

# FALLUTO 2.0

MODEL CHECKER PARA LA VERIFICACIÓN DE  
SISTEMAS TOLERANTES A FALLAS

# Organización de la charla

1. Introducción y motivación
2. Model Checking y Estructuras de Kripke
3. NuSMV
4. Falluto 2.0
  - a. Comportamiento
  - b. Compilación
  - c. Propiedades
5. Ejemplo

# Introducción

Surge como revisión y extensión de Falluto, trabajo final de grado de Edgardo Hames, dirigido por Pedro D'Argenio.

Al igual que su antecesor, es un front-end para el model checker NuSMV.

Busca presentar un lenguaje intuitivo y ágil para la descripción de sistemas tolerantes a fallas y la especificación de propiedades sobre el mismo.

# Introducción

Para ello presenta un lenguaje declarativo para la inyección de fallas en el sistema, ocultando los detalles funcionales de las mismas.

# Motivación

```
int y1 = 0;
int y2 = 0;
short in_critical = 0;
```

```
active proctype process_1() {
    do
        :: true ->
            y1 = y2+1;
            ((y2==0) || (y1<=y2));
            in_critical++;
            in_critical--;
            y1 = 0;
    od
}
```

```
active proctype process_2() {
    do
        :: true ->
            y2 = y1+1;
            ((y1==0) || (y2<y1));
            in_critical++;
            in_critical--;
            y2 = 0;
    od
}
```

# Introducción

Falluto busca ofrecer además una batería de propiedades predefinidas para experimentar sobre el comportamiento del sistema bajo el efecto de las fallas con simplemente nombrarlas.

# Model Checking

Es un método formal para la verificación de sistemas.

Mediante el mismo se explora exhaustivamente todos los escenarios de un sistema.

NuSMV es un Model Checker basado en BDD.

# Estructuras de Kripke

## Estructuras De Kripke:

Una estructura de Kripke sobre AP se define como la 4-upla  $M = (S, I, R, L)$  donde

$S$  es un conjunto finito de estados

$I \subseteq S$  estados iniciales

$R \subseteq S \times S$  relación de transición left-total.

$L: S \rightarrow P(AP)$  función de etiquetado o interpretación.



# Estructuras de Kripke

En model checking usualmente sucede que:

- $S$  es un conjunto de valuaciones sobre las variables del sistema.
- $AP$  es un conjunto de expresiones booleanas sobre las variables.
- $L(v) = \{ a \text{ en } AP \mid v(a) \text{ es verdadero} \}$  con  $v$  en  $S$
- $R$  explica la relación entre la valuación actual y la próxima

# Lenguaje de NuSMV

MODULE main()

VAR

var1:boolean;

var2:{1,2,a,b};

INIT

var1 & var2 = a

TRANS

( var1 -> next(var2) in {a,b}  
& !var1 -> next(var2) in  
{1,2}  
) | next(var1) = !var1

**Propiedades**

LTLSPEC G var1 = FALSE

CTLSPEC AG TRUE

**Fairness**

FAIRNESS var2 in {1,a}

COMPASSION (var2 = 1 , !var1)

# El Lenguaje de Falluto2.0: definición del comportamiento del sistema

```
PROCTYPE
  proc(cxtv1, ctxv2 ; sync1, sync2)
  VAR
    var1 : bool
  INIT
    {formula}
  TRANS
    [name]: enable => changes
    [sync1] ...
    []: ...
```

```
ENDPROCTYPE
```

```
...
```

```
INSTANCE inst1 = proc(inst2.v,  
TRUE,s1,s1)
```

```
INSTANCE inst2 = proc2(s1)
```

```
FAIRNESS ...
```

```
COMPASSION( ... )
```

# Descripción de transiciones en Falluto2.0

[nombre]: cond\_habilitación  $\Rightarrow$  post\_condición

La condición de habilitación es una fórmula booleana sobre el estado actual del sistema.

Las post condiciones son listas de next-valores.

# Ejemplo de descripción de un sistema:

```
PROCTYPE machine(; transfer)
```

```
VAR
```

```
    on : bool
```

```
    file : 0..9
```

```
INIT
```

```
    !on & file = 0
```

```
TRANS
```

```
    [transfer]: on => file' = (file + 1) % 10
```

```
    [OnOff]:      => on' = !on
```

```
ENDPROCTYPE
```

# Ejemplo de descripción de un sistema:

-- Instanciación

INSTANCE pc1 = machine(sync)

INSTANCE pc2 = machine(sync)

-- Restricciones

FAIRNESS just(sync)

# Inyección de fallas en Falluto

En la sección introducida por la palabra reservada FAULT de cada proctype:

nombre\_de\_falla : cond\_habilitación => post\_condición **is** TYPE

donde TYPE puede ser:

STOP[(trans1, trans2, ...)]

BYZ([var1, var2, ...])

