Programmation synchrone

Logiciels embarqués
temps réel critiques

Plan

Logiciel embarqué ?

Logiciel embarqué = Logiciel faisant partie d'un système

- Traitement de texte, tableur, système d'exploitation (Windows, Unix), logiciel mathématique, …

≠

- Téléphone portable, baladeur, ordinateur de bord (voiture, avion), …

Logiciel critique ?

Intuitivement, un système critique est un système dont la défaillance peut avoir des impacts graves:

- Nucléaires
- Transports
  - ✓ Automobile
  - ✓ Ferroviaire
  - ✓ Aéronautique
  - ✓ Spatial
- …

Classification des logiciels

Exemple de la norme aéronautique DO178B
(autre exemple: ECSS dans le spatial)

Catégorie **Effet de la défaillance**

A	Catastrophique	(perte de vies humaines)
B	Dangereux	(destruction de biens, blessures graves)
C	Majeure	(indisponibilité ou perte du système)
D	Mineure	(sans conséquence sur le système)
E	Sans effet	

Classification des logiciels

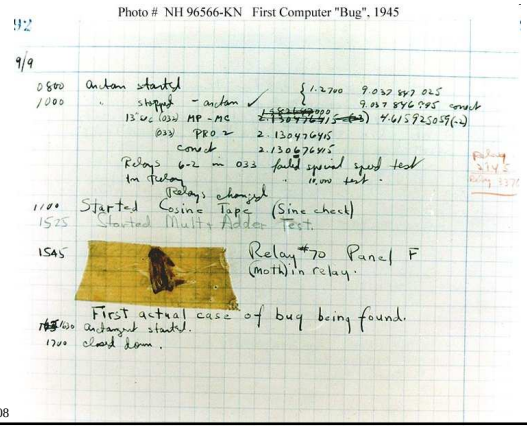
Probabilité acceptable de défaillance en fonction de la catégorie du logiciel

Mineure				
Majeure			**situation**	**acceptable**
Dangereux	**situation**	**inacceptable**		
Catastrophique	10^{-3} / heure	10^{-6} / heure	10^{-9} / heure	10^{-12} / heure
Conséquences / Probabilités	**Probable**	**Rare**	**Extrêmement rare**	**Extrêmement improbable**

Exemple de validations requises contractuellement

Méthode de vérification	cat. A & B	cat. C	cat. D	cat. E
Matrice de vérification	exigée	exigée	recom-mandée	non
Plan de vérification	exigé	exigé	recommandé	non
Dossier de couverture	exigé	exigé	recommandé	non
Revues et analyses indépendantes	au moins un	recomman dée	non	non
Tests 100%	exigés sur moyens cibles	exigés	recom-mandés	non
Service équivalent rendu	non	à négocier	à négocier	oui

Example 1: The First "Computer Bug"



Example 2: The Patriot Missile Failure

On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people. A report of the General Accounting office, GAO/IMTEC-92-26, entitled *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia* reported on the cause of the failure. It turns out that the cause was an inaccurate calculation of the time since boot due to computer arithmetic errors.

Autres impacts

En plus des impacts sur la vie humaine, prendre en compte

- L'aspect financier
  - ✓ Perte matériel, correction de bug
  - ✓ Rappel de matériel (exemple automobile)
- L'aspect image de marque
  - ✓ « Bug » numérique d'Intel

⇨ **Les techniques utilisées pour les logiciels sont utilisables dans d'autres domaines**

Temps réel ?

- Systèmes **transformationnels**
  - ✓ Entrées disponibles en début d'exécution
  - ✓ Sorties fournies en fin d'exécution
- Systèmes **interactifs**
  - ✓ Réagit à son environnement
  - ✓ A sa propre vitesse
- Systèmes **réactifs**
  - ✓ Réagit à son environnement
  - ✓ A une vitesse imposée par l'environnement

Réactifs ?

Différentes façons de « réagir » à son environnement:

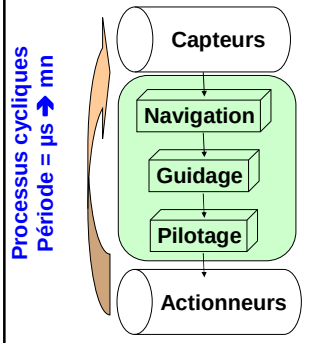
- Systèmes événementiels
  - ✓ Réception d'événements ponctuels
  - ✓ Réponse par des traitements ponctuels
- Systèmes à flots de données
  - ✓ Réception continue de données
  - ✓ Traitement permanent
- Systèmes temps réel
  - ✓ Le temps est une variable du système

Certains systèmes ont des composants dans les trois catégories

Il existe plusieurs « temps réel »

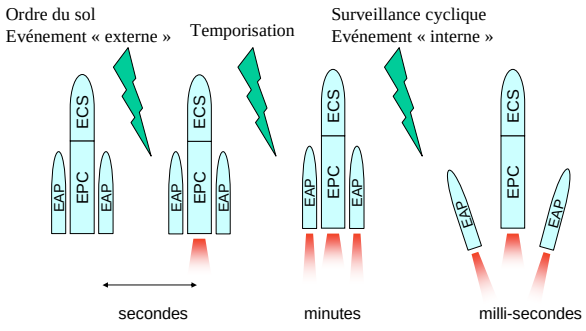
- Temps réel « mou »
  - ✓ Demande de réactivité « flexible »
  - ✓ Environnement complexe très indéterministe
  - ✓ Exemple: Réseau téléphonique
- Temps réel « dur »
  - ✓ La non tenue du temps réel est primordiale
  - ✓ Hypothèse stricte sur l'environnement
  - ✓ Exemple: Contrôle commande d'un véhicule

Flots de données, exemple Contrôle / commande véhicule



- **Acquisition des mesures**
- **Où suis je ?**
- **Où dois je aller ?**
- **Calcul des commandes**
- **Commandes vers les actionneurs**

Système événementiel, exemple Phases de vol d'Ariane 5

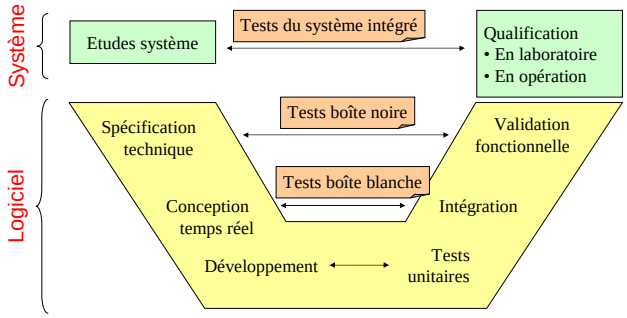


Logiciel temps réel critique

- Logiciel
  - ✓ Embarqué
  - ✓ Critique
  - ✓ Réactif
  - ✓ Temps réel critique

➔ Comment développer des logiciels temps réel critique?

Développement classique



Coût de développement d'un logiciel critique

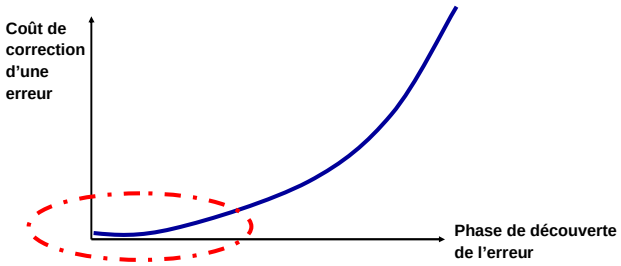
- Spécification 10%
- Conception 10%
- Développement/TU 25%
- Tests intégration 5%
- **Validation 50%**

Augmenter les efforts en début de cycle

Observation:

Plus une erreur est détectée tard dans le cycle de développement, plus elle est coûteuse à corriger

Coût de correction d'une erreur



Mettre plus d'efforts sur les phases amonts

Développement basé sur les modèles

Objectif de la spécification logicielle

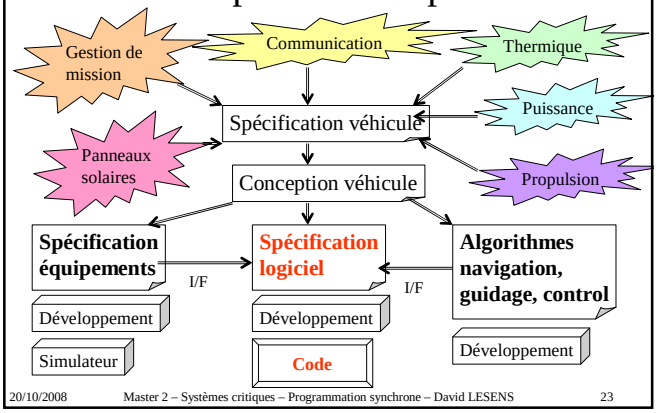
- Capturer le besoin système
  - ✓ Spécialistes métiers
- Servir d'entrée à l'activité de développement
  - ✓ Cohérence
  - ✓ Complétude
- Référence pour la validation fonctionnelle
  - ✓ Exigences validables

➔ **Utilisation de modèles formels**

1er objectif d'une spécification logicielle

- Formalisation du besoin
  - ✓Standard de communication
    - ❖Pour des informaticiens
    - ❖Pour des non informaticiens
  - ✓Différents types d'application
    - ❖Synchrone et/ou
    - ❖Asynchrone et/ou
    - ❖Algorithmique

Exemple dans le spatial



Exemple de mauvaise compréhension catastrophique

(CNN) -- NASA lost a $125 million Mars orbiter because one engineering team used metric units while another used English units for a key spacecraft operation.

For that reason, information failed to transfer between the Mars Climate Orbiter spacecraft team in Colorado and the mission navigation team in California.

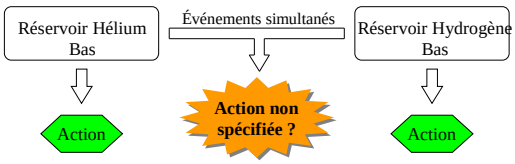
NASA's Climate Orbiter was lost September 23, 1999

2nd objectif d'une spécification logicielle

- Détecter les erreurs en phase amont de développement
  - ✓Validation de la spécification
    - ❖Cohérence de la spécification
    - ❖Complétude de la spécification
    - ❖Preuve sur la spécification
  - ✓Test
    - ❖Prototypage rapide
    - ❖Simulation de la spécification

Exemple d'incomplétude (1)

Exemple Ariane 5



Exemple 2: Un compteur. Spécification formelle?

