Matière VAS
Vérification par Analyse Statique

Intervenants :
- Intervenants :
  Marc Pantel, Pierre-Loïc Garoche, Xavier Thirioux
- Cours : 9 séances
- Travaux Pratiques : 3 séances
- Bureau d'études : 2 séances

Contacts : bâtiment F, bureau F 305
(courriel `Prénom.Nom@enseeiht.fr`)

Contrôle des connaissances : 1 Bureau d'études en temps limité

Forme des supports : Fonctionnement CoViD

- Cours à distance : capsules vidéos et séances questions/réponses
- Travaux dirigés à distance pendant cours
- Travaux pratiques en présence

Plan

Validation, Verification, Certification, Qualification

Validation

Verification

Certification

Qualification

Validation, Verification, Certification, Qualification

Validation	System satisfies the user needs (make the right product)
Verification	System satisfies the requirements (make the product right)
Certification	
Qualification	

Validation, Verification, Certification, Qualification

Validation	
Verification	
Certification	System satisfies standard rules (usually public rules)
Qualification	Development tools satisfies standard rules

Software and System Engineering

- Process, Methods and Tools
  - Process : Workflow of activities allowing to build a product (software or system)
  - Method : How to conduct an activity
  - Tools : Partially automated support for the activity and its method
- Common activities: requirement analysis, architectural and component design, implementation, verification, validation, delivery, maintenance
- Common V & V : Testing (unit, integration, function, system, user, performance), Proofreading
- Common lifecycle models (high-level workflows): cascade, V, W, iteration
- Model Driven Engineering: Rely on the most appropriate domain specific methods and tools for each activity (instead of generic ones)

Software and System Engineering

- Process, Methods and Tools
  - Process : Workflow of activities allowing to build a product (software or system)
  - Method : How to conduct an activity
  - Tools : Partially automated support for the activity and its method
- Common activities: requirement analysis, architectural and component design, implementation, verification, validation, delivery, maintenance
- Common V & V : Testing (unit, integration, function, system, user, performance), Proofreading
- Common lifecycle models (high-level workflows): cascade, V, W, iteration
- Model Driven Engineering: Rely on the most appropriate domain specific methods and tools for each activity (instead of generic ones)

Software and System Engineering

- Process, Methods and Tools
  - Process : Workflow of activities allowing to build a product (software or system)
  - Method : How to conduct an activity
  - Tools : Partially automated support for the activity and its method
- Common activities: requirement analysis, architectural and component design, implementation, verification, validation, delivery, maintenance
- Common V & V : Testing (unit, integration, function, system, user, performance), Proofreading
- Common lifecycle models (high-level workflows): cascade, V, W, iteration
- Model Driven Engineering: Rely on the most appropriate domain specific methods and tools for each activity (instead of generic ones)

Software and System Engineering

- Process, Methods and Tools
  - Process : Workflow of activities allowing to build a product (software or system)
  - Method : How to conduct an activity
  - Tools : Partially automated support for the activity and its method
- Common activities: requirement analysis, architectural and component design, implementation, verification, validation, delivery, maintenance
- Common V & V : Testing (unit, integration, function, system, user, performance), Proofreading
- Common lifecycle models (high-level workflows): cascade, V, W, iteration
- Model Driven Engineering: Rely on the most appropriate domain specific methods and tools for each activity (instead of generic ones)

Software and System Engineering

- Process, Methods and Tools
  - Process : Workflow of activities allowing to build a product (software or system)
  - Method : How to conduct an activity
  - Tools : Partially automated support for the activity and its method
- Common activities: requirement analysis, architectural and component design, implementation, verification, validation, delivery, maintenance
- Common V & V : Testing (unit, integration, function, system, user, performance), Proofreading
- Common lifecycle models (high-level workflows): cascade, V, W, iteration
- Model Driven Engineering: Rely on the most appropriate domain specific methods and tools for each activity (instead of generic ones)

Formal methods

- Rely on mathematical formalism to assess:
  - consistency (remove ambiguities)
  - completeness (everything has been handled)
  - correctness of something (implementation) versus something else (specification)
- Formal specification (logic, set theory, algebra, transition systems, ...)
  - Requirements
  - System execution context
  - System implementation
- Formal verification (consistency, completeness, correctness, ...)
  - Deductive methods: Translation to logic and semi-automated proof
  - Abstract execution: Execution in an abstract domain
  - Model checking: Exhaustive test of finite models (explicit or implicit – symbolic)

