Séminaire Collège de France, 11 mars 2015

Dominique Bolignano

77, avenue Niel, 75017 Paris, France

contact@provenrun.com

Utilisation des méthodes formelles pour la sécurisation de systèmes complexes: une avancée industrielle

Dominique Bolignano

Prove & Run

Paris

Utilisation des méthodes formelles pour la sécurisation de systèmes complexes: une avancée industrielle

Dominique Bolignano

Prove & Run

Paris

Plan

- Modélisation d’un micronoyau
  - Approche générale
  - Niveaux d’abstraction
  - Lien avec les schémas de sécurité
- Outillage et Chaine de développement
- Sécurité et Identification de la TCB
- Besoins spécifiques identifiés
- Spécificité du langage
- Conclusion

Modélisation d’un micro-noyau

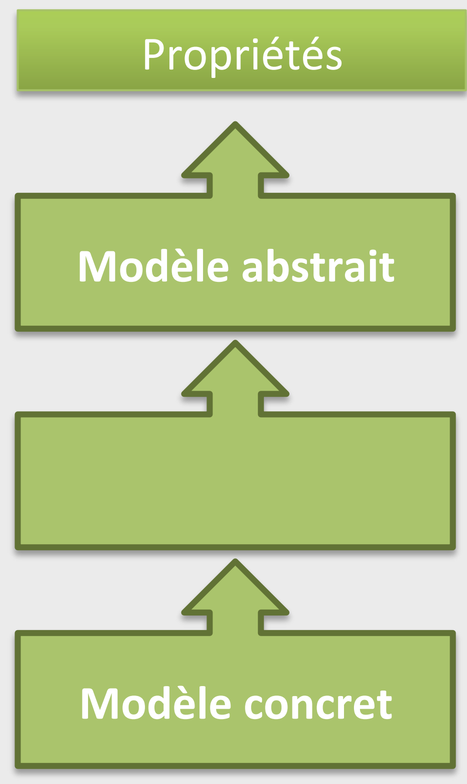
User Mode	User Processes	Shell	Make	User	....	Other
	Server Processes	File	PM	Reinc	....	Other
	Device Processes	Disk	TTY	Net	Print	Other
Kernel Mode	Kernel	Clock task	System task			

The MINIX 3 Microkernel Architecture

(Stéphane Lescuyer – Vincent Siles – Benoit Montagu)

Towards a Verified Isolation Micro-Kernel – Stéphane Lescuyer HIPEAC 01/2015

Modélisation d'un micro-noyau: approche générale



- Un niveau abstrait reprenant tous les comportements des niveaux plus concrets.
- Propriétés formelles exprimées au niveau le plus haut possible.
- Propriétés plus naturelles et surtout plus simples à comprendre.

Modélisation d'un micro-noyau: Lien avec les schémas de sécurités

Propriétés de sécurité

Propriétés principales

Propriétés de sécurité

✔

SPM ✔

✔

FSP ✔

✔

TDS ✔

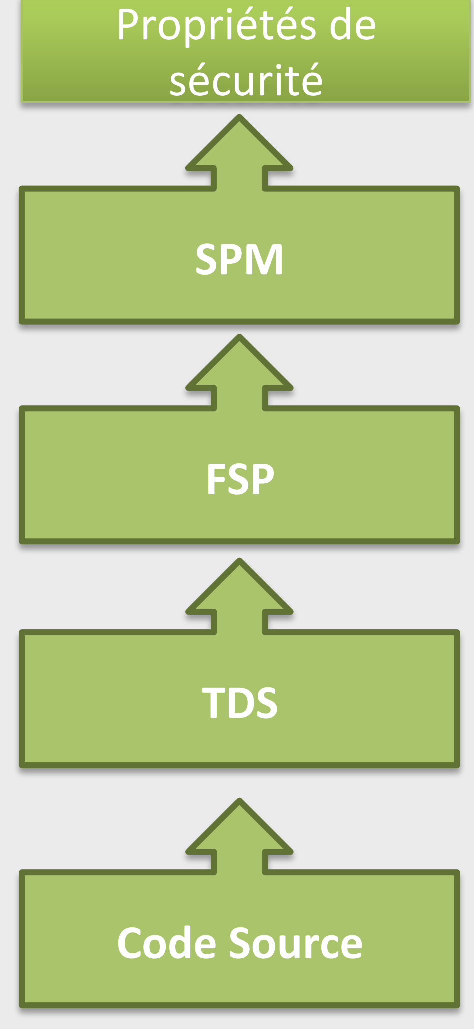
✔

Code Source ✔

✔ Prouvé

✔ Vérifié informellement

Modélisation d'un micro-noyau: Preuve du code source



Modélisation d'un micro-noyau: Propriétés

Propriétés de sécurité

SPM

P1 P3 Pn

SPM

FSP

TDS

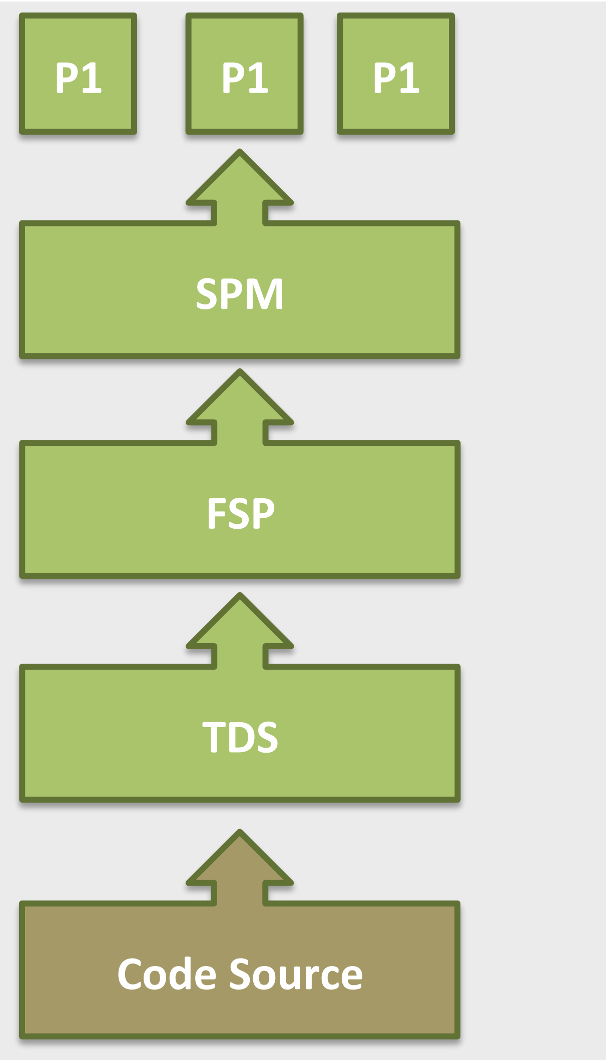
Code Source

P1 P2 Pn

SPM

Modélisation d'un micro-noyau: Récapitulatif

Propriétés de sécurité



P1 P1 P1

SPM

Propriétés et Modèle Abstrait

- Le modèle doit être aussi abstrait que possible tout en capturant la propriété souhaitée,
- Paradigme: machines indépendantes chacune exécutant avec ses propres ressources (code, donnée, mémoires, etc.), potentiellement communicant et/ou partageant certaines ressources, comme les zones mémoires, des systèmes de fichiers, etc.