Calculer un compteur

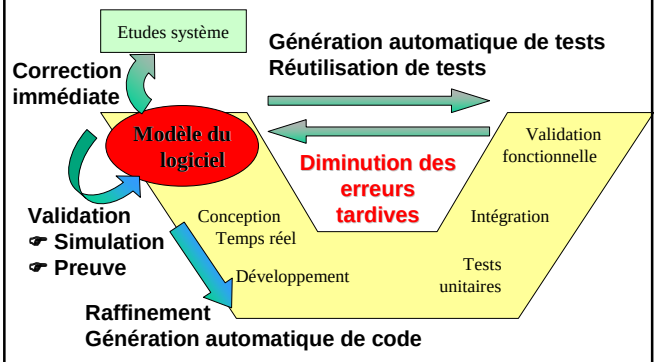
- Incrémenter à chaque cycle de « inc »
- Reseter le compteur sur ordre « reset »
  - **Valeur initiale du compteur?**
  - **Faut-il incrémenter au 1er cycle?**

0	**1**	**2**	**3**	**4**	**5**	**?**
1	**2**	**3**	**4**	**5**	**6**	**?**

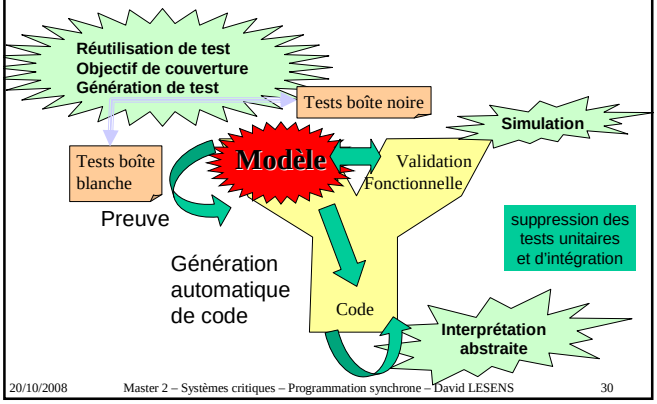
3ième objectif d'une spécification logicielle

- Faciliter le raffinement de la spécification vers une conception
  - ✓ Réutilisation des tests de simulation de la spécification
  - ✓ Formalisation de la conception
  - ✓ Génération de code
    - ❖ Séquentiel ou multitâche
    - ❖ Langage cible
    - ❖ Embarquable / qualité

Utilisation d'un modèle formel



Génération automatique de code



MDE

L'approche modélisation formelle est compatible de MDE

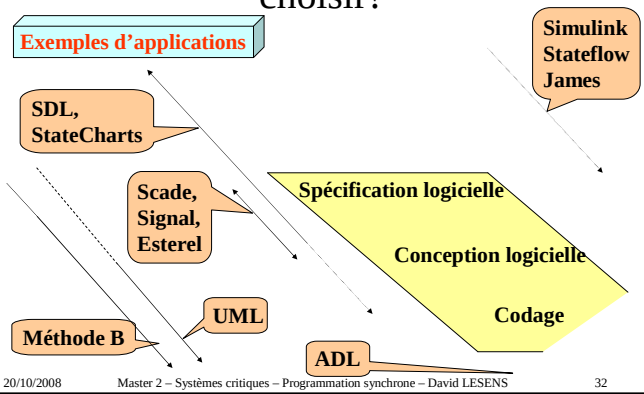
- ✓ Model Driven Engineering (MDE)
- ✓ Model Driven Approach (MDA)

➔ Regrouper toutes les informations dans un modèle
Diagramme de séquence, diagramme de classe, automates, codes, tests, documentation…

➔ Souvent lié à UML

- ✓ Unified Modelling Language (UML)

Quelles méthodes « formelles » choisir?



Logiciel temps réel critique

- Logiciel
  - ✓ Embarqué
  - ✓ Critique
  - ✓ Réactif
  - ✓ Temps réel critique

  } **Relatifs aux besoins**
- Utilisation de méthodes formelles — **Outillage**
- Langages synchrones
  - ➔ Hypothèse synchrone

  Choix de conception

Les langages synchrones

Principalement deux langages « industriels » synchrones

- SCADE
- ESTEREL

Plan

Synchronous model

Synchronous languages

Semantics = synchronous hypothesis

- **Existence of a global clock**
  - Software cyclically activated
  - Inputs read at the cycle beginning
  - Outputs delivered at cycle end
    (read / write forbidden during the cycle)
- **The cycle execution duration shall theoretically be null**
  - ➔ **No cycle overflow**

Features

- **Non blocking communication in null duration**
  - Trivial composition
- **Determinism**
  - An input sequence produces always the same output sequence
  - Ensure the repetitiveness of tests

  ➔ Required by DO248 (extension of DO178B)
- **Bounded memory**
  - No dynamic memory allocation

  ➔ Required by DO248 (extension of DO178B)

Introduction to the Scade language

SCADE

- **SCADE**
  - "Safety Critical Application Development Environment"
- o A textual language: Lustre
  - Formal language for reactive synchronous system
- o A graphical language
  - Semantics equivalence SCADE ⇔ Lustre
  - Adapted to data flow and automata
- o A software toolbox
  - Graphical editor, simulator, proof tool
  - Automatic documentation and certified code generation

Time in Scade

- **Global clock (known by all processes)**
  - o Time = discrete sequence of tick t_0, t_1, t_2, etc.
  - o At each tick t_i a cycle is running
- **Variable = flow which takes at each tick a unique value**

Example: integer variable x

	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5

Operators

- **An operator acts on flows of values (and not on values)**

Example

- o Operator « + »: int_n x int_n → int_n

	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5
x + x	10	16	4	6	26	10

"IF" operator

x = if b then y else z

If "b" is true, "x" takes the value "y",
else, "x" takes the value "z"

Note:

Does not mean
If "b" is true, execute "y",
else, execute "z"

} See automata or clocks for control flow specification

“Mutual exclusion” operator

#: bool x bool x … x bool → bool

n times

Returns **true** if **at most** one of its inputs is true

e1	e2	e3	#(e1, e2, e3)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Temporal operators

- The “PRE” operator takes as input a data flow (i.e. a variable) and returns its value at the previous tick.

  At initial tick, its value is undefined.
- The “→” operator takes as input an initialisation value and a data flow of the same type. It returns an identical data flow, except for the initial value.

Example

	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5
PRE x	null	5	8	2	3	13
9 -> x	9	8	2	3	13	5
9 -> PRE x	9	5	8	2	3	13

“Follow by” operator

FBY(x, n, init) = init → (PRE (PRE … x))

n times

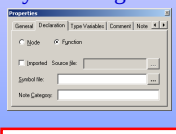
	t_0	t_1	t_2	t_3	t_4	t_5
x	5	8	2	3	13	5
9 -> PRE x	9	5	8	2	3	13
FBY(x,3,9)	9	9	9	5	8	2

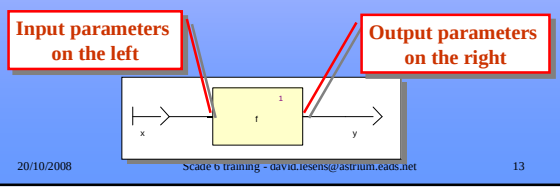
Node and function

y = f(x)

Function and nodes are represented by a rectangle

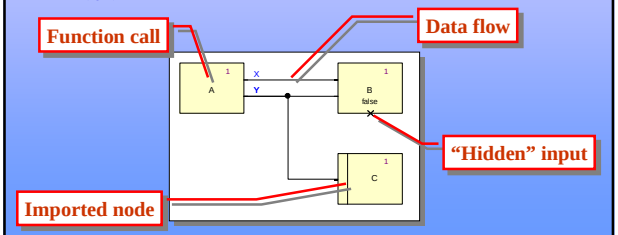
- A node has an internal state
- A function has **no** internal state



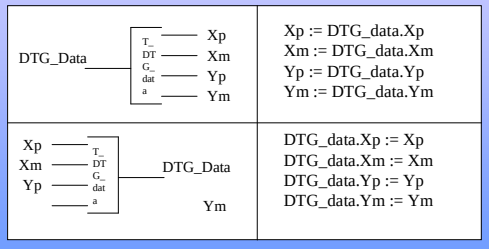


Data flows

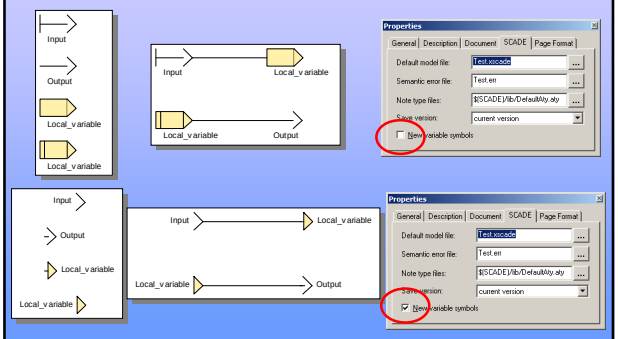
(x, y) = A();
B(x, y);
C(y)

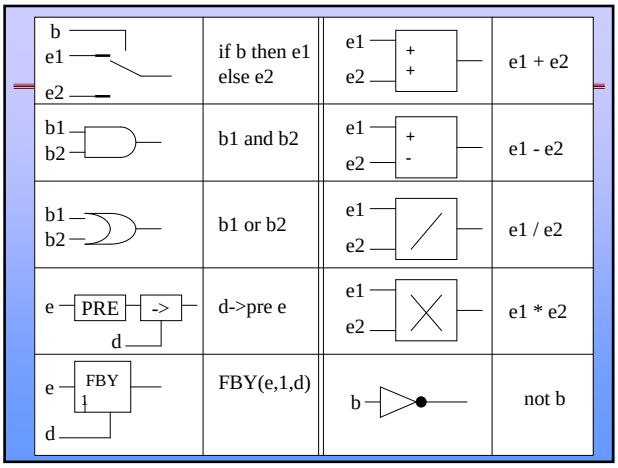


Data structure



Variables representation

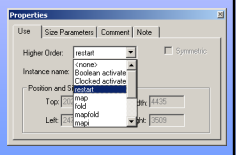




Activation conditions

Activation conditions

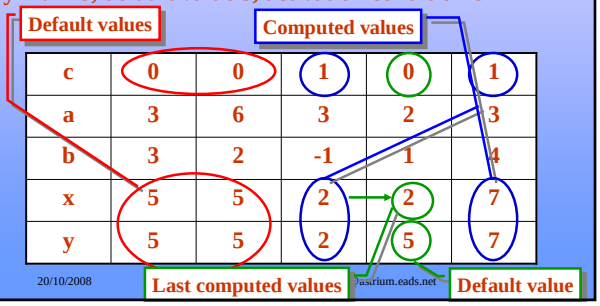
- **Activation condition**
  - Condition true = Block activated
  - Condition false = Previous outputs used (was “condact” in Scade 5)
    or Default values
  - Init values before first use