Formal methods

- Rely on mathematical formalism to assess:
  - consistency (remove ambiguities)
  - completeness (everything has been handled)
  - correctness of something (implementation) versus something else (specification)
- Formal specification (logic, set theory, algebra, transition systems, . . .)
  - Requirements
  - System execution context
  - System implementation
- Formal verification (consistency, completeness, correctness, . . .)
  - Deductive methods: Translation to logic and semi-automated proof
  - Abstract execution: Execution in an abstract domain
  - Model checking: Exhaustive test of finite models (explicit or implicit – symbolic)

Formal methods

- Rely on mathematical formalism to assess:
  - consistency (remove ambiguities)
  - completeness (everything has been handled)
  - correctness of something (implementation) versus something else (specification)
- Formal specification (logic, set theory, algebra, transition systems, ...)
  - Requirements
  - System execution context
  - System implementation
- Formal verification (consistency, completeness, correctness, ...)
  - Deductive methods: Translation to logic and semi-automated proof
  - Abstract execution: Execution in an abstract domain
  - Model checking: Exhaustive test of finite models (explicit or implicit – symbolic)

Plan

1 Introduction

2 Contexte certification
- DO178/ED12 safety standards

3 Approche déductive

DO-178/ED-12 safety standards: Certification

- Onboard software in aeronautics: Design Assurance Level
  Failure impact: DAL A – Catastrophic failure . . . DAL E – No impact
- Early releases in the 80s, major revision in 1992 (B – 3 years of work), and 2012 (C – 7 years of work): adaptation to technological changes
- Most constraining standard up to now
  accepted by other standards (automotive, space, . . .)
- Main concern: Safety of passengers
  System requirement : 10^{-9} per flight hour for DAL A – ARP 4754
- Main purpose:
  Provide confidence in the system and its development

DO-178/ED-12 safety standards: Certification

- Onboard software in aeronautics: Design Assurance Level
  Failure impact: DAL A – Catastrophic failure . . . DAL E – No impact
- Early releases in the 80s, major revision in 1992 (B – 3 years of work), and 2012 (C – 7 years of work): adaptation to technological changes
- Most constraining standard up to now
  accepted by other standards (automotive, space, . . .)
- Main concern: Safety of passengers
  System requirement : 10^{-9} per flight hour for DAL A – ARP 4754
- Main purpose:
  Provide confidence in the system and its development

DO-178/ED-12 safety standards: Certification

- Key issue:
  Choose the strategy and technologies that will minimize risks
- Assessment: Stochastic for system, Zero-default for Software
- Process and test-centered approach
  - Definition of a precise process (development/verification)
  - MC-DC test coverage for DAL A
    truth-table lines of sub-expressions in conditions (some can be merged)
  - Asymmetry with independence argument: several activities (and products) by different teams, with different tools, ...

Process centered approach

- Requirement: What is expected from a system
  - High level (HLR): focus on end users needs (user provided)
  - Low level (LLR): focus on technical solutions (developer provided)
- Traceability: Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification: System fulfills its requirements explicit specification (make the product right)
- Validation: System fulfills its requirements implicit human needs (make the right product)
- Certification: System (and its development) follows standards (safety in our case: DO-178/ED-12, IEC-61508, ISO-26262, ...)
- Qualification: Tools for system development follows standards
- Certification and qualification: Historically, system context related (no component, COTS, reuse, ... up to DO-178C/ED-12C)

Process centered approach

- Requirement: What is expected from a system
  - High level (HLR): focus on end users needs (user provided)
  - Low level (LLR): focus on technical solutions (developer provided)
- Traceability: Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification: System fulfills its requirements explicit specification (make the product right)
- Validation: System fulfills its requirements implicit human needs (make the right product)
- Certification: System (and its development) follows standards (safety in our case: DO-178/ED-12, IEC-61508, ISO-26262, . . .)
- Qualification: Tools for system development follows standards
- Certification and qualification: Historically, system context related (no component, COTS, reuse, . . . up to DO-178C/ED-12C)

Process centered approach

- Requirement: What is expected from a system
  - High level (HLR): focus on end users needs (user provided)
  - Low level (LLR): focus on technical solutions (developer provided)
- Traceability: Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification: System fulfills its requirements explicit specification (make the product right)
- Validation: System fulfills its requirements implicit human needs (make the right product)
- Certification: System (and its development) follows standards (safety in our case: DO-178/ED-12, IEC-61508, ISO-26262, . . .)
- Qualification: Tools for system development follows standards
- Certification and qualification: Historically, system context related (no component, COTS, reuse, . . . up to DO-178C/ED-12C)