Séparation

Machine 1 | Machine 2 | Machine 3 | ..... | Machine n

Regs | Regs | Regs | Regs

Data | Data | Data | Data

Code | Code | Code | Code

CONTROLLER

SPM - Stéphane Lescuyer

Outillage SMART

Environnement de développement : plugin Eclipse
Florence Plateau,
Laurent Hubert,
Sylvain Héraud

Prouveur : plugin Eclipse
Stéphane Lescuyer,
Clément Hurlin,

P&R Intermediate Language: SMIL

Générateur (source et documentation) : plugin Eclipse
Laurent Hubert,
Olivier Delande,
Thomas Jensen,
Denis Cousineau,
Alexandre Thévenet,

Automatisé

Source Code

- Compilable
- C, Java, etc.

Certification Documentation

- CC
- DO-178
- Etc

De l’intérêt des méthodes formelles pour la cybersécurité

- Chaîne de sécurité.
  - Algorithmes cryptographiques,
  - Secure elements (e.g. carte à puce),
  - Protocoles cryptographiques,
  - Résistance des systèmes à des attaques logiques.
- Problématique des erreurs/vulnérabilités, notamment dans les systèmes d’exploitation:
  - Une situation alarmante et qui se dégrade (e.g. statistiques du National Vulnerability Data Base).
- La réponse.
  - Une TCB bien délimitée et réduite.
  - Les méthodes formelles appliquées sur ces TCB.

Sécurité et identification de la TCB

INIT
SHELL
...
PM
PM data
FS
...
SYSTEM
TCB
KERNEL
CLOCK
Kernel data structures
IPC call
Function call

Processus d’identification des besoins Stratégie d’application (1/2)

- **Expériences, élaborés petit à petit.**
  - **En utilisant de nombreuses approches formelles sur de nombreux cas réels en entreprise sur de nombreuses années.**
  - **Dans des contextes où il fallait justifier à chaque fois l’applicabilité et l’intérêt pour le projet.**
- **Stratégie (1/2):**
  - **Choisir les situations dans lesquelles le ratio bénéfice/coût est très favorable et le marché représentatif.**
  - **Réduction du coût (micronoyau, …).**
    - Identification ou mieux meilleure définition de la TCB.
    - Bénéfices de la réutilisation,
  - **Permettre la maintenabilité,**
  - **Faciliter l’usage des Méthodes Formelles de manière à pouvoir les mettre dans les mains des développeurs,**

Processus d'identification des besoins Stratégie d'application (1/2)

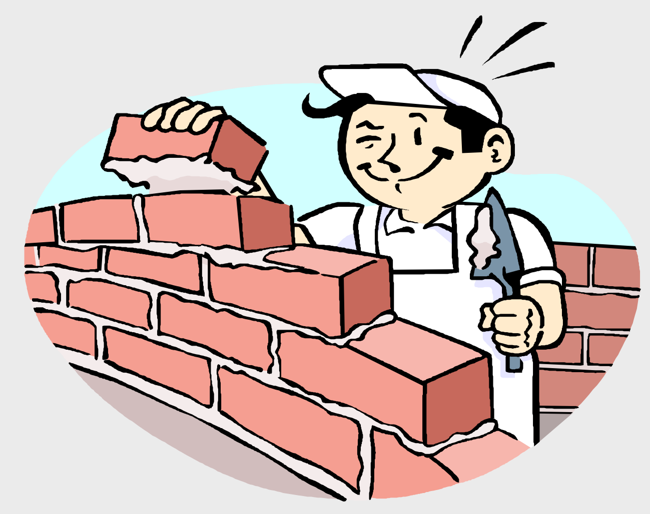
- **Expériences, élaborés petit à petit.**
  - **En utilisant de nombreuses approches formelles sur de nombreux cas réels en entreprise sur de nombreuses années.**
  - **Dans des contextes où il fallait justifier à chaque fois l'applicabilité et l'intérêt pour le projet.**
- **Stratégie (1/2):**
  - **Choisir les situations dans lesquelles le ratio bénéfice/coût est très favorable et le marché représentatif.**
  - **Réduction du coût (micronoyau, …).**
    - Identification ou mieux meilleure définition de la TCB.
    - Bénéfices de la réutilisation,
  - **Permettre la maintenabilité,**
  - **Faciliter l'usage des Méthodes Formelles de manière à pouvoir les mettre dans les mains des développeurs,**

Processus d'identification des besoins Stratégie d'application (1/2)

- Expériences, élaborés petit à petit.
  - En utilisant de nombreuses approches formelles sur de nombreux cas réels en entreprise sur de nombreuses années.
  - Dans des contextes où il fallait justifier à chaque fois l'applicabilité et l'intérêt pour le projet.
- Stratégie (1/2):
  - Choisir les situations dans lesquelles le ratio bénéfice/coût est très favorable et le marché représentatif.
  - Réduction du coût (micronoyau, …).
    - Identification ou mieux meilleure définition de la TCB.
    - Bénéfices de la réutilisation,
  - Permettre la maintenabilité,
  - Faciliter l'usage des Méthodes Formelles de manière à pouvoir les mettre dans les mains des développeurs,

Components

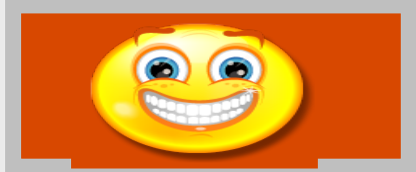
- Building complex systems by composing a small number of types of components is essential for any engineering discipline.
- This confers numerous advantages such as mastering complexity, enhanced productivity and correctness through reuse
- Component composition orchestrates interactions between components. It lies at the heart of the system integration challenge.