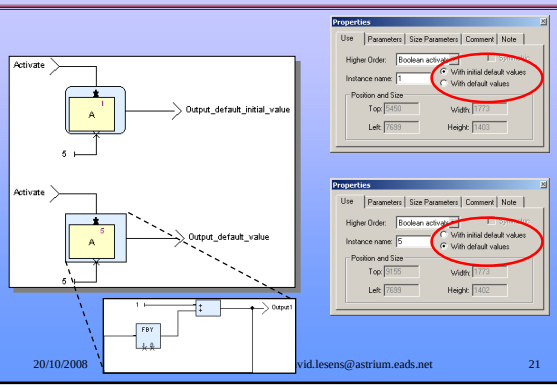
Activation: Example (1/3)

x = a + b, initial default value 5, activation condition c

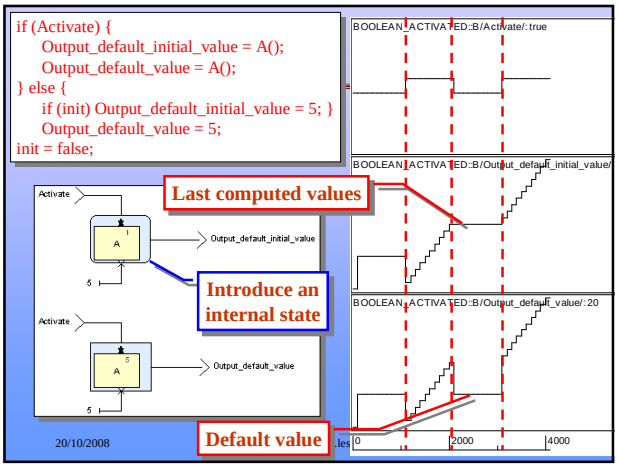
y = a + b, default value 5, activation condition c



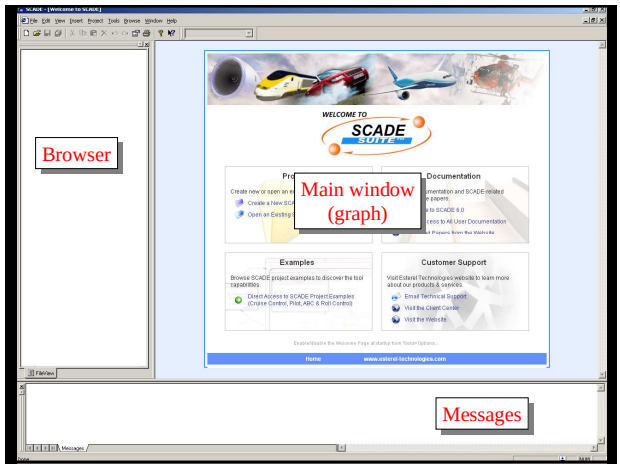
Activation: Example (2/3)



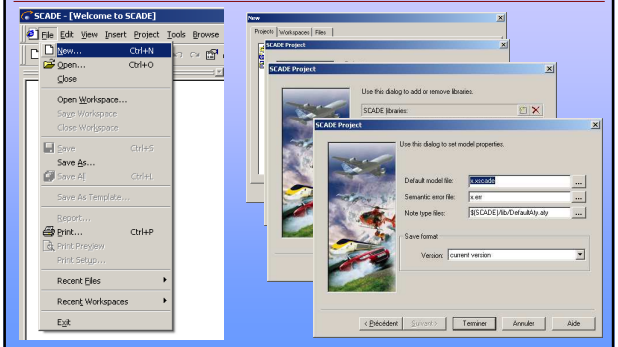

```
if (Activate) {
    Output_default_initial_value = A();
    Output_default_value = A();
} else {
    if (init) Output_default_initial_value = 5; }
    Output_default_value = 5;
init = false;
```

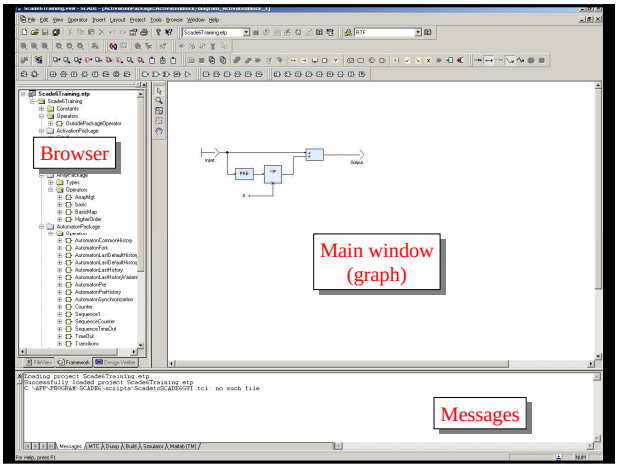


Editing a Scade model

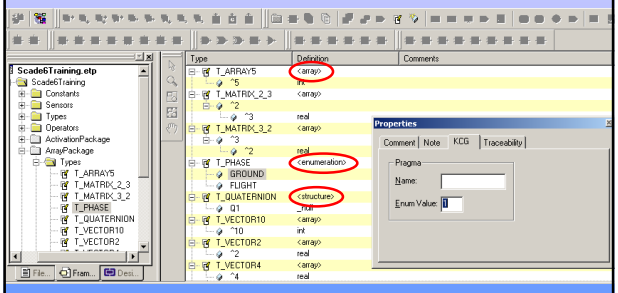


Creating a new project

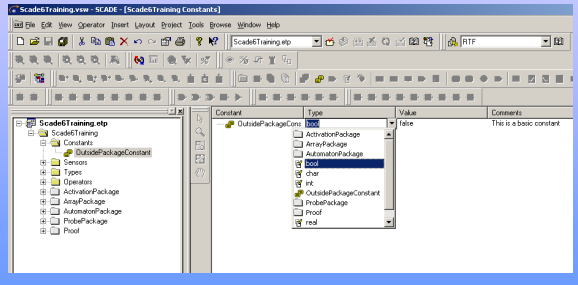




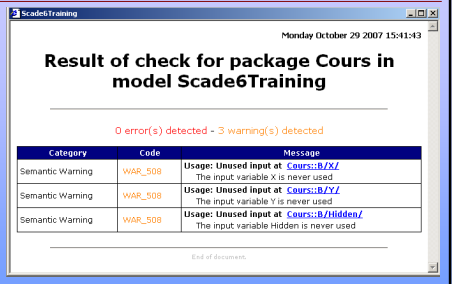
Management of types



Defining a constant



Quick check



The quick check performs syntax and semantics verification

It shall be frequently used

Documentation

- Welcome to SCADE 6.0
- Getting Started with SCADE
- Scade Language Tutorial
- Scade Language Primer
- Scade Language Reference Manual
- SCADE User Manual
- SCADE Technical Manual
- SCADE Libraries Manual
- SCADE UML Metamodel Card
- SCADE Gateway for Rhapsody Guidelines
- Simulink™ Gateway Guidelines
- Simulink™ Modeling Guidelines
- RTOS Wrapper Guidelines

About Requirements Management Gateway documentation, check from RMG interface at Help > Documentation or Coupling Notes

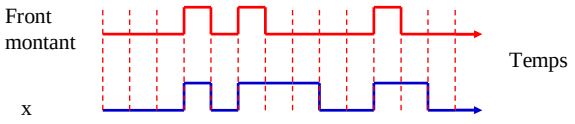
Plan

Exercices flots de données

- Front montant
- Compteur
- Compteur étendu
- Front montant étendu

Front montant (1)

- Un front montant est un opérateur prenant un booléen en entrée
- Il renvoie vrai dès que son entrée passe de l'état faux à l'état vrai

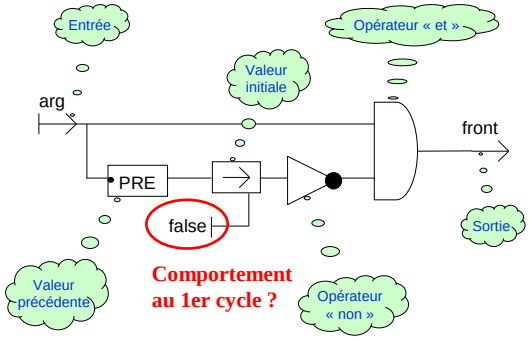


Front montant (2)

- Modéliser un opérateur « Front montant » en SCADE
- La spécification textuelle est-elle complète?
- Valider cet opérateur par simulation

✓ **Sortie lorsque l'entrée est vraie au 1er instant ?**

Front montant – solution



Compteur (1)

Calculer un compteur

Le compteur accepte deux entrées

- ✓ « inc » (incrément) de type entier
- ✓ « reset » de type booléen

il renvoie une sortie entière

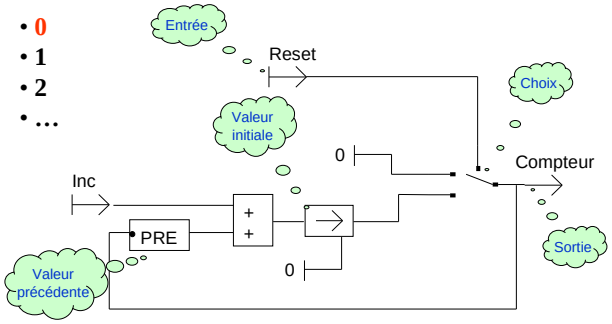
- ➢ Incrémenter à chaque cycle de « inc »
- ➢ Reseter le compteur sur ordre « reset »

Compteur (2)

- ➢ La spécification textuelle est-elle complète?
- ➢ Modéliser un opérateur « Compteur » en SCADE
- ➢ Valider cet opérateur par simulation
  - ✓ **Valeur initiale du compteur?**
  - ✓ **Faut-il incrémenter au 1er cycle?**

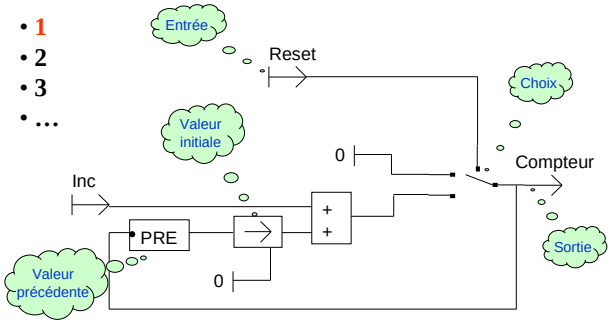
Compteur – solution 1

- **0**
- **1**
- **2**
- **…**



Compteur – solution 2

- **1**
- **2**
- **3**
- **…**



Compteur étendu

Calculer un compteur

Etendre « compteur » de la manière suivante:

- ✓Ajouter l'entrée booléen « start »
- ✓A l'initialisation et après un « reset », ne démarrer qu'à la réception du signal « start »

➔ Extension possible: un chronomètre

- ✓ Ajouter un bouton pour geler l'affichage

Front montant avec confirmation

Étendre « Front montant » de la manière suivante:

- ✓ Ajouter une entrée entière « confirmation »

➢ Ce nouvel opérateur renvoie vrai dès que son entrée passe de l'état faux à l'état vrai en y restant « confirmation » fois de suite