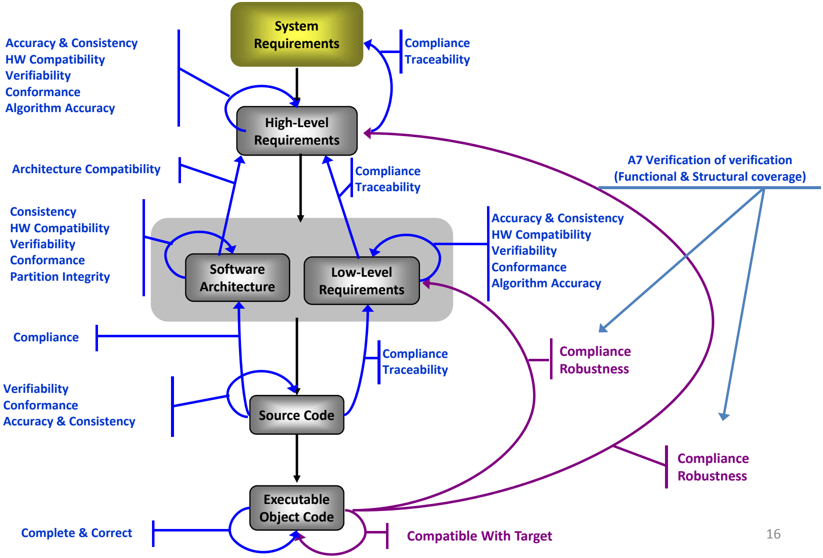
Process centered approach

- Requirement: What is expected from a system
  - High level (HLR): focus on end users needs (user provided)
  - Low level (LLR): focus on technical solutions (developer provided)
- Traceability: Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification: System fulfills its requirements explicit specification (make the product right)
- Validation: System fulfills its requirements implicit human needs (make the right product)
- Certification: System (and its development) follows standards (safety in our case: DO-178/ED-12, IEC-61508, ISO-26262, . . .)
- Qualification: Tools for system development follows standards
- Certification and qualification: Historically, system context related (no component, COTS, reuse, . . . up to DO-178C/ED-12C)

Process centered approach

- Requirement: What is expected from a system
  - High level (HLR): focus on end users needs (user provided)
  - Low level (LLR): focus on technical solutions (developer provided)
- Traceability: Explicit relations between various elements in a system development (requirements, design and implementation choices)
- Verification: System fulfills its requirements explicit specification (make the product right)
- Validation: System fulfills its requirements implicit human needs (make the right product)
- Certification: System (and its development) follows standards (safety in our case: DO-178/ED-12, IEC-61508, ISO-26262, . . .)
- Qualification: Tools for system development follows standards
- Certification and qualification: Historically, system context related (no component, COTS, reuse, . . . up to DO-178C/ED-12C)

DO-178/ED-12 Global process



Phase 1: Process definition and early certification

- Plan for Software Aspects of Certification (PSAC)
- Software Development Plan (SDP)
- Software Verification Plan (SVP)
- Software Configuration Management Plan (SCMP)
- Software Quality Assurance Plan (SQAP)
  applied only to the other plans
- Tool Qualification Plan (TQP)
  it tools are used to automatize activities

Phase 2: Process application verification

- User requirements (HLR)
- Software architecture (elementary parts and their assembly)
- Software requirements (Detailled design of elementary parts): Can be refined user requirements or derived requirements (linked to technology choices, should be avoided or strongly justified)
- Executable Object Code (EOC) integration on Hardware
- Verification results
- Traceability links between requirements and software

DO-178C/ED-12C: Main changes

- Convergence with DO-278 (ground software)
- Merge elements from DO-248 and many CASTs
- Supplements:
  - DO-331: Model based development and verification
  - DO-332: Object oriented technologies and related technics
  - DO-333: Formal methods
- New document: DO-330 Tool Qualification

Model based development and verification

- Use of models as requirements: HLR from System phases and LLR from Design
- Applies to any models related to Software elements (including System phases)
- Can be used for communication or automatization (analysis, code generation)
- Models can be more abstract the Software and partial
- Requires Higher Lever Requirements (HiLR) to assess the models
- Modeling language must be precise and appropriate
  - Specification models: HLR (can be Design models HiLR)
  - Design models: LLR (requires test based on HiLR)

Formal methods

- A formal method must be correctly defined, justified and appropriate
  - Correctly defined: precise, unambiguous, mathematically defined syntax and semantics
  - Justified: Sound (never assert a false property)
  - Appropriate: Assumptions required by formal analysis must be described and justified
- Requirement formalization correctness
- Formal analysis can replace:
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, ...)
- Adapted coverage analysis:
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintended data flow
  - Detection of extraneous code (dead or deactivated)
- But: Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