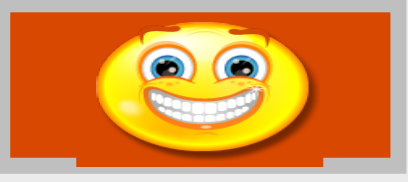
Joseph Sifakis – Extrait du séminaire Gérard Berry du 4 mars 2015

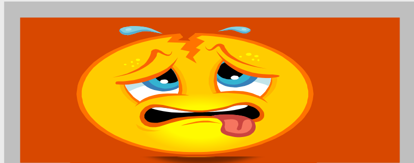
Components – Correctness-by-Construction

Composability rules guarantee that adding new components does not jeopardize essential properties of integrated components



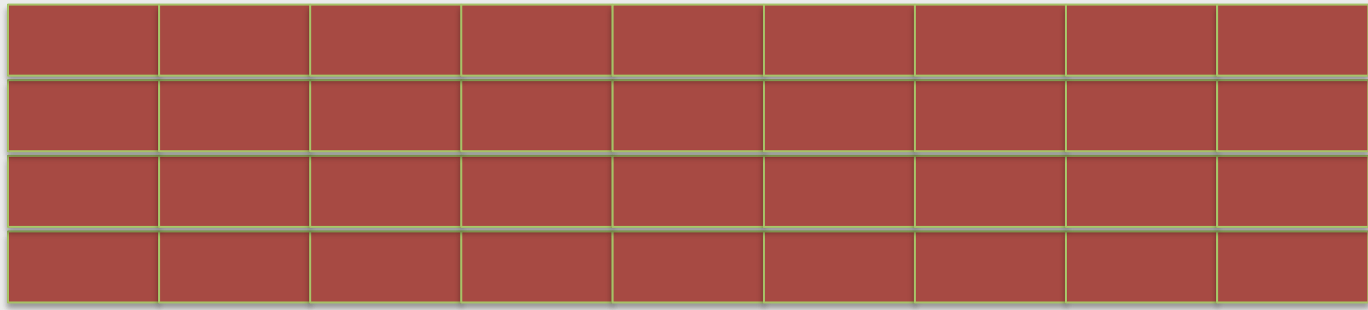
Feature interference in OS, middleware, telecommunication systems and web services are all due to lack of composability!



