ u.(u \in S \Rightarrow x \in u)\}$

Example

$inter(\{\{aa,\ ee,\ ff,\ cc\},\ \{bb,\ cc,\ gg\}, \{dd,\ ee,\ uu,\ cc\}\} = \{cc\}$

Quantified Intersection (à la B)

an operator to achieve the quantified intersection of of well-formed *set expressions*.

$\forall\ x.(x \ \in\ S\ \Rightarrow\ E\ \subseteq\ T)$

$\ \ \Rightarrow$

$\bigcap\ x.(x\ \in\ S\ \mid\ E)$

$\quad = \{y\ \mid\ y\ \in\ T\ \wedge\ \forall\ x.(x\ \in\ S\ \Rightarrow\ y\ \in\ E)\}$

Example

$INTER(x).(x\ \in\ \{1,\ 2,\ 3,\ 4\}\ \mid \{y\ \mid\ y\ \in\ \{1,\ 2,\ 3,\ 4,\ 5\}\ \ \wedge\ y\ >\ x\})$

$$= inter(\{\{1,\ 2,\ 3,\ 4,\ 5\}, \{2,\ 3,\ 4,\ 5\},\ \{3, 4, 5\},\ \{4, 5\}\})$$

Relations

RELATIONS

Recurrence Relations

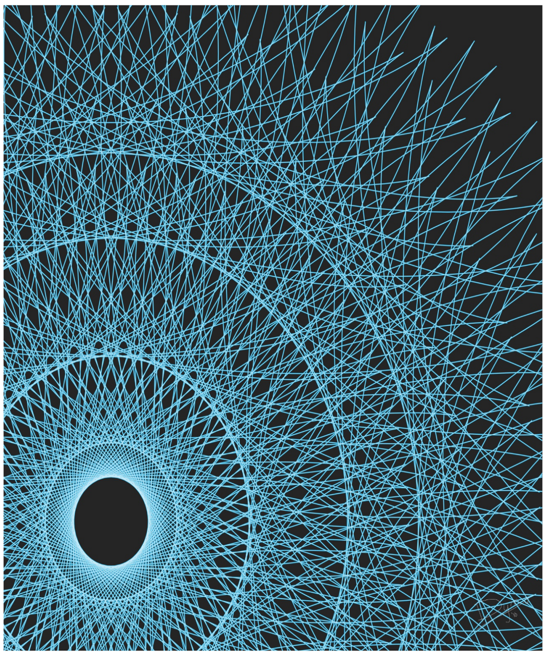


Figure: Amazing recurrence relation (from devanmatthews.files.wordpress.com)

Relations: definition, vocabulary

A relation r over D and A is a subset of the cartesian product $D \times A$
it is noted $r : D \leftrightarrow A$ or $r \subseteq D \times A$
r is a set of couples (d, a) also denoted by $d \mapsto a$

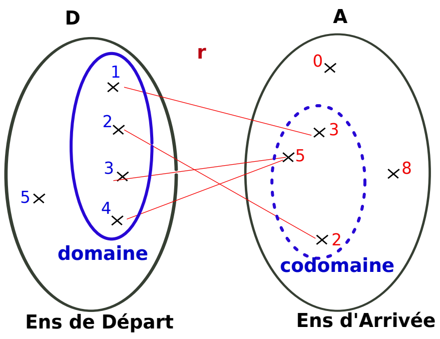


Figure: Euler-Venn diagram of r

$r = \{(1,3),(2,2),(3,5),(4,5)\}$ ou
$r = \{1 \mapsto 3, 2 \mapsto 2, 3 \mapsto 5, 4 \mapsto 4\}$