Formal methods

- A formal method must be correctly defined, justified and appropriate
  - Correctly defined: precise, unambiguous, mathematically defined syntax and semantics
  - Justified: Sound (never assert a false property)
  - Appropriate: Assumptions required by formal analysis must be described and justified
- Requirement formalization correctness
- Formal analysis can replace:
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, ...)
- Adapted coverage analysis:
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintended data flow
  - Detection of extraneous code (dead or deactivated)
- But: Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

Formal methods

- A formal method must be correctly defined, justified and appropriate
  - Correctly defined: precise, unambiguous, mathematically defined syntax and semantics
  - Justified: Sound (never assert a false property)
  - Appropriate: Assumptions required by formal analysis must be described and justified
- Requirement formalization correctness
- Formal analysis can replace:
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, . . .)
- Adapted coverage analysis:
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintended data flow
  - Detection of extraneous code (dead or deactivated)
- But: Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

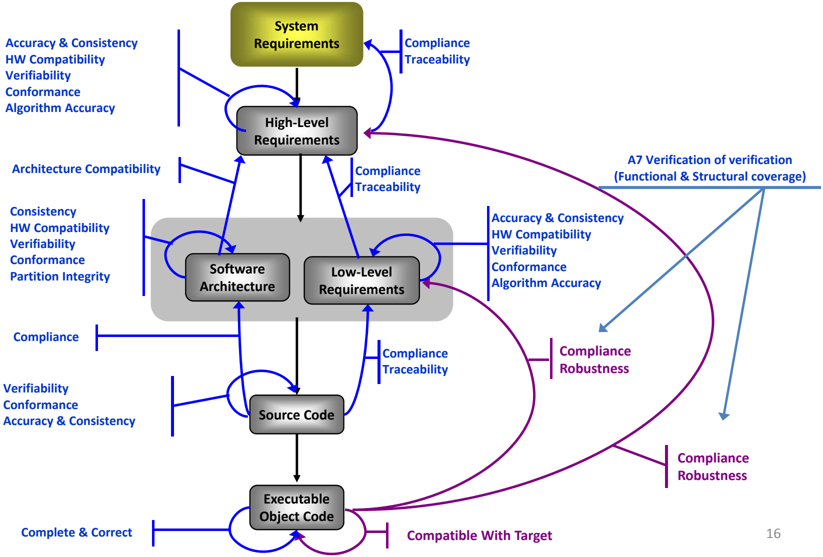
Formal methods

- A formal method must be correctly defined, justified and appropriate
  - Correctly defined: precise, unambiguous, mathematically defined syntax and semantics
  - Justified: Sound (never assert a false property)
  - Appropriate: Assumptions required by formal analysis must be described and justified
- Requirement formalization correctness
- Formal analysis can replace:
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, ...)
- Adapted coverage analysis:
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintended data flow
  - Detection of extraneous code (dead or deactivated)
- But: Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

Formal methods

- A formal method must be correctly defined, justified and appropriate
- Requirement formalization correctness
- Formal analysis can replace:
  - Review and analysis objectives
  - Conformance tests versus HLR and LLR
  - Robustness tests
  - Compatibility with the hardware (WCET, . . .)
- Adapted coverage analysis:
  - Complete coverage of each requirement
  - Completeness of the requirements
  - Detection of unintended data flow
  - Detection of extraneous code (dead or deactivated)
- But: Formal analysis cannot replace hardware/software integration tests. Tests is still a required activity at higher level

DO-178/ED-12 Formal method use



Plan

Principes généraux

- Travaux de Floyd (Turing 1978), Hoare (Turing 1980) et Dijsktra (Turing 1972)
  - Annotations des programmes par les propriétés des états intermédiaires
  - Préconditions (resp. Postconditions) : propriétés satisfaites avant (resp. Après) l'exécution d'un programme (d'une instruction)
  - Invariants : Propriétés toujours satisfaites durant l'exécution
  - Variants : Expressions strictement décroissantes et bornées inférieurement (ordre bien fondée)
- Origine de la programmation par contrats (Meyer)
- Exemples : Méthodes B et Event-B
- Exemples : Outils CAVEAT, frama-C, Spark-Ada, Spec-$\sharp$, Why3, Boogie, $F^{\star}$