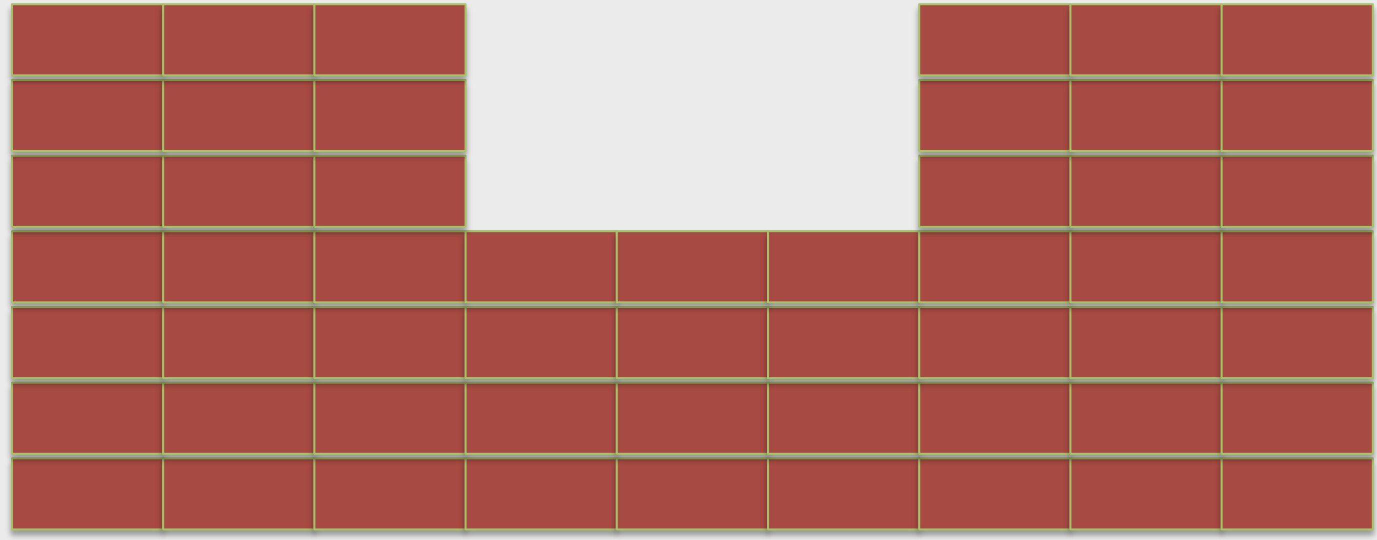


Joseph Sifakis – Extrait du séminaire Gérard Berry du 4 mars 2015

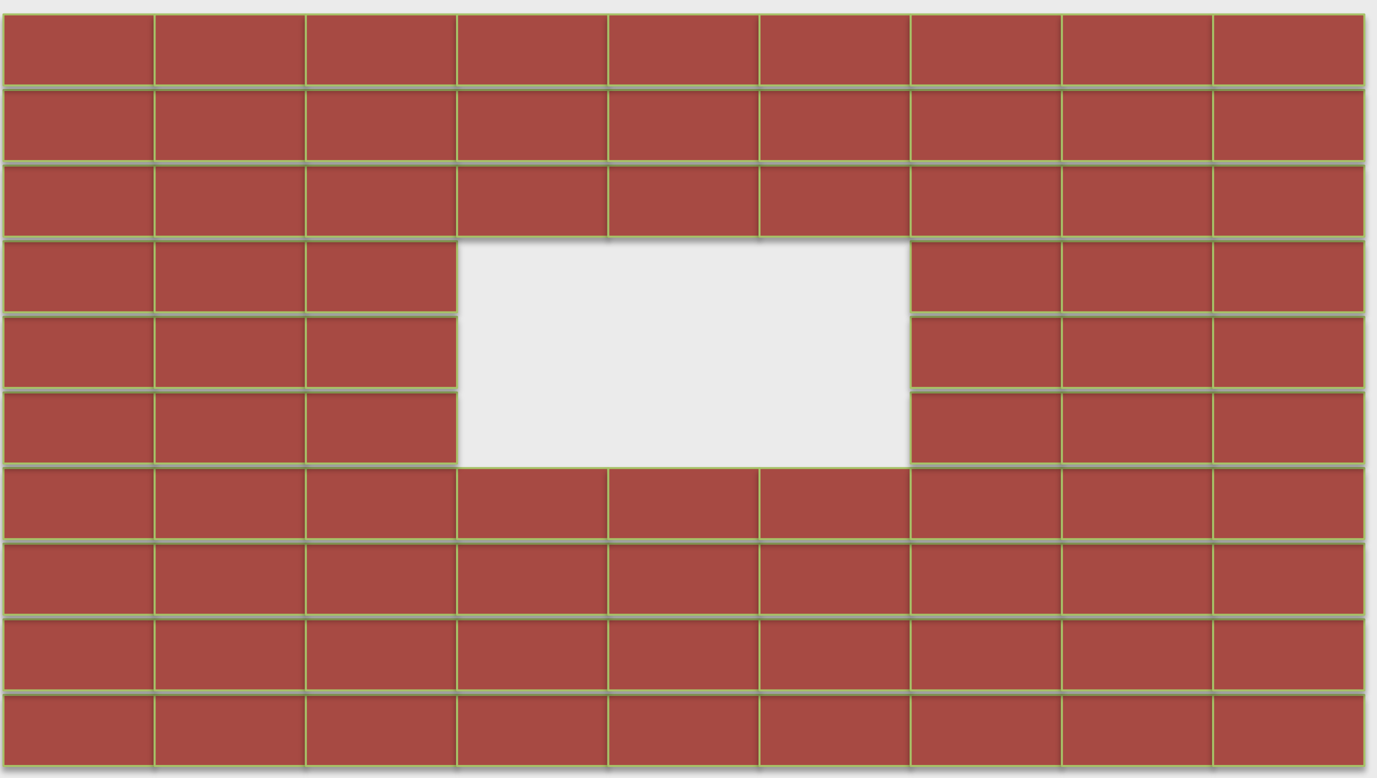
Composition



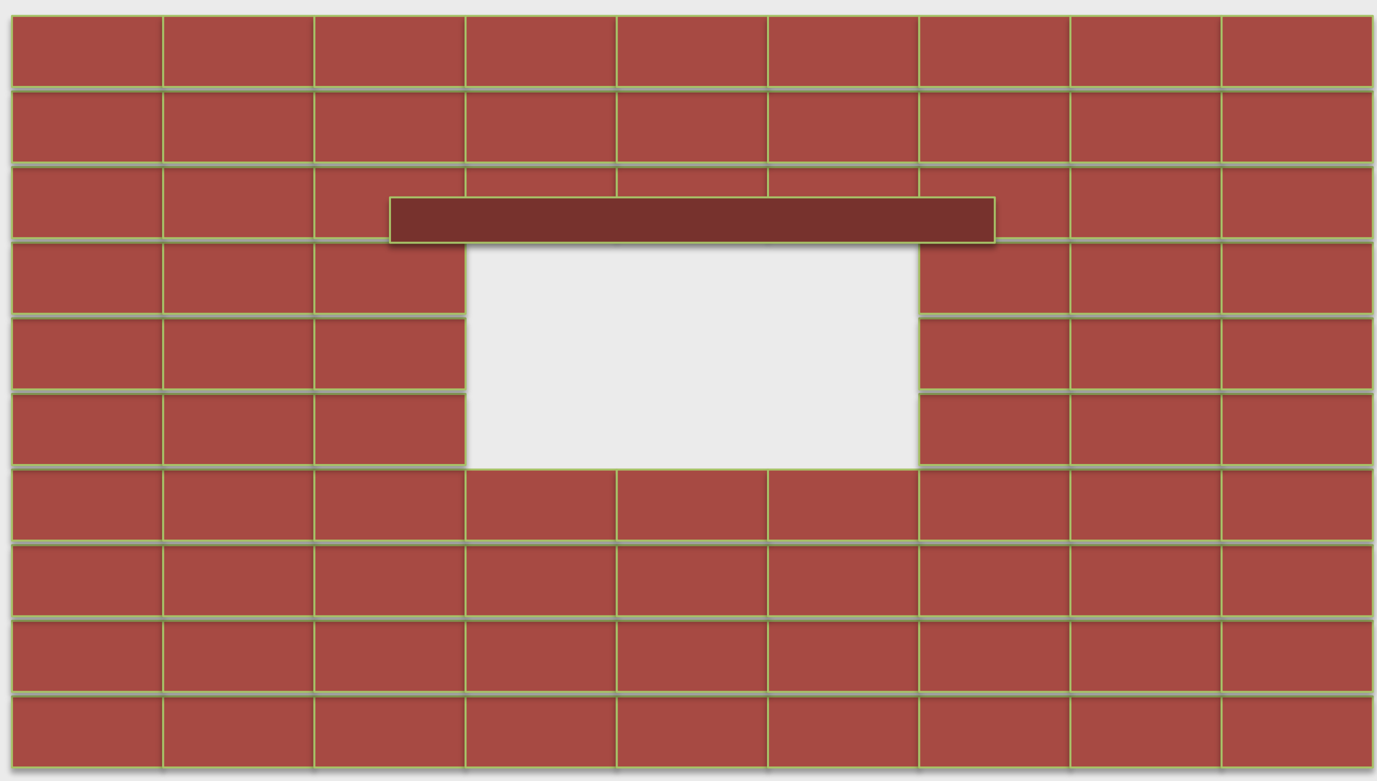
Composition



Composition



Composition



Processus d’identification des besoins Stratégie d’application (1/2)

- **Expériences, élaborés petit à petit.**
  - **En utilisant de nombreuses approches formelles sur de nombreux cas réels en entreprise sur de nombreuses années.**
  - **Dans des contextes où il fallait justifier à chaque fois l’applicabilité et l’intérêt pour le projet.**
- **Stratégie (1/2):**
  - **Choisir les situations dans lesquelles le ratio bénéfice/coût est très favorable et le marché représentatif.**
  - **Réduction du coût (micronoyau, …).**
    - Identification ou mieux meilleure définition de la TCB.
    - Bénéfices de la réutilisation,
  - **Permettre la maintenabilité,**
  - **Faciliter l’usage des Méthodes Formelles de manière à pouvoir les mettre dans les mains des développeurs,**

Processus d'identification des besoins Stratégie d'application (2/2)

- **Stratégie (2/2):**
  - **Maximiser les bénéfices en ciblant les domaines pour lesquels l'assurance est clé,**
    - Sécurité mobile
      - Importance des attaques logicielles,
      - Apporter un gain de confiance significatif,
      - Vérification formelle de la TCB,
      - Identifier et capturer formellement les propriétés d'intérêt
    - Aéronautique
    - Automobile (part de l'informatique, voiture connectée, voiture sans conducteur)
    - Domotique, gestion des bureaux
    - Systèmes médicaux, énergie (smart-grid, usine du future 4.0), …
  - **Permettre la certifiabilité,**

Spécificités du langage SMART

- Conçu pour répondre aux besoins identifiés dans l'application à grande échelle des méthodes formelles :
  - Utilisable par les développeurs, dans les niveaux d'abstractions élevés, mais aussi et surtout pour les niveaux les plus bas,
  - Permet aux développeurs de retrouver (et de s'appuyer sur) les paradigmes qu'ils utilisent habituellement que ce soit pour (bien) développer, ou pour déboguer, ou pour tester.
  - Permet aux développeurs de se poser les bonnes questions lors de l'écriture, et de pouvoir les formaliser facilement.

- Langage (et méthode sous-jacente) conçu par Dominique Bolignano
  - Etendu par Stéphane Lescuyer et Florence Plateau (polymorphisme, types structurés, modularité, …),
  - Sémantique formelle : Stéphane Lescuyer.