TRANSIENT

# Inyección de fallas en Falluto

```
PROCTYPE machine(; transfer)
```

```
VAR
```

```
    on : bool
```

```
    file : 0..9
```

```
FAULT
```

```
    f1: is STOP(OnOff)
```

```
    f2: file = 9 => is BYZ(file)
```

```
    f3: => file' = 0 is TRANSIENT
```

```
INIT
```

```
    !on & file = 0
```

```
TRANS
```

```
    [transfer]: on => file' = (file + 1) % 10
```

```
    [OnOff]: => on' = !on
```

```
ENDPROCTYPE
```



# Fairness en Falluto2.0

## **FAULT\_FAIR\_DISABLE:**

deshabilita la condición de fairness de fallas con respecto al sistema.

- G F transicion\_buena


## **INST\_WEAK\_FAIR\_DISABLE:**

deshabilita la condición de fairness del sistema con respecto a las instancias del mismo.

- G habilitada -> G F atendida

# Compilación: transiciones

## Transiciones comunes:

[trans]: pre  $\Rightarrow$   pos1, pos2, ...

act#var' = trans & !stop & pre & pos1 & pos2 &  
... & unchanged

unchanged = &(var<sub>i</sub>' = var<sub>i</sub>) con var<sub>i</sub> en Vars -  
POS - act#var

# Compilación: transiciones

## Transiciones sincronizadas:

$\text{act\#var}' = \text{syncname} \ \& \ \text{!STOP}$   
 $\quad \& \ \text{pre1} \ \& \ \text{pre2} \ \& \ \dots$   
 $\quad \& \ \text{POS1} \ \& \ \text{POS2} \ \& \ \dots$   
 $\quad \& \ \text{unchanged}$

$\text{unchanged} = \&(\text{var}_i' = \text{var}_i) \text{ con } \text{var}_i \text{ en Vars} - \text{U}$   
 $(\text{POS}_j) - \text{act\#var}$

# Compilación: transiciones

## Transiciones de falla:

fault: pre  $\Rightarrow$  pos1, pos2, ... is Type

si no es  
transient

act#var' = fault & !fault\_active & pre &  
fault\_active' & pos1 & pos2 & ... & unchanged

si no es  
transient

# Compilación: transiciones

## Transición de deadlock:

$act\#var' = dk\#trans \ \& \ neg\_pre \ \& \ nada\_cambia$

$neg\_pre = \&\{i \text{ in instance}\} \ \&\{t \text{ in transN}\}$   
 $(! pre_{i,t} \mid Stop_{i,t})$   
 $\&\{s \text{ in sincro}\} (! pre_s \mid Stop_s)$

$nada\_cambia = \&(var_i' = var_i) \text{ con } var_i \text{ in Var -}$   
 $act\#var$

# Compilación: TRANS

La fórmula booleana que define R en el sistema compilado:

$$\bigvee_{t \in \text{Trans}} t$$

donde :

Trans = transiciones\_normales U  
transiciones\_de\_falla U  
transicion\_de\_deadlock U ...

# Compilación: fairness

Tanto:

FAIRNESS  $q$

como:

COMPASSION(  $p$ ,  $q$ )

se traducen de manera directa a NuSMV

# Compilación: fairness

**Instance weak fairness:**

$\text{instanceDK} = \&(! \text{pre}_{\text{trans}_i})$

$\text{FAIRNESS } (V_{\text{transN}} \text{ act\#var} = \text{trans}_i) \mid \text{instanceDK}$



# Compilación: fairness

## Fault-System Fairness:

FAIRNESS ( $V_{\text{transN}} \text{ act\#var} = \text{trans}_i$ ) |  $\text{act\#var} = \text{dk\#trans}$

Para que las fallas no se pongan de acuerdo para no dejar actuar a las transiciones buenas.

# Especificación de propiedades en Falluto

## Propiedades como en NuSMV:

LTLSPEC  $q$

CTLSPEC  $q$

## Propiedades sobre escenarios comunes

NORMAL\_BEHAVIOUR  $\rightarrow q$

*Si solo hago buenas transiciones entonces  $q$  es verdadera*

FINITELY\_MANY\_FAULTS  $\rightarrow q$

*Si eventualmente las fallas dejan de ocurrir entonces  $q$  es verdadera.*

# Especificación de propiedades en Falluto

CHECK\_DEADLOCK

compila a:

CTLSPEC AG act#var != dk#trans

# Trazas de contraejemplo

-> State: 1.1 <-

action# = dk#action

lvar#inst#var1 = FALSE

lvar#inst#var2 = 0

ipc#inst = 0

-> State: 1.2 <-

action# = trans#inst#trans1

lvar#inst#var2 = 1

-> State: 1.3 <-

lvar#inst#var2 = 2

-> State: 1.4 <-

action# = trans#inst#trans2

lvar#inst#var1 = TRUE

ipc#inst = 1

---> State: 0 <---

inst var1 = FALSE

inst var2 = 0

@ [action] inst / trans1

---> State: 1 <---

inst var2 = 1

@ [action] inst / trans1

---> State: 2 <---

inst var2 = 2

@ [action] inst / trans2

---> State: 3 <---

inst var1 = TRUE

**ESO ES TODO**