➢ Modéliser un opérateur « Front montant étendu »

➔ En « partant de zéro »

➔ En utilisant les opérateurs « Compteur étendu » et « Front montant »

Plan

Résumé

- Logiciel embarqué temps réel critique
  - ➔ Besoin
- Utilisation de méthodes formelles
  - ➔ Adaptées au besoin
  - ✓ Non interprétable (complet, cohérent), facile à comprendre, simulable, génération de code
- Sémantique synchrone
  - ➔ Choix de conception
  - ✓SCADE, ESTEREL ➔ ***Langages spécialisés***

validable

Validation logicielle

Logiciel correct
- Pas d'erreurs à l'exécution
- Satisfaction des contraintes temps réels
- Conformité des résultats

Solutions
- Relecture manuelle / analyse de code → Coûteux, Erreur humaine
- Test dynamique → Coûteux, Non exhaustif
  - Au niveau code
  - Au niveau modèle
- ~~Interprétation abstraite~~
- Preuve

Mais la preuve ne remplace jamais le test

Vérification sémantique (1)

Sémantique d'un modèle SCADE
- Syntaxe
- Vérification des types
  - ✓ Compatibilité des types
    - ❖ Exemple: « type position » = « type vitesse »
  - ✓ Compatibilité de nom
    - ❖ « x = y » est correct ⇔ « type x » = « type y »
- Pas de variables non initialisées
- Causalité temporelle

Causalité temporelle (1)

SCADE est un langage équationnelle
- L'ordre d'évaluation ne dépend que des flots de données

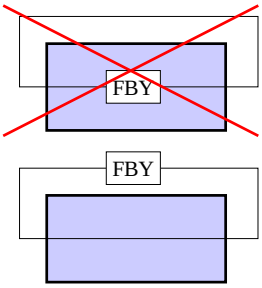
```
x = y;
y = z;   →  « y = z » évalué en premier
            « x = y » évalué en second

x = y;
y = z;   →  Calcul de l'ordre d'évaluation impossible
z = x;       « x = y = z = x = … »
```

Problème de causalité

Causalité temporelle (2)

Théoriquement, les deux constructions suivantes sont correctes



Interdit pour la génération automatique de code en mode « non expansée »

Vérification sémantique (2)

Lorsqu'un modèle SCADE a une sémantique correcte (« quick check »), le modèle est:

- Complet
- Cohérent
- Implémentable

➔ Les bonnes propriétés d'une spécification

➔ « Check sémantique » à effectuer systématiquement

Mais, le logiciel fait-il ce que l'on veut?

Qu'est-ce que le test ?

Comparer le comportement observé au comportement de référence

- Différents niveaux
  - ✓ Unitaire / intégration / fonctionnel / qualification système
  - ✓ Hôte / cible
  - ✓ Equipement réel / simulateur
  - ✓ Boîte blanche / Boîte noire

+ sur le modèle

(approche méthode formelle)

Activités liées au test

- Préparer les tests
  - ✓ Plan de test
  - ✓ Choisir / décrire / vérifier
- Exécuter les tests
  - ✓Exécuter
  - ✓Contrôler le résultat
- Couverture des tests
- Tests en non-régression

Même activités pour la simulation d'un modèle

Objectifs des tests unitaires

- Robustesse
  - ✓Absence de « plantage »
- Validité fonctionnelle
  - ✓Comparaison au résultat attendu ➔ **Observateur**
- Objectifs contractuels
  - ✓**Taux de couverture**
    - ❖Intuitivement satisfaisant
    - ❖Chiffrable
    - ❖Non exhaustif

Attention

Traduction non évidente pour un modèle SCADE

Tests unitaires : Taux de couverture maximal

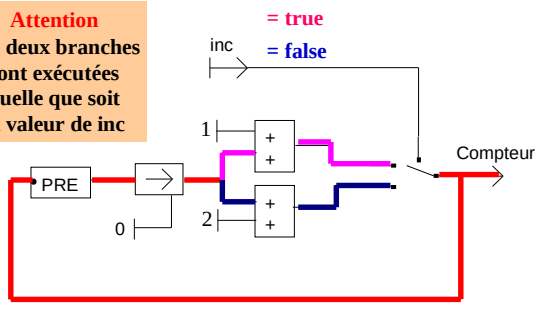
```
Procedure f(x : in real; y: in real; z : out real)
    if (x > 1.0) or (x < -1.0) then
        z := y/x;
    else
        z := y;
    if z < 2.0 then
        z = 2.0;
```

Taux de couverture

- **de branche** (x=2.0, y=6.0), (x=-1.0,y=1.0)
- **décisionnel** + (x=-2, y=3.0)
- **de chemin** + (x=2.0, y=1.0), (x=0.5,y=2.0)

Taux de couverture d'un modèle SCADE

Attention
Les deux branches sont exécutées quelle que soit la valeur de inc



Tests d'intégration

Validés en Tests Unitaires

Module A **Module B**

Fonctionnent-ils ensemble ?

$y = f(x_1, x_2)$ **ou** $y = f(x_2, x_1)$

Module A **Module B**

Validation des interfaces en boîte blanche

➔ **Applicable à SCADE**

Limitation de l'approche boîte blanche

```
Procedure f(x : in real; y: in real; z : out real)
        if (x > 1.0) or (x < -1.0) then
                z := y/x;
        else
                z := y;
```

Boîte blanche ➔ débuggeur

(1) La présence d'un espion modifie le comportement temps réel
(2) Que se passe-t-il si le débuggeur / simulateur a … un bug?

Validation fonctionnelle

- Tests en boîte noire
  - ✓ Contrôle des entrées
  - ✓ Observations des sorties

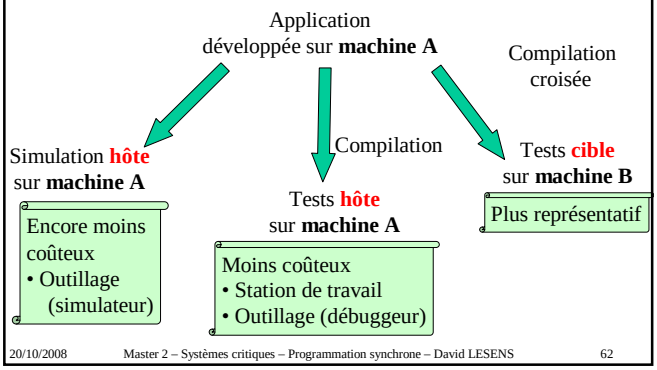
Non intrusifs

➔Jeux de test « simulation » « re-jouable » sur le code final

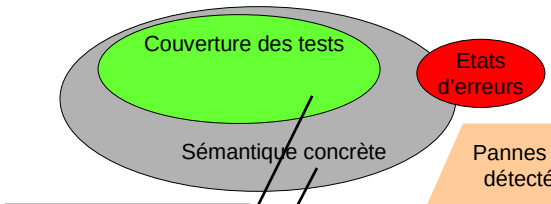
- Définir le niveau de validation
  - ✓Test unitaire / intégration / fonctionnel
  - ✓Revue de code / Preuve…
  - ✓Cible / hôte

➔ La simulation ne remplace pas la validation

Chaînes de compilation hôte et cible

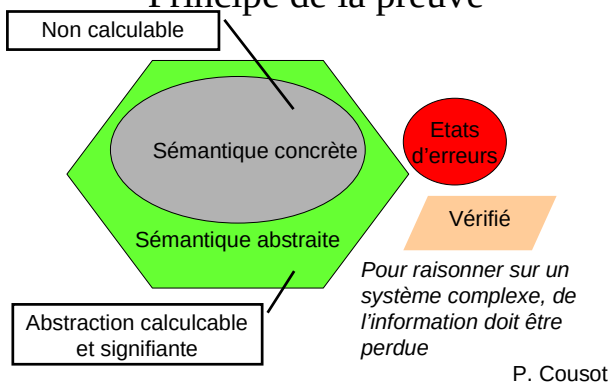


Test de programme



Le test de programme ne peut servir qu'à montrer la présence de bug, mais pas à montrer leur absence!
E. Dijkstra

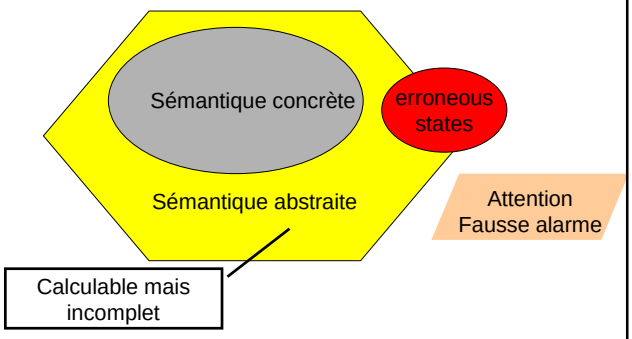
Principe de la preuve



Pour raisonner sur un système complexe, de l'information doit être perdue

P. Cousot

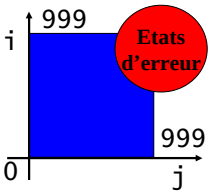
Limitation de la preuve



Exemple (1)

```
int a[1000];
for (i = 0; i < 1000; i++) {
  for (j = 0; j < 1000-i; j++) {
    // 0 <= i <= 999
    // 0 <= j <= 999
    a[i+j] = 0;
  }
}
```

Warning

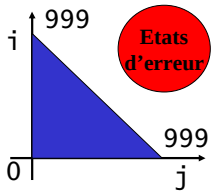


Non conclusif

Exemple (2)

```
int a[1000];
for (i = 0; i < 1000; i++) {
  for (j = 0; j < 1000-i; j++) {
    // 0 <= i and 0 <= j
    // i+j <= 999
    a[i+j] = 0;
  }
}
```

Safe



Conclusif

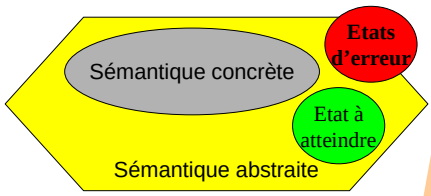
Propriétés de sûreté et vivacité

- **Sûreté** (« safety »)

Quelque chose de « mauvais » n'arrive jamais

- **Vivacité** (« liveness »)

Quelque chose de « bien » arrivera forcément dans le futur



L'outil de preuve de SCADE ne peut pas prouver une propriété de vivacité

Intérêt des propriétés de vivacité

- Propriété de « vivacité » / propriété « temporelle »
  - ✓ Exemple: lorsqu'une erreur est détectée, le logiciel doit émettre une alarme vers l'utilisateur
    - ❖ Vivacité: l'alarme sera obligatoirement émise à un moment donné ou à un autre

Mais quand ?