Spécificités du langage SMART

- Fonctionnel (manipulation de valeurs) mais avec un style impératif,
- Utilisable à différents niveaux d'abstraction,
- Les fonctions ne sont pas forcément totales,
- Possibilité d'associer les obligations de preuves à des chemins logiques dans le graphe d'exécution.
  - Preuves présentées et guidées comme du debugging symbolique,
  - Facilité la certification,
- Propriétés peuvent être exprimées comme des tests,
  - Invariants aussi exprimables comme des programmes ou des tests,
- Possibilité d'utiliser des modèles/programmes pour prouver,

Prédicats

```
public member(elt x, seq e) -> [true, false]
program {{ seq f, elt y }}
{
    f := e;
    while
    {
        ?removeFirst(y+, f, f+);
        if x = y; then return true;
    }
}
```

Prédicats implicites

```
public removeFirst(elt x+, seq e, seq f+) -> [true, empty]
implicit program

public equals(elt x, elt y) -> [true, false]
implicit program

public equals(elt x+, elt y)
implicit program
```

Spécificités du langage SMART

- Fonctionnel (manipulation de valeurs) mais avec un style impératif,
- Utilisable à différents niveaux d'abstraction,
- Les fonctions ne sont pas forcément totales,
- Possibilité d'associer les obligations de preuves à des chemins logiques dans le graphe d'exécution.
  - Preuves présentées et guidées comme du debugging symbolique,
  - Facilité la certification,
- Propriétés peuvent être exprimées comme des tests,
  - Invariants aussi exprimables comme des programmes ou des tests,
- Possibilité d'utiliser des modèles/programmes pour prouver,

Spécificités du langage SMART

- Fonctionnel (manipulation de valeurs) mais avec un style impératif,
- Utilisable à différents niveaux d’abstraction,
- Les fonctions ne sont pas forcément totales,
- Possibilité d’associer les obligations de preuves à des chemins logiques dans le graphe d’exécution.
  - Preuves présentées et guidées comme du debugging symbolique,
  - Facilité la certification,
- Propriétés peuvent être exprimées comme des tests,
  - Invariants aussi exprimables comme des programmes ou des tests,
- Possibilité d’utiliser des modèles/programmes pour prouver,

Traitements par cas / structure de contrôle

```
public member(elt x, seq e) -> [true, false]
program {{ seq f, elt y }}
{
     f := e;
     while
     {
          ?removeFirst(y+, f, f+);
          if x = y; then return true;
     }
}
```

Traitements par cas / structure de contrôle

```
public member(elt x, seq e) -> [true,false]
program {{ seq f, elt y }}
[found:true]
{
    f :=e ;
    while
    {
        [empty :false]removeFirst(y+,f,f+) ;
        [true:found,false:true](x=y) ;
    }
}
```

Traitements par cas / structure de contrôle

```
public member(elt x, seq e) -> [true,false]
program {{ seq f, elt y }}
[found:true]
{
    f :=e ;
    while
    {
        [empty :false]removeFirst(y+,f,f+) ;
        [true:found,false:true](x=y) ;
    }
}
```

Séparation Controle / Données

```
public member(elt x, seq e) -> [true, false]
program {{ seq f, elt y }}
{
    f := e;
    while
    {
        ?removeFirst(y+, f, f+);
        if x = y; then return true;
    }
}
```

Cas impossibles / Propriétés locales associées

// programme

```
public lemma propx(elt x,seq e)
program
{
 empty(e) => !member(x,e);
}
```

// ou de manière équivalente:

```
public lemma propxa(elt x,seq e)
program
[OK:true]
{
    [false:OK]empty(e);
    [true:error,false:true]member(x,e);
}
```

Cas impossibles / Propriétés locales associées

// fragments de programme

// Propriété (1):

```
{
 empty(e+) => !?removeFirst(_, e, _);
}
```

// ou de manière équivalente:

```
{
    empty(e+);
    [true:error,empty:true]removeFirst(_x+,e,e+);
}
```

Spécificités du langage SMART

- Fonctionnel (manipulation de valeurs) mais avec un style impératif,
- Utilisable à différents niveaux d'abstraction,
- Les fonctions ne sont pas forcément totales,
- Possibilité d'associer les obligations de preuves à des chemins logiques dans le graphe d'exécution.
  - Preuves présentées et guidées comme du debugging symbolique,
  - Facilité la certification,
- Propriétés peuvent être exprimées comme des tests,
  - Invariants aussi exprimables comme des programmes ou des tests,
- Possibilité d'utiliser des modèles/programmes pour prouver,

Un exemple plus complet

// fragments de programme // Propriété (2)

```
{
    addLast(x,e,f+) ;
    g:=e;
    while
    {
        [empty :exit]removeFirst(y+,g,g+) ;
        [empty :error]removeFirst(z+,f,f+) ;
        [false:error](y=z) ;
    }
    [empty :error]removeFirst(z+,f,f+) ;
    [false:error](x=z) ;
    [true:error, empty:true]removeFirst(z+,f,f+) ;
}
```

Un exemple plus complet

```
// Propriétés 2
public lemma prop2(seq e, elt x)
program {{ seq f, seq g, elt y, elt z }}
{
    addLast(x,e,f+) ;
    g:=e;
    while
    {
        [empty :exit]removeFirst(y+,g,g+) ;
        [empty :error]removeFirst(z+,f,f+) ;
        [false:error](y=z) ;
    }
    [empty :error]removeFirst(z+,f,f+) ;
    [false:error](x=z) ;
    [true:error, empty:true]removeFirst(z+,f,f+) ;
}
```

Un exemple plus complet

```
public hypothesis prop1()
program {{ seq e}}
{
 empty(e+) => !?removeFirst(_, e, _);
}

public hypothesis prop2(seq e, elt x)
program {{ seq f, seq g, elt y, elt z }}
{
    addLast(x,e,f+) ;
    g:=e;
    while
    {
        [empty :exit]removeFirst(y+,g,g+) ;
        [empty :error]removeFirst(z+,f,f+) ;
        [false:error](y=z) ;
    }
    [empty :error]removeFirst(z+,f,f+) ;
    [false:error](x=z) ;
    [true:error, empty:true]removeFirst(z+,f,f+) ;
}
```

Exemples de propriétés à prouver

- Theorem0: removeFirst(x+,e,f+) => member(x,e) ;
- Theorem1: addLast(x,e,f+) => member(x,f) ;
- Theorem2: member(x,e) => addLast(y,e,f+) => member(x,f) ;

```
//Théorème 0:
public theorem Theorem0(seq e)
program {{ elt x }}
{
    [empty :false]removeFirst(x+,e,_) => member(x,e) ;
}


//Théorème 1:
public theorem Theorem1(elt x,seq e)
program {{ seq f }}
{
    addLast(x,e,f+) => member(x,f) ;
}

// Théorème 2:
public theorem Theorem2(elt x, elt y, seq e)
program {{  seq f }}
{
    member(x,e) => addLast(y,e,f+) => member(x,f) ;
}
```