Exemple : factorielle

$\{n \geq 0\}$

$\Rightarrow$

$\{1 \times n! = n! \wedge n \geq 0\}$

$x := 1;$

$\{x \times n! = n! \wedge n \geq 0\}$

$y := n;$

$\{x \times y! = n! \wedge y \geq 0\}$

while $y \neq 0$ *inv* $x \times y! = n! \wedge y \geq 0$ *do*

$\quad \{x \times y! = n! \wedge y \geq 0 \wedge y \neq 0\}$

$\quad \Rightarrow$

$\quad \{x \times y \times (y-1)! = n! \wedge y \geq 1\}$

$\quad x := x \times y;$

$\quad \{x \times (y-1)! = n! \wedge (y-1) \geq 0\}$

$\quad y := y - 1$

$\quad \{x \times y! = n! \wedge y \geq 0\}$

od;

$\{x \times y! = n! \wedge y \geq 0 \wedge \neg y \neq 0\}$

$\Rightarrow$

$\{x = n!\}$

Exemple : factorielle

$$
\begin{array}{l}
\{n \geq 0\} \\
\Rightarrow \\
\{1 \times n! = n! \land n \geq 0\} \\
x := 1; \\
\{x \times n! = n! \land n \geq 0\} \\
y := n; \\
\{x \times y! = n! \land y \geq 0\} \\
\textit{while } y \neq 0 \textit{ inv } x \times y! = n! \land y \geq 0 \textit{ do} \\
\qquad \{x \times y! = n! \land y \geq 0 \land y \neq 0\} \\
\qquad \Rightarrow \\
\qquad \{x \times y \times (y-1)! = n! \land y \geq 1\} \\
\qquad x := x \times y; \\
\qquad \{x \times (y-1)! = n! \land (y-1) \geq 0\} \\
\qquad y := y - 1 \\
\qquad \{x \times y! = n! \land y \geq 0\} \\
\textit{od}; \\
\{x \times y! = n! \land y \geq 0 \land \neg y \neq 0\} \\
\Rightarrow \\
\{x = n!\}
\end{array}
$$

Exemple : factorielle

$\{n \geq 0\}$

$\Rightarrow$

$\{1 \times n! = n! \wedge n \geq 0\}$

$x := 1;$

$\{x \times n! = n! \wedge n \geq 0\}$

$y := n;$

$\{x \times y! = n! \wedge y \geq 0\}$

$\mathit{while}\ y \neq 0\ \mathit{inv}\ x \times y! = n! \wedge y \geq 0\ \mathit{do}$

$\quad \{x \times y! = n! \wedge y \geq 0 \wedge y \neq 0\}$

$\quad \Rightarrow$

$\quad \{x \times y \times (y-1)! = n! \wedge y \geq 1\}$

$\quad x := x \times y;$

$\quad \{x \times (y-1)! = n! \wedge (y-1) \geq 0\}$

$\quad y := y - 1$

$\quad \{x \times y! = n! \wedge y \geq 0\}$

$\mathit{od};$

$\{x \times y! = n! \wedge y \geq 0 \wedge \neg y \neq 0\}$

$\Rightarrow$

$\{x = n!\}$

Exemple : factorielle

$\{n \geq 0\}$
$\Rightarrow$
$\{1 \times n! = n! \wedge n \geq 0\}$
$x := 1;$
$\{x \times n! = n! \wedge n \geq 0\}$
$y := n;$
$\{x \times y! = n! \wedge y \geq 0\}$
$\mathit{while}\ y \neq 0\ \mathit{inv}\ x \times y! = n! \wedge y \geq 0\ \mathit{do}$
  $\{x \times y! = n! \wedge y \geq 0 \wedge y \neq 0\}$
  $\Rightarrow$
  $\{x \times y \times (y-1)! = n! \wedge y \geq 1\}$
  $x := x \times y;$
  $\{x \times (y-1)! = n! \wedge (y-1) \geq 0\}$
  $y := y - 1$
  $\{x \times y! = n! \wedge y \geq 0\}$
$\mathit{od};$
$\{x \times y! = n! \wedge y \geq 0 \wedge \neg y \neq 0\}$
$\Rightarrow$
$\{x = n!\}$