dom(r) = {1, 2, 3, 4}
ran(r) = {3, 5, 2}

Domaine : domaine Codomaine : range

Relations: definition, vocabulary

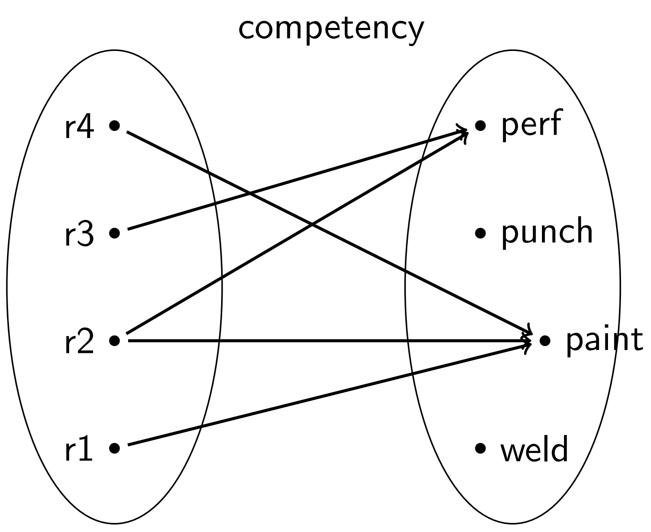
S and T are sets.
An element of $r : S \leftrightarrow T$ is a couple.
A element s of S can have several images in T.

Description	Notation	Ascii
relation	$r : S \leftrightarrow T$	r : S <-> T
domain	$dom(r) \subseteq S$	dom(r) <: S
range	$ran(r) \subseteq T$	ran(r) <: T
composition	$r;s$	r;s
composition r(s)	$r \circ s$	r(s)
identity	$id(S)$	id(S)

Relations (continued)

Robots

Consider a plant with a set of polyvalent articulated robots, for welding, painting, punching, perforating, etc. Ho to model? know the available painting robots? and how many? ...



Relations (continued)

Robots

Consider a plant with a set of polyvalent articulated robots, for welding, painting, punching, perforating, etc. A robot can be assigned at most only one task at time. But one task among its competencies

given abstract sets: BOT, $TASK$

$robots \subseteq BOT$

$tasks \subseteq TASK$

$$assignR : robots \rightarrow tasks$$

$$assignR \subseteq competency$$

Relations (continued)

to build new relation r' from $r : S \leftrightarrow T$

Description	Notation	Ascii
domain restrictition	$S \lhd r$	S <| r
range restriction	$r \rhd T$	r |> T
domain antirestriction	$S \lhd r$	S <<| r
range antirestriction	$r \rhd T$	r |>> T
inverse	$r^{\sim}$	r ~
relationnelle image	$r[S]$	r[S]
overiding	$r1 \oplus r2$	r1 <+ r2
direct product of rel.	$r1 \otimes r2$	r1 >< r2
closure	$closure(r)$	closure(r)
reflexive trans. closure	$closure1(r)$	closure1(r)

Functions

FUNCTIONS

Functions

S and T are sets. $f : S \mathrel{+\!\!\!\rightarrow} T$ a function
Unlike in a relation, an element of S can have at most one image via f.

example

Consider a set of tasks to be achieved by a set of robots.
A robot can be assigned at most only one task at time.

$robots \subseteq BOT$
$tasks \subseteq TASK$
$sched : robots \mathrel{+\!\!\!\rightarrow} tasks$ // partial function

What do we need if the requirements say *All tasks should be assigned* ?.

$sched : robots \mathrel{+\!\!\!\twoheadrightarrow} tasks$ // partial sujective function

Functions

S and T are sets. $f : S ⇸ T$ a function
Unlike in a relation, an element of S can have at most one image via f in T.

Description	Notation	Ascii
partial function	$f : S ⇸ T$	f : S +-> T
total function	$f : S \rightarrow T$	f : S --> T
partial injection	$f : S ⤔ T$	f : S >+-> T
total injection	$f : S \rightarrowtail T$	f : S >-> T
partial surjection	$f : S ⤀ T$	f : S +->> T
total surjection	$f : S \twoheadrightarrow T$	f : S -->> T
total bijection	$f : S ⤖ T$	f : S >->> T
lambda abstraction	$\%x.(P \mid E)$	

Powerful mathematical structures

$$\begin{tikzcd}
SETS\ (with\ \ \in, \times, \cup, \cap, ...) \\
Relations \arrow[u] \\
Functions \arrow[u] \\
Sequences \arrow[u]
\end{tikzcd}$$

Hoare Logic (Reminder?): Fundamentals of reasoning

Hoare Logic

10 min.