Langage intermédiaire SMIL

```
//Théorème 0:
public theorem Theorem0(seq e)
program {{ elt x }}
{
    [empty :false]removeFirst(x+,e,_) => member(x,e) ;
}

public theorem Theorem0b(seq e)
program {{ elt x }}
[empty :true]
{
    removeFirst(x+,e,_);
    member(x,e) ;
}
```

Un exemple plus complet

SM Prover - CDF/src/default/Exemples.sm - Eclipse - /Volumes/SM/WorkSpace

Java | SM | SM Prover | Resource

Project Explorer

- src
  - default
    - Exemples.sm
      - Exemples
        - addLast(elt, seq, se
        - egale(elt, elt)
        - egale(elt+, elt)
        - elt
        - empty(seq)
        - empty(seq+)
        - member(elt, seq)
        - memberb(elt, seq)
        - prop1()
        - prop1a()
        - prop1c()
        - prop2(seq, elt)
        - propx(seq)
        - propxa(seq)
        - removeFirst(elt+, se
        - seq
        - Theorem0(seq)
        - Theorem0b(seq)
        - Theorem1(elt, seq)
        - Theorem2(elt, elt, s

Exemples.sm

```
        [empty :exit]removeFirst(y+,f,f+) ;
        [true:found,false:true](x=y) ;
    }
}

public member(elt x, seq e) -> [true, false]
program {{ seq f, elt y }}
{
    f := e;
    while
    {
        ?removeFirst(y+, f, f+);
        if x = y; then return true;
    }
}
public theorem Theorem0b(seq e)
program {{ elt x }}
[empty :true]
{
    removeFirst(x+,e,_);
    member(x,e) ;
}

//Théorème 0:
public theorem Theorem0(seq e)
program {{ elt x }}
{
    [empty :false]removeFirst(x+,e,_) => member(x,e) ;
}

public hypothesis prop1()
```

Proof | Hints | Outline

- JDM2 prover [14/18]
  - removeFirst [0/0]
  - egale [0/0]
  - egale [0/0]
  - addLast [0/0]
  - empty [0/0]
  - empty [0/0]
  - memberb [0/0]
  - member [0/0]
  - Theorem0b [0/1] (lemmas: 0 available, 1 provided)
    - No lemma available
    - Unfold of member(x, e) ——[false]——›
      - Unproved (configuration: depth 3, width 2)
        - removeFirst(x, e, _) ——›
        - removeFirst(y, f, f#2) ——[empty]——› [Unfold of member(x, e)]
  - Theorem0 [1/1] (lemmas: 1 available, 1 provided)
  - prop1 [0/0]
  - prop1a [1/1] (lemmas: 3 available, 1 provided)
  - propx [0/0]
  - propxa [1/1] (lemmas: 5 available, 1 provided)
  - prop1c [1/1] (lemmas: 6 available, 1 provided)
  - prop2 [0/5] (lemmas: 7 available, 0 provided)
  - Theorem1 [0/1] (lemmas: 7 available, 1 provided)
  - Theorem2 [0/1] (lemmas: 8 available, 1 provided)

Progress | Proof Servi

No operations to display at this time.

Problems | Tasks | Error Log

0 items

Description	Resource	Path	Lo

Un exemple plus complet

```
public theorem Theorem0b(seq e)
program {{ elt x }}
[empty :true]
{
    removeFirst(x+,e,_);
    member(x,e) ;
}
```

```
▶ memberb [0/0]
▶ Theorem0b [0/1] (lemmas: 0 available, 1 provided)
▶ Theorem0 [1/1] (lemmas: 1 available, 1 provided)
▶ prop1 [0/0]
▶ prop1a [1/1] (lemmas: 3 available, 1 provided)
▶ propx [0/0]
▶ propxa [1/1] (lemmas: 5 available, 1 provided)
```

Un exemple plus complet

```
public theorem Theorem0b(seq e)
program {{ elt x }}
[empty :true]
{
    removeFirst(x+,e,_);
    member(x,e) ;
}
```

memberb [0/0]
Theorem0b [0/1] (lemmas: 0 available, 1 provided)
No lemma available
Unproved (configuration: depth 3, width 2)
removeFirst(x, e, _) ——▹
member(x, e) ——[false]——▹
Theorem0 [1/1] (lemmas: 1 available, 1 provided)

Un exemple plus complet

```
public member(elt x, seq e) -> [true, false]
program {{ seq f, elt y }}
{
    f := e;
    while
    {
        ?removeFirst(y+, f, f+);
        if x = y; then return true;
    }
}
public theorem Theorem0b(seq e)
program {{ elt x }}
[empty :true]
{
    removeFirst(x+,e,_);
    member(x,e) ;
}
```

```
▶ egale [0/0]
▶ addLast [0/0]
▶ empty [0/0]
▶ empty [0/0]
▶ memberb [0/0]
▶ member [0/0]
▼ Theorem0b [0/1] (lemmas: 0 available, 1 provided)
    No lemma available
  ▼ Unfold of member(x, e) ——[false]——▹
    ▼ Unproved (configuration: depth 3, width 2)
        removeFirst(x, e, _) ————▹
          removeFirst(y, f, f#2) ——[empty]——▹   [Unfold of member(x, e)]
▶ Theorem0 [1/1] (lemmas: 1 available, 1 provided)
▶ prop1 [0/0]
▶ prop1a [1/1] (lemmas: 3 available, 1 provided)
▶ propx [0/0]
```

Variables indicées. Traces Logiques. Structuration Preuves. Congruence

```
// Theorem0 : Unfold
[empty:true]
{
        removeFirst(x+,e,_);
        {
                f :=e ;
                while
                {
                        [empty :error]removeFirst(y+,f,f+) ;
                        [true:empty,false :true](x=y) ;
                }
        }
}
```

Variables indicées. Traces Logiques. Structuration Preuves. Congruence

```
// Theorem0 : Unfold
[empty:true]
{
        removeFirst(x+,e,_);
        {
                f :=e ;
                while
                {
                        [empty :error]removeFirst(y+,f,f+) ;
                        [true:empty,false :true](x=y) ;
                }
        }
}
// b* -> [b] b*
```

Variables indicées. Traces Logiques. Structuration Preuves. Congruence

```
// Theorem0 : Unfold + Unrolled
[empty:true]
{
          removeFirst(x+,e,_);
          {
                    f :=e ;
                    [empty:error]removeFirst(y+,f,f+) ;
                    [true:empty, false:true](x=y) ;
                    while
                    {
                              [empty :error]removeFirst(y+,f,f+) ;
                              [true:OK, false:true](x=y) ;
                    }
          }
}
```