Exemple : factorielle

$\{n \geq 0\}$
$\Rightarrow$
$\{1 \times n! = n! \wedge n \geq 0\}$
$x := 1;$
$\{x \times n! = n! \wedge n \geq 0\}$
$y := n;$
$\{x \times y! = n! \wedge y \geq 0\}$
$\mathit{while}\ y \neq 0\ \mathit{inv}\ x \times y! = n! \wedge y \geq 0\ \mathit{do}$
$\quad\{x \times y! = n! \wedge y \geq 0 \wedge y \neq 0\}$
$\quad\Rightarrow$
$\quad\{x \times y \times (y-1)! = n! \wedge y \geq 1\}$
$\quad x := x \times y;$
$\quad\{x \times (y-1)! = n! \wedge (y-1) \geq 0\}$
$\quad y := y - 1$
$\quad\{x \times y! = n! \wedge y \geq 0\}$
$\mathit{od};$
$\{x \times y! = n! \wedge y \geq 0 \wedge \neg y \neq 0\}$
$\Rightarrow$
$\{x = n!\}$

Exemple : factorielle

$\{n \geq 0\}$
$\Rightarrow$
$\{1 \times n! = n! \wedge n \geq 0\}$
$x := 1;$
$\{x \times n! = n! \wedge n \geq 0\}$
$y := n;$
$\{x \times y! = n! \wedge y \geq 0\}$
$while\ y \neq 0\ inv\ x \times y! = n! \wedge y \geq 0\ do$
$\quad \{x \times y! = n! \wedge y \geq 0 \wedge y \neq 0\}$
$\quad \Rightarrow$
$\quad \{x \times y \times (y-1)! = n! \wedge y \geq 1\}$
$\quad x := x \times y;$
$\quad \{x \times (y-1)! = n! \wedge (y-1) \geq 0\}$
$\quad y := y - 1$
$\quad \{x \times y! = n! \wedge y \geq 0\}$
$od;$
$\{x \times y! = n! \wedge y \geq 0 \wedge \neg y \neq 0\}$
$\Rightarrow$
$\{x = n!\}$

Correction partielle : Système de déduction

$$\{\varphi\}\, P\, \{\psi\}$$

φ et ψ fonctions de l'état de la mémoire

$$\{[E/x]\,\varphi\}\, x := E\, \{\varphi\}$$

$$\frac{\{\varphi \wedge C\}\, P\, \{\psi\} \quad \{\varphi \wedge \neg C\}\, Q\, \{\psi\}}{\{\varphi\}\ \mathit{if}\ C\ \mathit{then}\ P\ \mathit{else}\ Q\ \mathit{fi}\ \{\psi\}}$$

$$\frac{\{\varphi \wedge C\}\, P\, \{\varphi\}}{\{\varphi\}\ \mathit{while}\ C\ \mathit{do}\ P\ \mathit{od}\ \{\varphi \wedge \neg C\}}$$

$$\frac{\{\varphi\}\, P\, \{\chi\} \quad \{\chi\}\, Q\, \{\psi\}}{\{\varphi\}\, P\,;\ Q\, \{\psi\}}$$

$$\frac{\{\varphi\}\, P\, \{\chi\} \quad \chi \Rightarrow \psi}{\{\varphi\}\, P\, \{\psi\}}$$

$$\frac{\varphi \Rightarrow \chi \quad \{\chi\}\, P\, \{\psi\}}{\{\varphi\}\, P\, \{\psi\}}$$

Exemple : division entière

$\{x \geq 0 \wedge y > 0\}$
$q := 0;$
$r := x;$
$\mathit{tantque}\ y \leq r\ \mathit{faire}$
$\qquad q := q + 1;$
$\qquad r := r - y$
$\mathit{fait};$
$\{x = q \times y + r \wedge 0 \leq q \wedge 0 \leq r < y\}$

Correction totale : Système de déduction

$$\{\varphi\}\, P\, \{\psi\}$$

Construction d'une relation d'ordre bien fondée pour chaque boucle : Variant (pratiquement une fonction de la mémoire vers $\mathbb{N}$, c'est à dire une expression v qui exploite les variables qui représentent la mémoire)

$$\frac{\{\varphi \wedge C \wedge v = V \wedge v \in \mathbb{N}\}\, P\, \{\varphi \wedge v < V \wedge v \in \mathbb{N}\}}{\{\varphi\}\, \textit{while } C \textit{ do } P \textit{ od}\, \{\varphi \wedge \neg C\}}$$