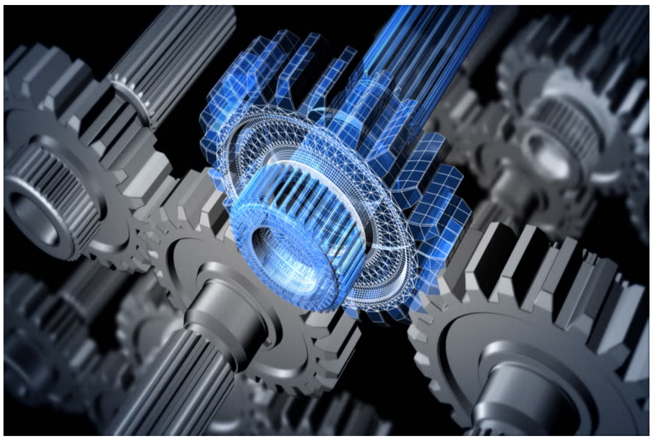


Figure: Basic engineering (from ak2.picdn.net)

Floyd-Hoare Logic

Elementary to reason on the correction of programs
Consider a pseudo-programming language, described by the grammar

$$
\begin{array}{lll}
E & ::= & N \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 \times E_2 \mid \cdots \\
\\
B & ::= & E_1 = E_2 \mid E_1 \leq E_2 \mid \cdots \\
\\
C & ::= & \texttt{SKIP} \\
 & \mid & V := E \\
 & \mid & C_1 \ ; \ C_2 \\
 & \mid & \texttt{IF } B \texttt{ THEN } C_1 \texttt{ ELSE } C_2 \\
 & \mid & \texttt{WHILE } B \texttt{ DO } C
\end{array}
$$

Floyd-Hoare Logic

The Hoare triple denotes the partial correction of a statement.

P a formula of the first order logic,
$\vdash P$ means P can be deducted from the laws of logics and arithmetics.

$\vdash \{P\}\ C\ \{Q\}$ means that $\{P\}\ C\ \{Q\}$ is

- either an instance of the schema of the axioms A1, A2 (above)
- or is deductible by a sequence of applications of the rules R_i.

Axioms and rules of Hoare logic

A1: Axiom of SKIP. For any formula P

$$\vdash \{P\}\ \texttt{SKIP}\ \{P\}$$

A2: Substitution Axiom. P a formula, V a programme variable, E nd expression

$$\vdash \{P[E/V]\}\quad V := E\quad \{P\}$$

($P[E/V]$ denotes the result of the substitution of E to the free occurrences of V in P.)

Axioms and rules of Hoare logic

R1: rule of the precondition (strengthening - renforcement)

$$\frac{\vdash P' \Rightarrow P \qquad \vdash \{P\}\ C\ \{Q\}}{\vdash \{P'\}\ C\ \{Q\}}$$

R2: rule of the postcondition (weakening - affaiblissement)

$$\frac{\vdash \{P\}\ C\ \{Q\} \qquad \vdash Q \Rightarrow Q'}{\vdash \{P\}\ C\ \{Q'\}}$$

Axioms and rules of Hoare logic

R3: rule of the sequence

$$\frac{\vdash \{P\}\ C_1\ \{Q\} \qquad \vdash \{Q\}\ C_2\ \{R\}}{\vdash \{P\}\ C_1\,;C_2\ \{R\}}$$

R4: rule of the `IF` structure

$$\frac{\vdash \{P \wedge B\}\ C_1\ \{Q\} \qquad \vdash \{P \wedge \neg\ B\}\ C_2\ \{Q\}}{\vdash \{P\}\ \ \texttt{IF}\ B\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \ \{Q\}}$$

R5: rule of the `WHILE` structure

$$\frac{\vdash \{P \wedge B\}\ C_1\ \{Q\}}{\vdash \{P\}\ \ \texttt{WHILE}\ B\ \texttt{DO}\ C_1\ \ \{Q \wedge \neg\ B\}}$$

System Properties (6 min.)