Variables indicées. Traces Logiques. Structuration Preuves. Congruence

Trace logique:

- [true]removeFirst(x1+,e1,g1+) ;
- [true]equals(f1+,e1);
- [empty:error]removeFirst(y1+,f,f+) ;

Fermeture transitive de la congruence

- [true]removeFirst(x1+,e1,g1+) ;
- [true]equals(e1+,e1);
- [empty:error]removeFirst(x1+,e1,g1+) ;

Variables indicées. Traces Logiques. Structuration Preuves. Congruence

Trace logique:

- [true]removeFirst(x1+,e1,g1+) ;
- [true]equals(f1+,e1);
- [true]removeFirst(y1+,f,f+) ;
- [false :true](x1=y1) ;

Fermeture transitive de la congruence

- [true]removeFirst(x1+,e1,g1+) ;
- [true]equals(e1+,e1);
- [true]removeFirst(x1+,e1,g1+) ;
- [false :error](x1=x1) ;

Variables indicées. Traces Logiques. Structuration Preuves. Congruence

```
// Theorem0 : Unfold + Unrolled
[empty:true]
{
        removeFirst(x+,e,_);
        {
                f :=e ;
                [empty:error]removeFirst(y+,f,f+) ;
                [true:empty, false:error](x=y) ;
                while
                {
                        [empty :error]removeFirst(y+,f,f+) ;
                        [true:OK, false:true](x=y) ;
                }
        }
}
```

Composition avec Synchronisation

```
// Theorem1 Unfold
[OK:true]
{
1:          [empty :OK]addLast(x,e,f+);
            {
2:                      g:=f ;
                        while
                        {
3:                                  [empty:false]removeFirst(y+,g,g+);
4:                                  [true:OK, false:true](x=y);
            }
            }
}
```


Rappel Theorem1: addLast(x,e,f+) ; => member(x,f) ;

Composition avec Synchronisation

```
// fragments de programme  // Propriété (2)
{
1:        addLast(x,e,f+) ;
          while
          {
2:                  [empty :exit]removeFirst(y+,e,e+) ;
3:                  [empty :error]removeFirst(z+,f,f+) ;
4:                  [false:error](y=z) ;
          }
5:        [empty :error]removeFirst(z+,f,f+) ;
6:        [false:error](x=z) ;
7:        [true:error,empty :true]removeFirst(z+,f,f+) ;
}
```

Composition Theorem1Unfold Axiom2

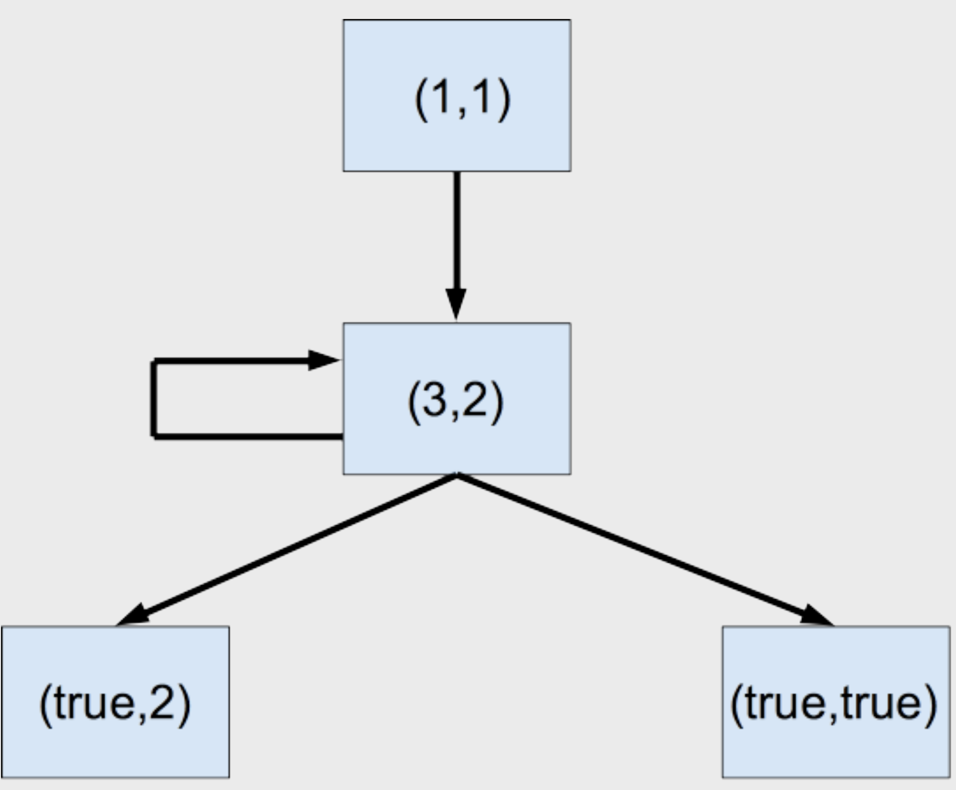
(1,1)

(3,2)

(3,true) | (true,2) | (true,true) | (error,2) | (error,true)

3 | true | error

Simplification Composition Theorem1Unfold Axiom2 (propagation congruence)

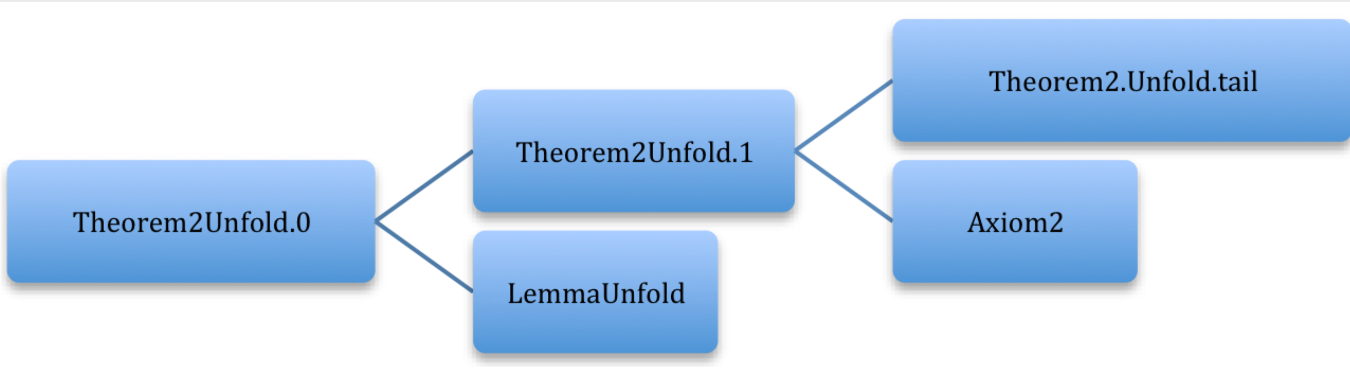


Composition avec Synchronisation

```
// fragments de programme  // Propriété (2)
{
1:      addLast(x,e,f+) ;
        while
        {
2:              [empty :exit]removeFirst(y+,e,e+) ;
3:              [empty :error]removeFirst(z+,f,f+) ;
                [false:error](y=z) ;
        }
4:      [empty :error]removeFirst(z+,f,f+) ;
5:      [false:error](x=z) ;
6:      [true:error,empty :true]removeFirst(z+,f,f+) ;
}
```