Exemple : factorielle

Variant : $v = y$

$$
\begin{array}{l}
\{n \geq 0\} \\
\Rightarrow \\
\{1 = 1 \wedge n \geq 0\}(\varphi_1) \\
x := 1; \\
\{x = 1 \wedge n \geq 0\}(\varphi_1) \\
y := n; \\
\{x \times y! = n! \wedge y \geq 0\} \\
\textit{while } y \neq 0 \textit{ inv } x \times y! = n! \wedge y \geq 0 \textit{ var } y \textit{ do} \\
\quad \{x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \wedge y = V \wedge y - 1 < y \wedge y \in \mathbb{N}\} \\
\quad \Rightarrow \\
\quad \{x \times y! = n! \wedge y \geq 0 \wedge y \neq 0 \wedge y = V \wedge y - 1 < V \wedge y - 1 \in \mathbb{N}\} \\
\quad x := x \times y; \\
\quad \{x \times (y-1)! = n! \wedge (y-1) \geq 0 \wedge y - 1 < V \wedge y - 1 \in \mathbb{N}\} \\
\quad y := y - 1 \\
\{x \times y! = n! \wedge y \geq 0 \wedge y < V \wedge y \in \mathbb{N}\} \\
\textit{od}; \\
\{x \times y! = n! \wedge y \geq 0 \wedge y = 0\} \\
\Rightarrow \\
\{x = n!\}
\end{array}
$$

Automatisation : Abstraction du langage

Les règles de preuve sont spécifiques aux langages
Les propriétés prouvées sont génériques
Obligation de preuves (appelée ici Verification Condition) : Transformation des triplets de Hoare en formules logiques indépendantes du langage

$$\forall \varphi, \psi, vc(\{\varphi\}\, P\, \{\psi\}) \Rightarrow \{\varphi\}\, P\, \{\psi\}$$

$$vc(\{\varphi\}\, \texttt{skip}\, \{\psi\}) = \{\varphi \Rightarrow \psi\}$$

$$vc(\{\varphi\}\, id := E\, \{\psi\}) = \{\varphi \Rightarrow [E/id]\psi\}$$

Automatisation : Abstraction du langage

$$vc(\{\varphi\}\, P\,;\, \{\chi\}\, Q\, \{\psi\}) = vc(\{\varphi\}\, P\, \{\chi\}) \cup vc(\{\chi\}\, Q\, \{\psi\})$$

$$vc(\{\varphi\}\, \texttt{if}\, (C)\, \texttt{then}\, P\, \texttt{else}\, Q\, \texttt{fi}\, \{\psi\}) = \begin{array}{l} vc(\{C \wedge \varphi\}\, P\, \{\psi\}) \\ \cup \\ vc(\{\neg C \wedge \varphi\}\, Q\, \{\psi\}) \end{array}$$

$$vc(\{\varphi\}\, \texttt{while}\, (C)\, \texttt{invariant}\, \chi\, \texttt{do}\, P\, \texttt{od}\, \{\psi\}) = \begin{array}{l} \{\varphi \Rightarrow \chi, \chi \wedge \neg C \Rightarrow \psi\} \\ \cup \\ vc(\{C \wedge \chi\}\, P\, \{\chi\}) \end{array}$$

Exercice : Calculer les verification conditions pour l'exemple précédent (factorielle)

Automatisation : Minimisation des formules générées

Problème majeur : déterminer les variants et invariants
Annotation du programme par l'utilisateur au niveau des boucles
Calcul de la plus faible précondition (weakest precondition) telle que :

$$\forall\psi, \{wp(P,\psi)\}P\{\psi\}$$

$$\text{c'est-à-dire } \forall\varphi,\psi,\{\varphi\}\,P\,\{\psi\} \Rightarrow (\varphi \Rightarrow wp(P,\psi))$$

Peut prendre la forme d'un transformateur de prédicat $[P]$:

$$[P](\psi) = wp(P,\psi)$$

Weakest precondition

$$wp(\texttt{skip}, \psi) = \psi$$

$$wp(id := E, \psi) = [E/id]\psi$$

$$wp(P\,;\; Q, \psi) = wp(P, wp(Q, \psi))$$

$$wp(\texttt{if}\,(C)\,\texttt{then}\,P\,\texttt{else}\,Q\,\texttt{fi}, \psi) = \begin{array}{l}(C \Rightarrow wp(P, \psi)) \\ \wedge \\ (\neg C \Rightarrow wp(Q, \psi))\end{array}$$

Weakest precondition

Répetition et invariant

$$wp(\texttt{while}\,(C)\,\texttt{do}\,P\,\texttt{od},\psi) = \begin{array}{l}(C \Rightarrow wp(P, wp(\texttt{while}\,(C)\,\texttt{do}\,P\,\texttt{od},\psi)))\\ \wedge \\ (\neg C \Rightarrow \psi)\end{array}$$

Dans le cas du `while`, il faut développer expliciter le point fixe en fonction du nombre d'étapes de calcul ce que n'est pas praticable.