➔ *Pas acceptable pour un logiciel temps réel critique*

- ❖ Propriété temporelle: l'alarme sera obligatoirement émise 1 seconde après l'occurrence de l'erreur

➔ **Propriété de sûreté**

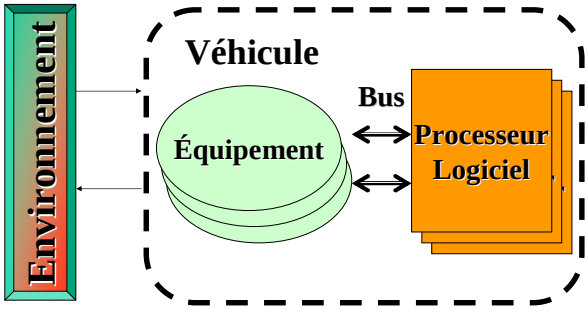
La preuve

- Démonstration « mathématique » exhaustive qu'un programme satisfait une propriété

➔ Très rarement le cas!

Un logiciel ne remplit généralement une propriété que s'il fonctionne dans un environnement correct

Le logiciel n'est qu'une brique d'un ensemble complexe



Principe de la preuve

- Considérer le logiciel à valider
- Définir les propriétés qu'il doit satisfaire
- Définir dans quelles conditions ces propriétés sont satisfaites
  - ➔ Définir formellement son environnement

  (□ environnement correct) ∧ logiciel ⇒ propriété

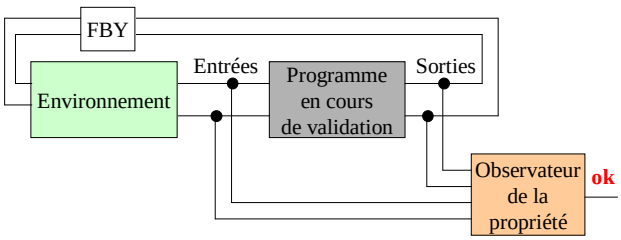
  - ✓ Environnement en boucle fermée ou ouverte

Expression de propriétés

Notion d'observateur

- Un observateur est un programme qui observe le programme en cours de validation et qui renvoie « vrai » tant que la propriété est satisfaite
  - ❖ Observation des entrées du programme
  - ❖ Observation des sorties du programme
- Idem pour propriété sur l'environnement

Observateurs en SCADE



- ➔ Utile pour le test (oracle)
- ➔ Utilisé par l'outil de preuve de SCADE

Environnement indéterministe (1)

L'environnement du logiciel ne peut pas être défini de façon complètement déterministe

- ✓ Intervention humaine
- ✓ Occurrence d'une panne
- ✓ …

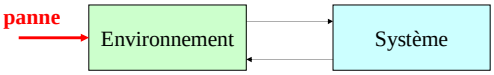
➔ Environnement indéterministe

Mais SCADE est un langage déterministe !

Environnement indéterministe (2)

L'indéterminisme est modélisé par une entrée supplémentaire

Exemple: occurrence d'une panne



Assertion

Une assertion permet de faire une hypothèse sur un environnement « trop » indéterministe

Exemple:

- ✓ Entrée « pg » modélise une panne d'un gyroscope
- ✓ Entrée « pt » modélise une panne d'une tuyère
  - ➔ Développer un système tolérant à une faute (« one fault tolerant »)

Preuve: assert #(pg, pt)

Causalité

Principe de la preuve:
(□ assertion) ∧ environnement correct ∧ logiciel ⇒ propriété

Une assertion peut être vraie à un instant t…
et devenir « inévitablement » fausse dans le future

➔ Une telle assertion n'est pas causale

Une assertion est causale ⇔ **(assertion ⇒ ∃Ψ, assertion)**

Un outil de preuve doit calculer l'assertion causale minimale

Exemple: front montant

- ➢ Un opérateur front montant a déjà été validé « non exhaustivement » par simulation

➔ Quelle propriété exhaustive peut-on demander d'un front montant ?

Un front montant n'est jamais vrai deux fois de suite

➔ Écrire l'observateur de cette propriété
➔ Prouver la!

Environnement plus complexe - 1

- Assertion de l'environnement suivant

  Le système s'initialise correctement par la séquence d'actions « b », « a », « c »

  ➔ Écrire l'expression régulière correspondante

  ➔ Écrire l'assertion correspondante

  – En SCADE « data-flow »

(b a c) .*

□* b □* a □* c .*

Environnement plus complexe - 2

- Assertion de l'environnement suivant

  L'événement « a » se produit toujours entre les événements « b » et « c »

  ➔ Écrire l'expression régulière correspondante

  ➔ Écrire l'assertion correspondante

  – En SCADE « data-flow »

[(b | c) * (b a c)] *

[(b | c | □)* (b □* a □* c)] *

Plan

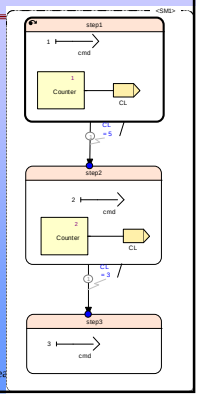
Automata

Automata

- **Safe State Machine (SSM)**
  - Automata from Esterel language
  - Added to Scade in versions 4 and 5
    - Weak semantics integration
    - In two separated tools
- **Event driven and data flow driven**
  - The two views are merged in Scade version 6
    - In the same semantics
    - In the same tool

Data flow and automata

- **A node is composed of**
  - Equations (data-flow)
  - Automata (event driven)
- **An automaton is composed of**
  - States
  - Transitions
- **A state is composed of**
  - Equations
  - Automata



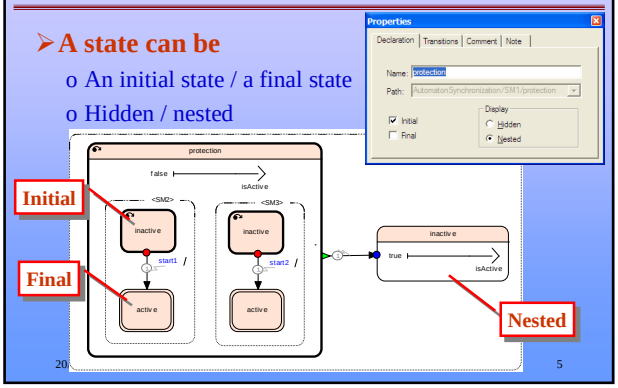
Principles of Automata

- **Semantics equivalence**
  - There exists a data-flow model semantically equivalent to any automaton
- **Automaton scheduling**
  - At most one transition fired *per cycle*
  - Exactly one active state per *cycle*

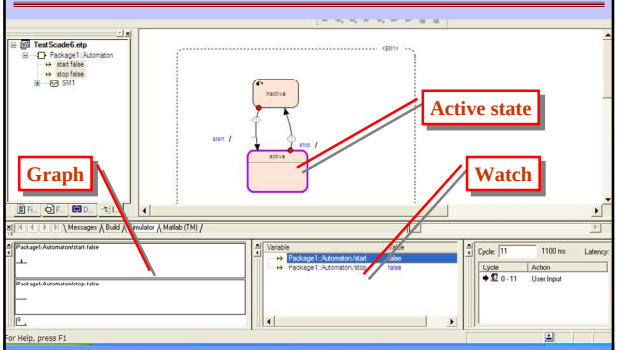
    (except then parallel states are defined)

States

- A state can be
  - An initial state / a final state
  - Hidden / nested

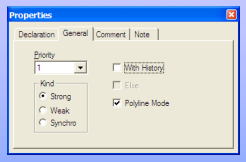


Automaton simulation

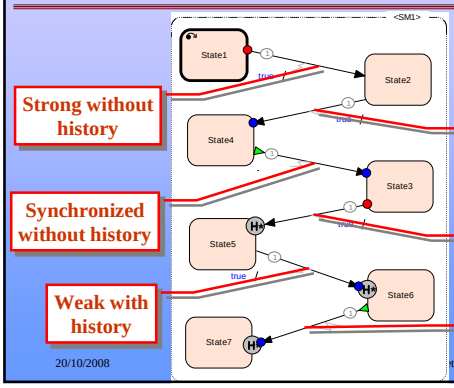


Transitions

- A transition can
  - have a **weak** pre-emption
  - have a **strong** pre-emption
  - be **synchronized**
- It can have
  - A guard
  - An action
- It has a priority
- It can be with or without a history



Graphical transitions

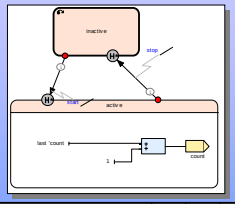


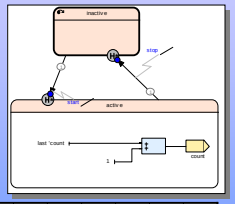
Strong and weak transitions

- **Strong transition**
  - o The transition is triggered before the state execution
  - ➔ The guard can not depend on the current value of a data
- **Weak transition**
  (or "*weak delayed*")
  - o The state is executed before the transition triggering
  - ➔ The guard can depend on the current value of a data

Example (1/2)

Strong transition **Weak transition**

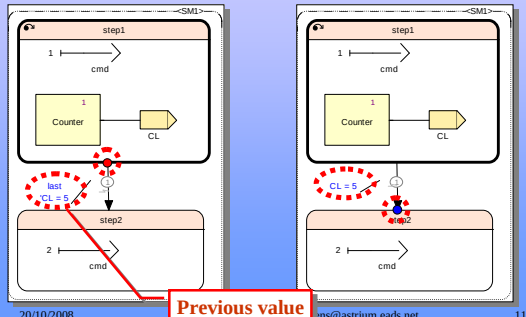




start				T						
stop							T			
count strong	0	0	0	1	2	3	4	4	4	4
count weak	0	0	0	0	1	2	3	4	4	4

Example (2/2)

The behaviours of the two following models are equivalent

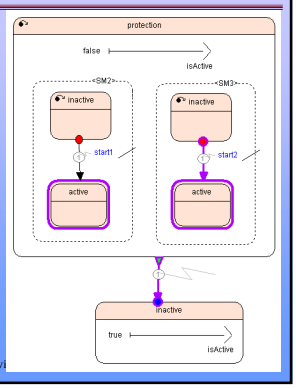


Synchronized transition

- **A synchronized transition**
  - o Has no guard
  - o Is triggered as soon as all nested automata reach a final state

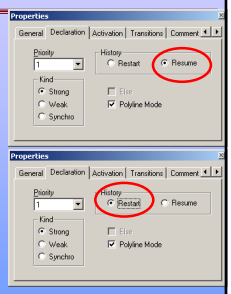
Example

- **Start1 received**
  - Still in protection state
- **Start2 received**
  - Final states reached
- **Transition inactive triggered**



Transition history

- **Transition without history**
  - The state resumes its execution
  - The memories are reset
- **Transition with history**
  - The state resumes its execution
  - The memories are not reset
- **Two types of memories**
  - PRE : local to the state
  - LAST : common to the node