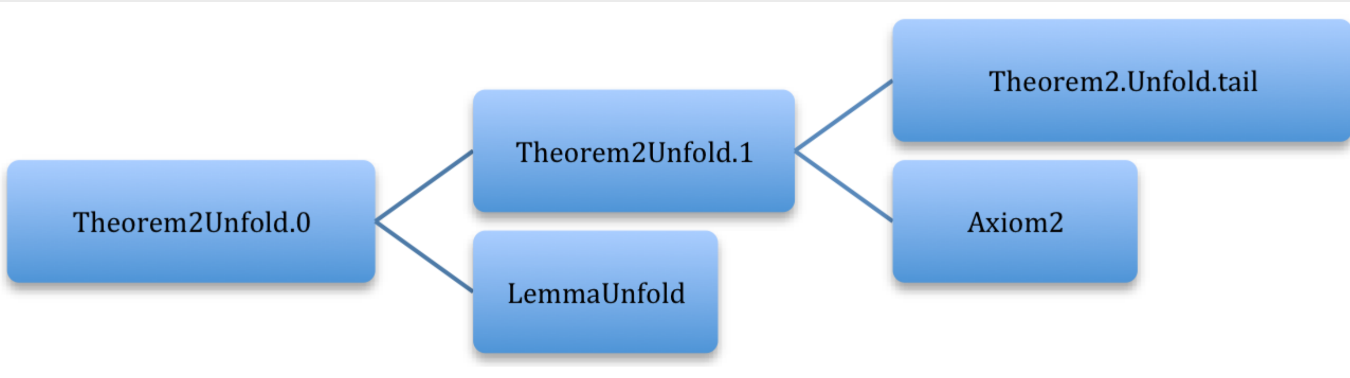
Theorem2: member(x,e); addLast(y,e,f+); => member(x,f) ;

Nouvel exemple de (double) composition pour la preuve du théorème 2



LemmaUnfold : member(x,e) => member(x,e) ;

Nouvel exemple de (double) composition pour la preuve du théorème 2



LemmaUnfold : [false:true]member(x,e) ;

Spécificités du langage SMART

- Fonctionnel (manipulation de valeurs) mais avec un style impératif,
- Utilisable à différents niveaux d'abstraction,
- Les fonctions ne sont pas forcément totales,
- Possibilité d'associer les obligations de preuves à des chemins logiques dans le graphe d'exécution.
  - Preuves présentées et guidées comme du debugging symbolique,
  - Facilité la certification,
- Propriétés peuvent être exprimées comme des tests,
  - Invariants aussi exprimables comme des programmes ou des tests,
- Possibilité d'utiliser des modèles/programmes pour prouver,

Spécificités du langage SMART

- Fonctionnel (manipulation de valeurs) mais avec un style impératif,
- Utilisable à différents niveaux d'abstraction,
- Les fonctions ne sont pas forcément totales,
- Possibilité d'associer les obligations de preuves à des chemins logiques dans le graphe d'exécution.
  - Preuves présentées et guidées comme du debugging symbolique,
  - Facilité la certification,
- Propriétés peuvent être exprimées comme des tests,
  - Invariants aussi exprimables comme des programmes ou des tests,
- Possibilité d'utiliser des modèles/programmes pour prouver,

Conclusion (1/6) : Un rationnel complet

- Permet d'identifier un rationnel complet organisant les raisons qui nous permettent de penser qu'un système a les bons comportements et propriétés,
  - Certaines parties sont formelles,
  - D'autres pouvant être informelles mais analysable avec un bon degré de confiance.
- Possibilité d'associer n'importe quelle faille à une ou plusieurs erreurs dans le rationnel

Conclusion (2/6): Fonction d’abstraction

- Chaque état concret bien fondé et chaque comportement a une correspondance dans la vue plus abstraite.
- Utilisé pour le raffinement mais aussi potentiellement pour l’expression de propriétés.

Conclusion (3/6) : Propriétés

- Confidentialité, intégrité, authentification, et séparation, …
- Modèles versus propriétés.
- Redondance.

Conclusion (4/6) : Ce que nous gagnons en terme de coût ?

- Un micro-noyau peut-être fait à cout raisonnable,
- La (gestion de la) preuve s'apparente de l'analyse d'exécution symbolique (et ainsi d'une certaine manière à du debugging) symbolique,
- L'expression de propriétés peut se ramener à du test symbolique,
- Il est crucial de réutiliser autant que possible les briques de bases pour la définition d'architecture de sécurité (e.g. micro-noyau)
  - sans modification,
  - Ou après adaptation dans de rares cas.

Conclusion (5/6): Que gagne-t-on en terme de temps de développement ?

- Compatible avec le prototypage rapide quand certains modèles utilisés pour le raffinement peuvent être utilisés,
- La preuve de certaines propriétés peuvent être différé sans empêcher l'utilisation du code,

Conclusion (6/6): Que gagne-t-on en terme de maintenabilité ?

- Quand les modifications sont mineures l'outil peut généralement retrouver plus facilement les preuves automatiquement en s'inspirant des preuves précédentes,
- Certaines modifications ne concernent que les niveaux bas et laissent inchangés les niveaux les plus abstraits.
- La preuve et la modélisation induisent une prise de recul qui permet in fine d'utiliser des architectures plus stables vis-à-vis de la maintenance.

Conclusions

- Utilisation possible par une large communauté de développeurs (contour précis à valider),
- Le problème n’est pas tant la correction que la confiance.
- Ce n’est pas parce que tout n’est pas modélisable ou prouvable (hypothèses, résistance aux attaques physiques, adéquation des propriétés, architectures inadaptées, maillons humains, …) que l’utilisation des méthodes formelles n’est pas LA bonne réponse aux enjeux de sécurité et de confiance que posent les nouvelles architectures ouvertes,

MERCI POUR VOTRE ATTENTION.

DES QUESTIONS?

Prove & Run S.A.S.
dominique.bolignano@provenrun.com
77, avenue Niel, 75017 Paris, FRANCE

Copyrights

- Toutes les marques et logos utilisées dans ce document appartiennent à leurs propriétaires respectifs.