$$\begin{array}{l} wp(\texttt{while}\,(C)\,\texttt{do}\,P\,\texttt{od},\psi) = \bigwedge_{i \in \mathbb{N}} \psi_i \\ \psi_0 = \psi \\ \psi_{i+1} = (C \Rightarrow wp(P, \psi_i)) \wedge (\neg C \Rightarrow \psi_i) \end{array}$$

Mélange des deux approches

J'utilise ici le terme de *Proof Obligation* pour la combinaison de WP et VC.

$$vc(\{\varphi\}\, P\, \{\psi\}) = \{\varphi \Rightarrow \chi\} \cup \mathcal{E} \text{ avec } \langle \chi, \mathcal{E} \rangle = po(Q, \psi)$$

$$po(\texttt{skip}, \psi) = \langle \psi, \emptyset \rangle$$

$$po(id := E, \psi) = \langle [E/id]\psi, \emptyset \rangle$$

$$po(P\,;\ Q, \psi) = \langle \varphi_2, \mathcal{E}_1 \cup \mathcal{E}_2 \rangle \text{ avec } \langle \varphi_2, \mathcal{E}_2 \rangle = po(P, \varphi_1)\text{et}\langle \varphi_1, \mathcal{E}_1 \rangle = po(Q, \psi)$$

Mélange des deux approches

$$
\begin{aligned}
po(\texttt{if}\,(C)\,\texttt{then}\,P\,\texttt{else}\,Q\,\texttt{fi},\psi) = \ & \langle (C \Rightarrow \varphi_1) \wedge (\neg C \Rightarrow \varphi_2), \mathcal{E}_1 \cup \mathcal{E}_2 \rangle \\
& \text{avec } \langle \varphi_1, \mathcal{E}_1 \rangle = po(P,\psi) \\
& \text{avec } \langle \varphi_2, \mathcal{E}_2 \rangle = po(Q,\psi)
\end{aligned}
$$

$$
\begin{aligned}
po(\texttt{while}\,(C)\,\texttt{invariant}\,\chi\,\texttt{do}\,P\,\texttt{od},\psi) = \ & \langle \chi, \mathcal{E}_1 \cup \mathcal{E}_2 \rangle \\
& \text{avec } \mathcal{E}_2 = \left\{ \begin{array}{c} C \wedge \chi \Rightarrow \varphi, \\ \neg C \wedge \chi \Rightarrow \psi \end{array} \right\} \\
& \text{avec } \langle \varphi, \mathcal{E}_1 \rangle = po(P,\chi)
\end{aligned}
$$

Exemple : factorielle

Exercice : Calculer les obligations de preuve pour le programme suivant.

$$\begin{array}{l}
\{n \geq 0\} \\
x := 1; \\
y := n; \\
\mathit{while}\ y \neq 0\ \mathit{invariant}\ x \times y! = n! \wedge y \geq 0\ \mathit{do} \\
\qquad x := x \times y; \\
\qquad y := y - 1 \\
\mathit{od}; \\
\{x = n!\}
\end{array}$$

Notons le programme P, la partie avant la boucle A, la boucle B, le corps de boucle C.

Exemple : factorielle

Calculons les obligations de preuve pour les différentes parties de l'exemple A, C et B puis concluons pour P.

$$\begin{aligned} po(A, x \times y! = n! \wedge y \geq 0) &= \langle 1 \times n! = n! \wedge n \geq 0, \emptyset \rangle \\ &= \langle n \geq 0, \emptyset \rangle \end{aligned}$$

$$\begin{aligned} po(C, x \times y! = n! \wedge y \geq 0) &= \langle x \times y \times (y-1)! = n! \wedge y - 1 \geq 0, \emptyset \rangle \\ &= \langle x \times y! = n! \wedge y > 0, \emptyset \rangle \end{aligned}$$

$$po(B, x = n!) = \langle x \times y! = n! \wedge y \geq 0, \mathcal{E} \rangle$$

$$\mathcal{E} = \left\{ \begin{array}{l} (x \times y! = n! \wedge y \geq 0) \wedge y \neq 0 \Rightarrow (x \times y! = n! \wedge y > 0), \\ (x \times y! = n! \wedge y \geq 0) \wedge \neg(y \neq 0) \Rightarrow x = n! \end{array} \right\}$$

$$vc(\{n \geq 0\}\, P\, \{x = n!\}) = \{n \geq 0 \Rightarrow n \geq 0\} \cup \mathcal{E}$$

Toutes les obligations sont valides donc le programme est correct.