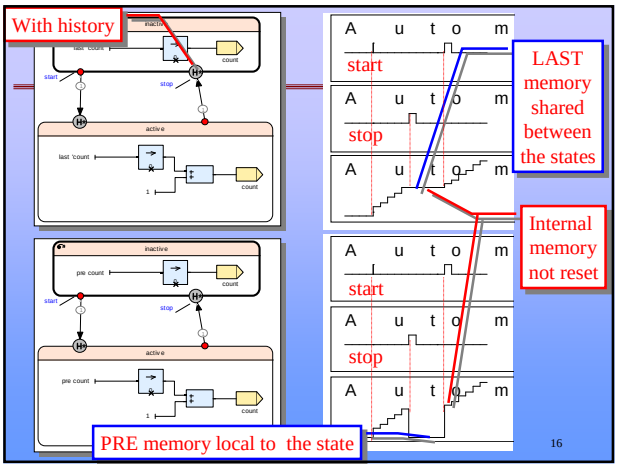


Shared memory

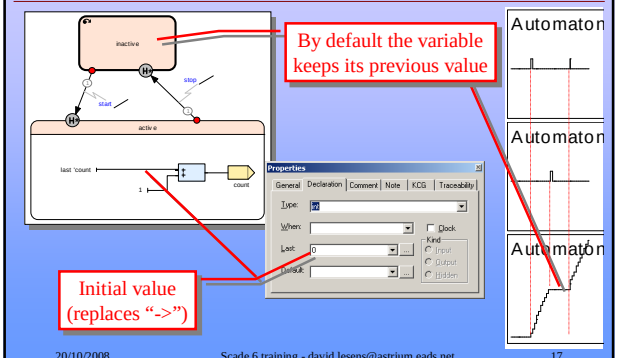
- **Data flow point of view**
  - Access to the last value of a flow in its scope
    - "pre *expression* "
- **Mode automata point of view**
  - Access to values computed in other states
    - "last 'x"

  ("x" is a named flow, not an expression

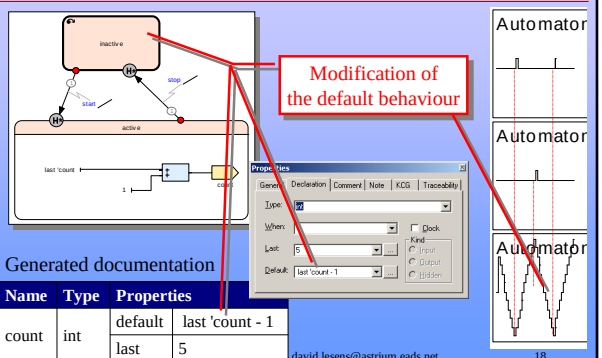
  ➔ utilization of ')



Default actions in state



Modifying the default action

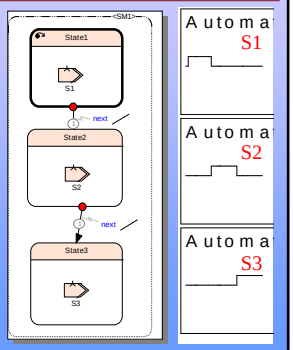


Generated documentation

Name	Type	Properties	
count	int	default	last 'count - 1
		last	5

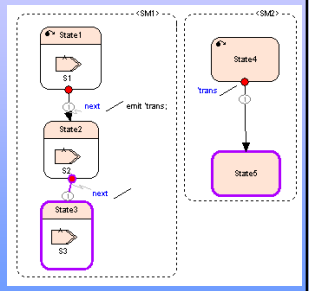
Signals

- **A signal can be**
  - Present ➔ true
  - Absent ➔ false
- **A signal can not be**
  - An input / output
- **≠ Boolean value**
  - A Boolean value keeps its previous value then non updated in a state



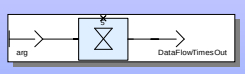
Composition and communication

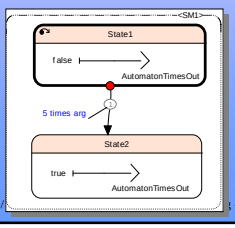
- **A signal can be**
  - Emitted in a state
  - Emitted on a transition
- **A transition can be triggered by a signal**

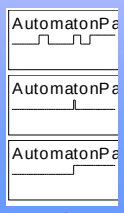


Factor

- **A factor specifies on many time a condition must be true**
  - In a data flow view
  - In a guard (automaton)





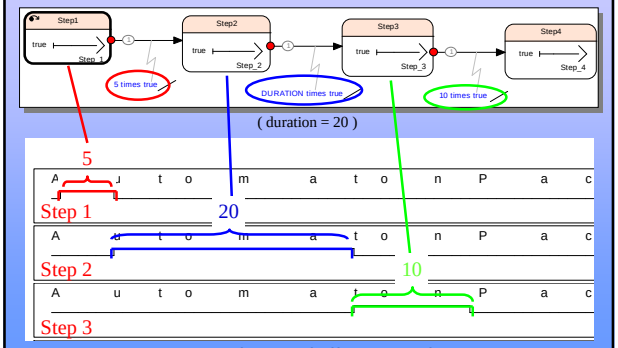


arg

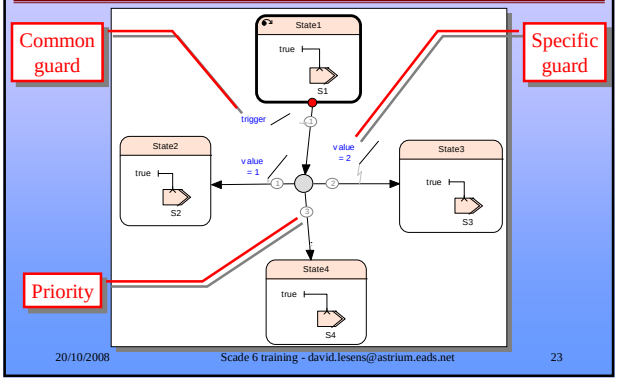
DataFlow

Automaton

Time-out with factor



Fork



Plan

- Logiciel embarqué temps réel critique
- Approche synchrone et introduction à Scade
  - ✓Exercices: flots de données
- Validation et preuve
- Automates synchrones
  - ✓Exercices: automates
- Tableaux et boucles
  - ✓Exercices: tableaux
- Front montant tolérant aux bruits

Exercices: automates

- Expressions régulières
- Chronomètres
- Observateur

Environnement plus complexe - 1

- Assertion de l'environnement suivant

  Le système s'initialise correctement par la séquence d'actions « b », « a », « c »

  ➔ Écrire l'expression régulière correspondante
  ➔ Écrire l'assertion correspondante
  – En SCADE « data-flow »
  – Avec un automate SCADE

(b a c) .*

□* b □* a □* c .*

Environnement plus complexe - 2

- Assertion de l'environnement suivant

  L'événement « a » se produit toujours entre les événements « b » et « c »

  ➔ Écrire l'expression régulière correspondante
  ➔ Écrire l'assertion correspondante
  – En SCADE « data-flow »
  – Avec un automate SCADE

[(b | c) * (b a c)] *

[(b | c | □)* (b □* a □* c)] *

Exercice: Chronomètre

Le chronomètre accepte quatre entrées

- ✓ « inc » et « reset » (voir compteur simple)
- ✓ « start » de type booléen
- ✓ « freeze » pour geler / dégeler l'affichage

➢ Reseter le compteur sur ordre « reset »

➢ A l'initialisation et après un « reset », ne démarrer qu'à la réception du signal « start »

Exercice: Observateur

Ecrire l'observateur de la propriété suivante

➢ Avant 10 tics d'horloge

- ✓le système ne satisfait aucune propriété particulière (initialisation)

➢ Après 10 tics d'horloge

- ✓lorsque le système reçoit l'événement « a », il émet le signal « b »
  - ❖Au minimum 5 tics après la réception de « a »
  - ❖Au maximum 10 tics après la réception de « a »

➢ « a » est reçu au plus tous les 20 cycles

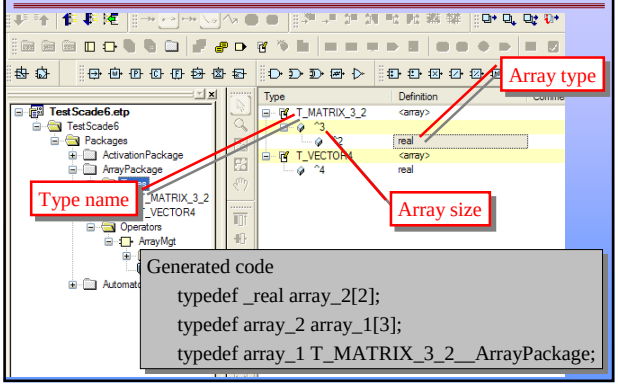
Plan

Arrays

Arrays definition

- **Restrictions**
  - Static size
  - First element = index 0
- **Definitions**
  - type VECTOR = real ^ 4 ;
  - type MATRIX_2_3 = real ^ 3 ^ 2 ;
    - 2 lines, 3 columns
    - typedef real LINE_3[3];
    - typedef LINE_3 MATRIX_2_3 [2];

Editing array types



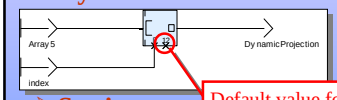
Generated code

```
typedef _real array_2[2];
typedef array_2 array_1[3];
typedef array_1 T_MATRIX_3_2__ArrayPackage;
```

Array access

Array5=[2,4,6,8,10], Index=3

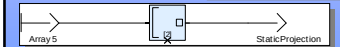
- **Dynamic**



Index = 3 ➔ Output = 8
Index = 10 ➔ Output = 12

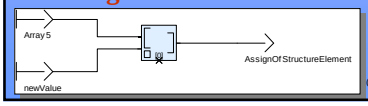
Default value for index out of range

- **Static**



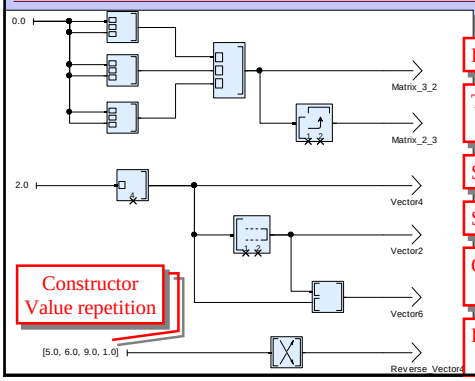
Output = 6

- **Assignment**



newValue = 3 ➔
Output = [3,4,6,8,10]

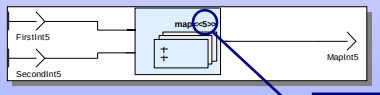
Some operators on arrays



Iterations

Equivalent to “for” in C
Map / Mapi / Mapw / Mapiw
Fold / Foldi / Foldi / Foldiw

Map

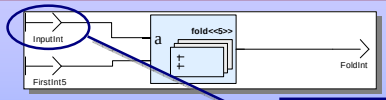


Size of the input vector

```
for (i = 0; i < 5; i++) {
        MapInt5[i] = FirstInt5[i] + SecondInt5[i];
}
```