Figure: Protect your system! (from medievart.com)

Kripke Structure (Saul Kripke, 1960)

Directed graph used as internal representation of software. Each state has a name/label and a list of propositions in $\mathcal{P}$ which are valid in this state.

$$(S, S_{init}, S_{final}, \delta : S \leftrightarrow S, \sigma : S \rightarrow \mathcal{P})$$

$$\begin{tikzcd}
 & & \overset{3}{\underset{p}{\circ}} \arrow[dl] & & \\
\overset{1}{\underset{p}{\circ}} \arrow[r] & \overset{2}{\underset{p,q}{\circ}} \arrow[rr] & & \overset{4}{\underset{q}{\circ}} \arrow[ul] \arrow[r] & \overset{5}{\underset{q,s}{\circ}} \arrow[loop right]
\end{tikzcd}$$

Behaviour, Run, trace

Trace = sequence of states (their names) linked in the system.

$$s_1.s_2.\cdots.s_i.s_{i+1}.\cdots \mid (s_i, s_j) \in \delta$$

From a trace, we compute the sequence of set of valid propositions (possibly infinite!)

Safety/liveness - Safety

Safety property

Safety property expresses that "something bad must not happen"

Examples:

- The index values never over the bounds.
- Only one vehicle will be in the tunnel (critical section)
- The program never loose the requests

Predicate Logic(+ set theory): logic for specifying liveness properties

Safety/liveness - Liveness

Liveness property

Liveness property expresses that "something good must happen" (in the future runs)

Examples:

- The user will get her access after the attempts of connection.
- All requests will be treated before the server closing

Temporal Logic: logic for specifying properties over time
(Behavior of a finite-state system)

System Properties: Safety or Liveness

- The majority of properties are safety properties
- Liveness properties are often considered as more complicated safety properties (for instance with real-time response constraints)

Learn how to specify both; it depends on the project under work.

Linear-Time Logic (LTL) - Pnueli, during 1970

- Used to describe properties on individual execution traces (succession of dates)
- each moment in time has a well-defined successor moment. (function)
- Semantics: a set of (execution) traces

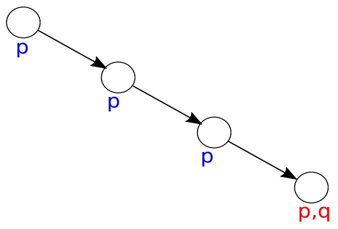


Figure: Pêche à la ligne (from madfred-angling.com)

Computation Tree Logic (CTL) - Branchng Time family

- Used to describe properties on several execution traces simultaneously (using quantifiers on the traces).
- from a state, reason about multiple possible time. (relations)
- Semantics: defined on terms of states

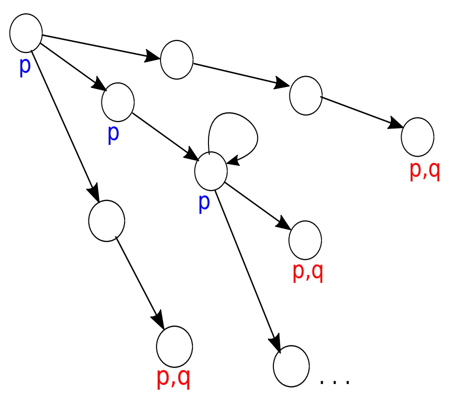


Figure: Pêche au filet (from esoxiste.com)

Tools and References

Model checking:
exponential in the size of LTL formula ; linear for TCL formula.
For both LTL, CTL, model checking is linear in the size of the state graph.

Some model checking tools:
SPIN, SMV, BLAST (Turing Award 2007: Clarke, Emerson, Siffakis)
BLAST, CADP, UPPAAL, PRISM, CBMC
UPPAAL-SMC, PLASMA, ...

Tools and References

- Hubert Garavel, *Formal Methods for Safe and Secure Computers Systems* `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/formal_methods_study_875/formal_methods_study_875.pdf?__blob=publicationFile`, 2013
- Jeannette M. Wing and Mandan Vaziri-Farahani*A Case Study in Model Checking Software Systems* https://www.cs.cmu.edu/afs/cs.cmu.edu/project/venari/www/scp.html
- Jeannette Wing, *Formal Methods: State of the Art and Future Directions*, https://www.cs.cmu.edu/afs/cs/project/calder/www/acm.html
- *Principles of Model Checking*, Christel Baier and Joost-Pieter Katoen. MIT Press, 2008.
- *Model Checking* Javier, Esparza, Stephan MERTZ `https://members.loria.fr/SMerz/talks/mc-tutorial.pdf`