MAP: Apply the operator successively on each element of the input vector(s) element[i] .element’[i]

Fold

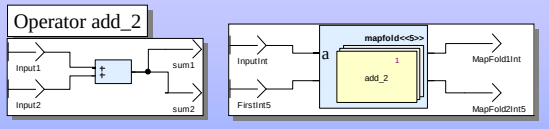


First element of the iteration

```
FoldInt = InputInt;
for (i = 0; i < 5; i++) {
        FoldInt = FoldInt + FirstInt5[i];
}
```

FOLD: Apply recursively the operator on input vector element[i] .element[i+1]

Mapfold



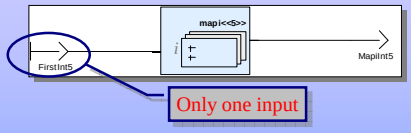

```
MapFold1Int = InputInt;
for (i = 0; i < 5; i++) {
        add_2_ArrayPackage(MapFold1Int, FirstInt5[i],
        &MapFold1Int[i], &MapFold2Int[i]);
}
```

Recursion

Succession

Nodes used with a mapfold iterator should duplicate their output
We obtain both results at the same time

Mapi = Map with iterator as input

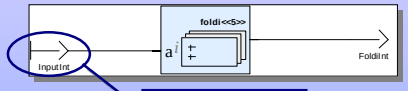


Only one input

```
for (i = 0; i < 5; i++) {
        MapiInt5[i] = i + FirstInt5[i];
}
```

The index of the iteration
is the first argument of the node

Foldi = Fold with iterator as input



No vector in input

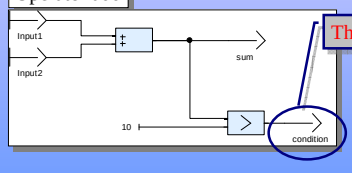
```
FoldiInt = InputInt;
for (i = 0; i < 5; i++) {
        FoldiInt = i + FoldiInt;
}
```

The input flow is the iterator

Mapw / Foldw = Partial operators

- **Capability to stop an iteration on a Boolean condition computed by the operator**

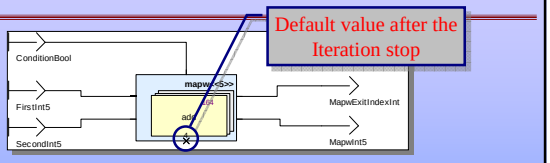
Operator add



The iteration can be stopped

As soon as the condition is false, the iteration is topped

Mapw = Map partial operator



Default value after the Iteration stop

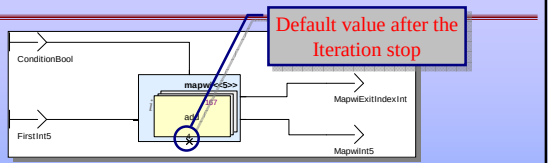
```
MapwExitIndexInt = 0;
for (i = 0; i < 5; i++) {
 if (ConditionBool) {
  add(FirstInt5[i], SecondInt5[i], &ConditionBool, &MapwInt5[i]);
  MapwExitIndexInt = i + 1;
 } else { MapwInt5[i] = 4; } }
```

The iteration can be stopped

Default value after the Iteration stop

It is recommended to not use this operator (WCET)

Mapwi = Mapi + Mapw



Default value after the Iteration stop

```
MapwiExitIndexInt = 0;
for (i = 0; i < 5; i++) {
 if (ConditionBool) {
  add(i, FirstInt5[i], & ConditionBool, &MapwiInt5[i]);
  MapwiExitIndexInt = i + 1;
 } else { outC->MapwiInt5[i] = 4; } }
```

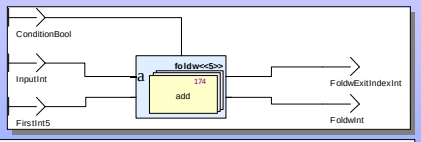
The iteration can be stopped

The iterator is the first argument

Default value after the Iteration stop

It is recommended to not use this operator (WCET)

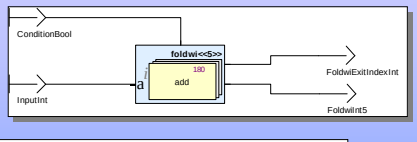
Foldw = Fold partial operator




```
FoldwInt = InputInt;
for (i = 0; i < 5; i++) {
 if (ConditionBool) { break; }
 add(FoldwInt, FirstInt5[i], & ConditionBool, &tmp);
 FoldwInt = tmp;
}
```

The iteration can be stopped

Foldwi = Foldi + Foldw




```
FoldwiInt5 = InputInt; tmp = ConditionBool;
for (i = 0; i < 5; i++) {
 if (ConditionBool) { break; }
 add(i, FoldwiInt5, & ConditionBool, &tmp);
 FoldwiInt5 = tmp;
}
FoldwiExitIndexInt = i;
```

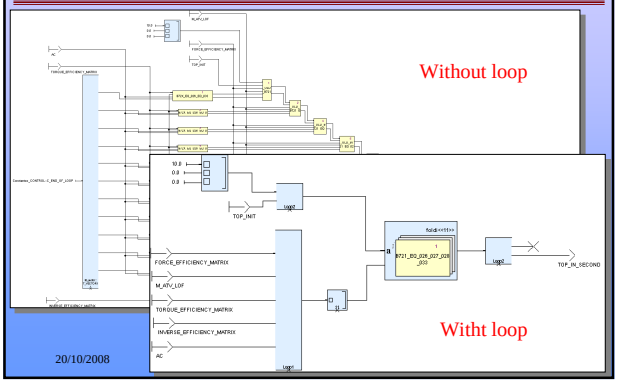
The iteration can be stopped

The input flow is the iterator

Iteration summary

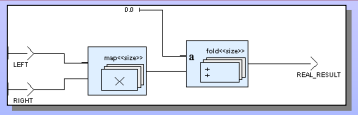
- Map = Successive application
- Fold = Recursive application
- Mapfold = Map + Fold
- Mapi = Map with iterator as input
- Foldi = Fold with iterator as input
- Mapw = Map partial operator
- Mapwi = Mapi + Mapw
- Foldw = Fold partial operator
- Foldwi = Foldi + Foldw

Example 1

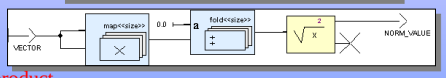


Example 2: cross product

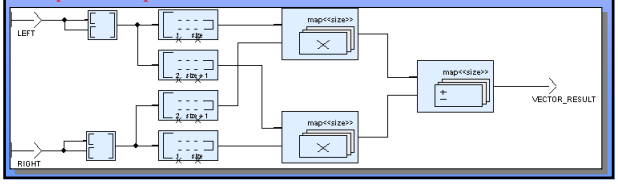
Compute scalar product



Compute vector norm

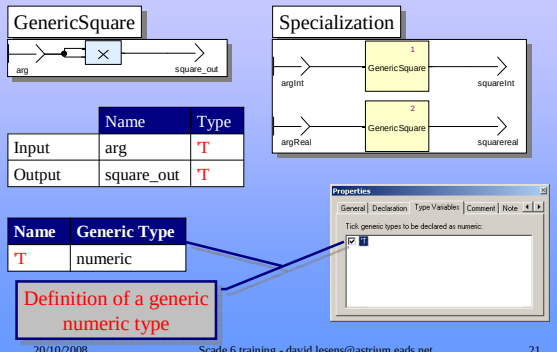


Compute cross product



Genericity

Generic operator definition



	Name	Type
Input	arg	'T
Output	square_out	'T

Name	Generic Type
'T	numeric

Definition of a generic numeric type

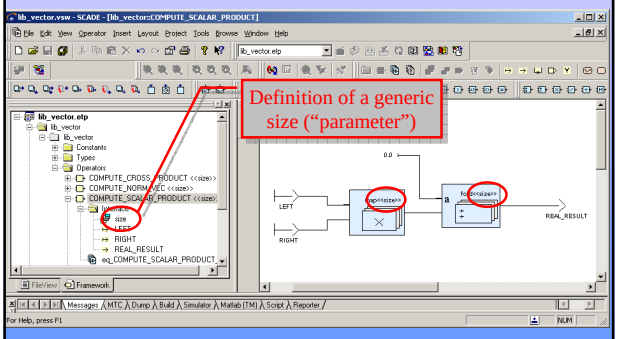
Generic operator instantiation

```
int GenericSquare_int ( int arg ) {
        int square_out;
        square_out = arg * arg;
        return square_out;
}
```

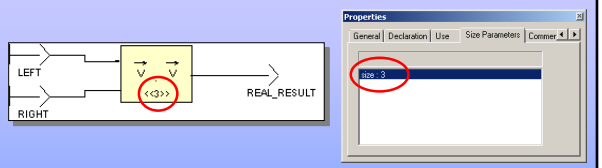
```
real GenericSquare_real ( real arg ) {
        real square_out;
        square_out = arg * arg;
        return square_out;
}
```

```
void Specialization(    int argInt; real argReal;
                        int squareInt; real squarereal; ) {
*squareReal = GenericSquare_real ( argReal );
*squareInt =  GenericSquare_int ( argInt );
}
```


Definition of parameters



Parameter instantiation




```
REAL_RESULT = 0.0;
for (i = 0; i < 3; i++) {
   REAL_RESULT = REAL_RESULT + (*LEFT)[i] * (*RIGHT)[i];
}
return REAL_RESULT;
```

Plan

- Logiciel embarqué temps réel critique
- Approche synchrone et introduction à Scade
  - ✓Exercices: flots de données
- Validation et preuve
- Automates synchrones
  - ✓Exercices: automates
- Tableaux et boucles
  - ✓Exercices: tableaux
- Front montant tolérant aux bruits

Exercice: Produit scalaire

- Soit deux vecteurs de même taille

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad \vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

- Leur produit scalaire est défini de la manière suivante

$$\vec{x} \cdot \vec{y} = x_1 \times y_1 + x_2 \times y_2 + \cdots + x_n \times y_n$$

➔ Ecrire l'opérateur paramétrique « scalar_product »

Exercice: Produit vectoriel

- Soit deux vecteurs de taille 3

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad \vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

- Leur produit vectoriel est défini de la manière suivante

$$\vec{x} \wedge \vec{y} = \begin{pmatrix} x_2 \times y_3 - x_3 \times y_2 \\ x_3 \times y_1 - x_1 \times y_3 \\ x_1 \times y_2 - x_2 \times y_1 \end{pmatrix}$$

➔ Ecrire l'opérateur « cross_product » avec des opérations sur les tableaux

(i.e. sans accéder à x_1, x_2, x_3, y_1, y_2, y_3)

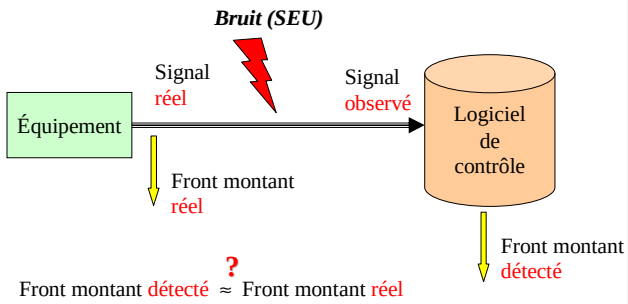
Plan

- Logiciel embarqué temps réel critique
- Approche synchrone et introduction à Scade
  - ✓Exercices: flots de données
- Validation et preuve
- Automates synchrones
  - ✓Exercices: automates
- Tableaux et boucles
  - ✓Exercices: tableaux
- Front montant tolérant aux bruits

Front montant tolérant au bruit

- Écrire un programme détectant un front montant en présence de SEU (« Single Event Upset »)
  - ✓ Un SEU peut modifier de façon aléatoire un signal
    - ❖ « Vraie » valeur = true ➔ Valeur « observée » = false
    - ❖ « Vraie » valeur = false ➔ Valeur « observée » = true
  - ➔ Spécifier textuellement le problème
  - ➔ Modéliser le système et son environnement
  - ➔ Prouver la correction du modèle
  - Nota: les hypothèses nécessaires seront clairement écrites et formalisées par des assertions

Objectif



Principe: Confirmation du signal

- Front montant sur le signal observé
  - ⇒ Front montant du signal réel

ou

  - ⇒ Occurrence d'un SEU (bruit)
    - ***Signal observé ≠ Signal réel***

Principe de la solution:

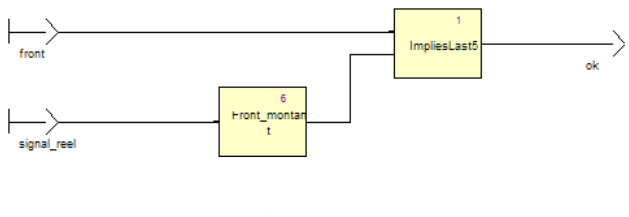
Confirmation de l'observation

- ➔ Résultat retardé par rapport au signal réel

Propriétés

- Propriété de sûreté
  - ✓ ¬ Front(réel) ⇒ ¬ Front(observé)
    - ↳ avec un délai
  - ✓ Front(observé) ⇒ ∃ Front(réel) *dans les 5 derniers cycles*
- Propriété de vivacité
  - ✓ Front(réel) ⇒ Front(observé)
    - ↳ avec un délai
  - ✓ Front(réel) *dans les 5 derniers cycles*
    - ⇒ Front(observé) *depuis les 5 derniers cycles*
- Propriété supplémentaire
  - ✓ Front montant non détecté deux fois de suite

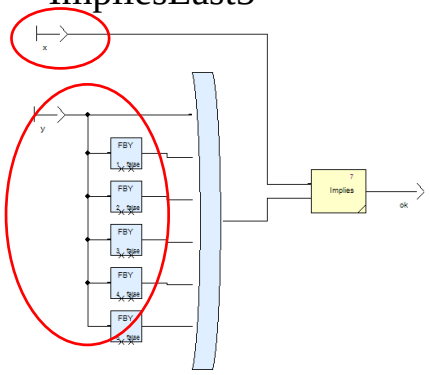
Propriété de sûreté



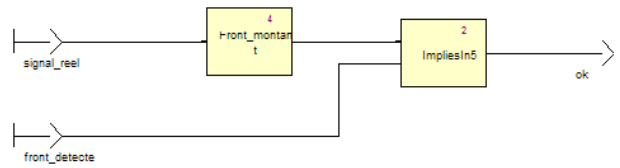
ImpliesLast5

Si le front a été détecté

Le signal réel a effectivement eu un front lors des 5 derniers cycles



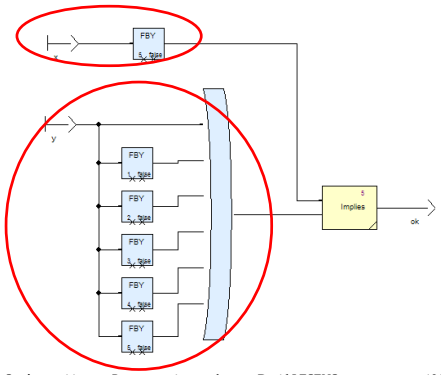
Propriété de vivacité



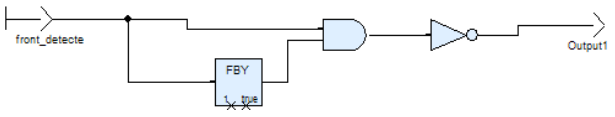
ImpliesIn5

Si le signal réel a eu un front il y a 5 cycles

Le front a été détecté depuis



Propriété supplémentaire



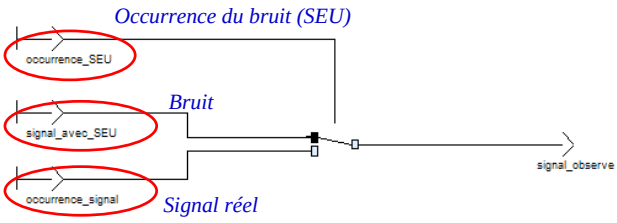
Front jamais détecté deux fois de suite

Environnement

- Indéterministe
  - ✓ Occurrence d'un SEU ou non

➔ Ajout de deux entrées
- ✓ « occurrence_SEU »: Un SEU est arrivé
- ✓ « signal perturbé »: Signal observé en présence de bruit

Environnement



Loi d'arrivée des SEU

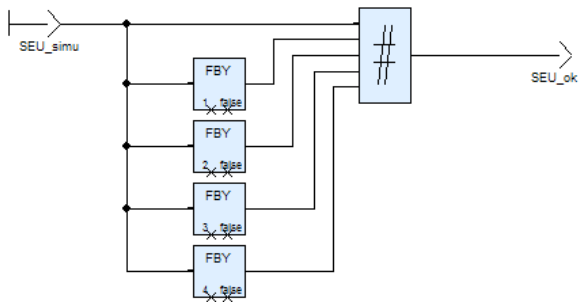
- Bruit quelconque (SEU à chaque cycle)
  - ➔ *Problème impossible à résoudre*

- Hypothèse
  - ✓ SEU pas trop fréquent

  (en réalité, pas plus d'un sur plusieurs minutes)
  - ➔ Assertion suffisante

    « Au maximum un SEU tous les 5 cycles »

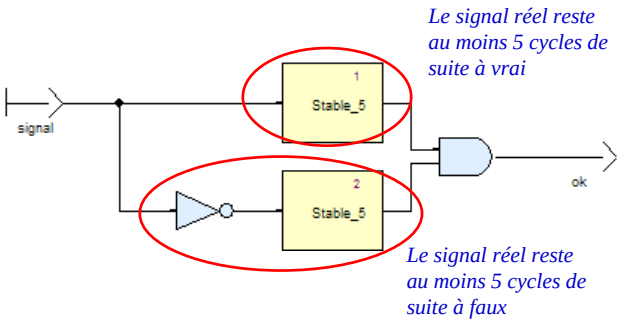
Loi d'arrivée des SEU



Hypothèse sur le signal observé

- Signal réel quelconque (changement possible à chaque cycle)
  - ➔ *Problème impossible à résoudre*
    (pas de confirmation possible)
- Hypothèse
  - ✓ Signal réel « assez » stable (pas de changement trop fréquent)
  - ➔ Assertion suffisante
    « Le signal réel reste au minimum 5 cycles à une valeur »

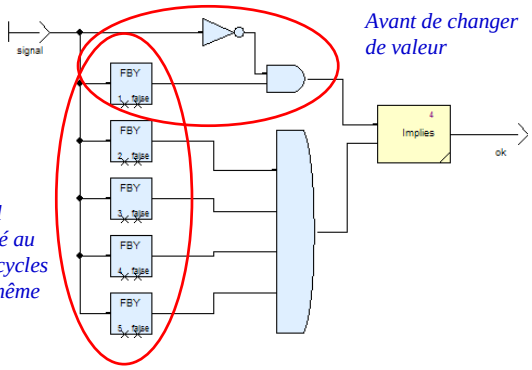
Hypothèse sur le signal observé



Le signal réel reste au moins 5 cycles de suite à vrai

Le signal réel reste au moins 5 cycles de suite à faux

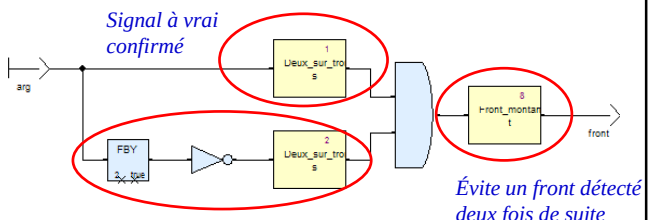
Stable 5



Avant de changer de valeur

Le signal était resté au moins 5 cycles dans le même état

Front montant tolérant au bruit

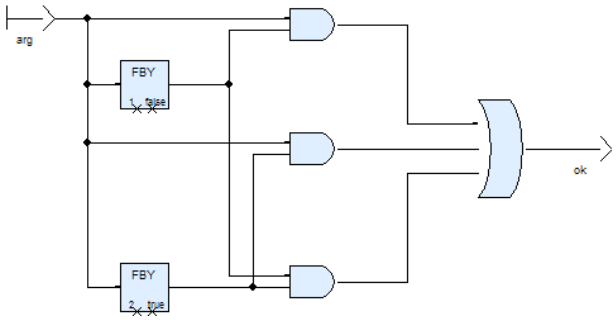


Signal à vrai confirmé

Évite un front détecté deux fois de suite

Auparavant, signal à faux confirmé

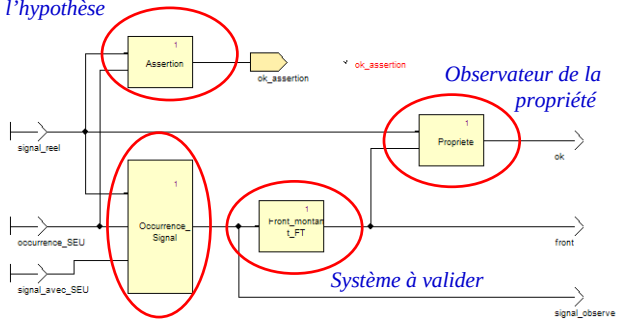
Deux sur trois



Preuve

Observateur de l'hypothèse

Observateur de la propriété



Système à valider

Description de l'environnement