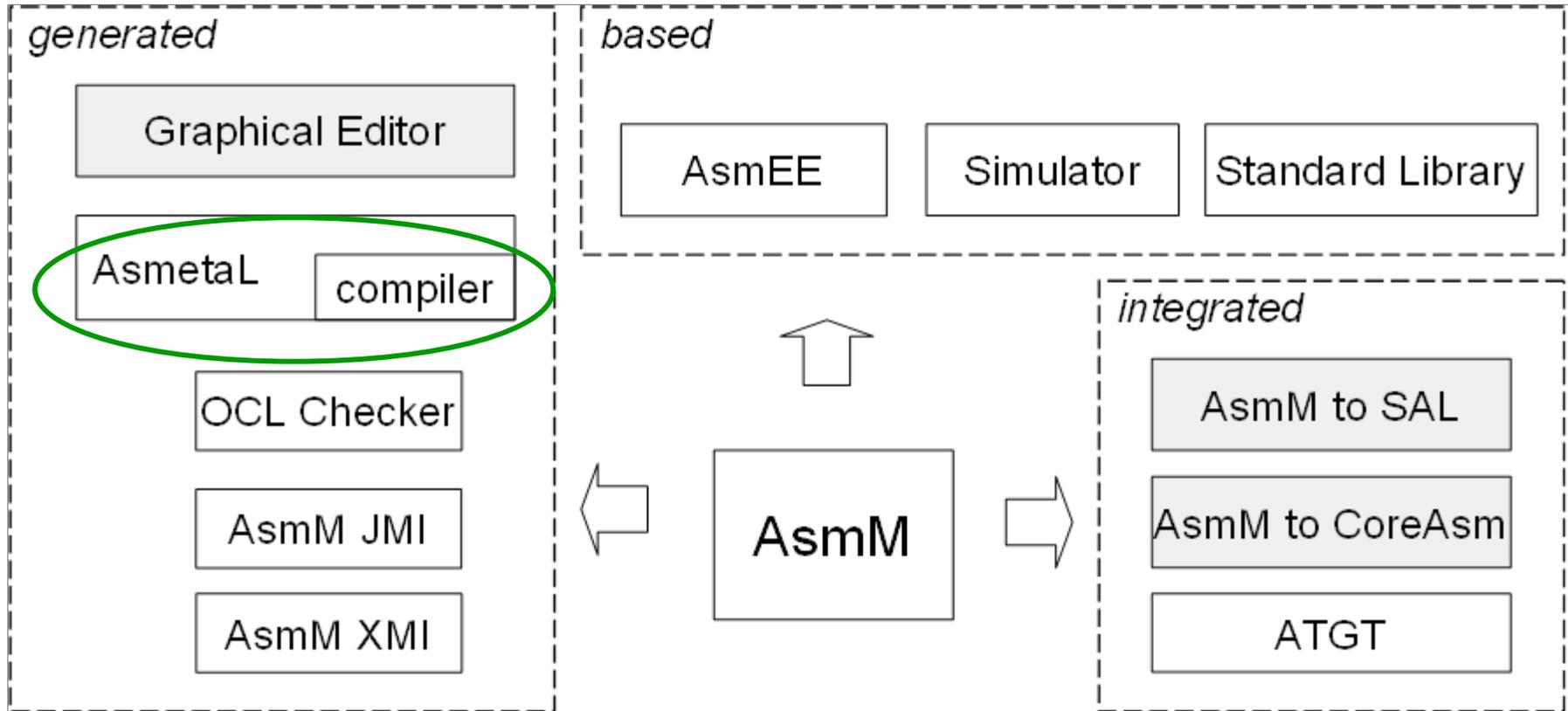


The AsmetaL Language
per la specifica/progettazione di SW

ASMETA toolset

Un insieme di tool per le **ASMs**, sviluppato con tecniche di **meta-modellazione** dell'approccio **Model-driven Engineering (MDE)**



Architettura di ASMETA

Istanza del framework di meta-modellazione OMG per le ASMs

Asmeta è disponibile:

<http://asmeta.sourceforge.net>

- compilatore
- Simulatore
- Validatore
- Documentazione
- StandardLibrary

Il Linguaggio AsmetaL

Linguaggio strutturale

- costrutti per definire la struttura (scheletro) di una ASM mono-agente o sinc./asinc. multi-agente

Linguaggio delle definizioni

- costrutti per introdurre domini (*tipi del linguaggio*), funzioni (con domini e codomini), regole di transizione, e assiomi

Linguaggio dei termini

- **termini di base** come nella logica del primo ordine (costanti, variabili, termini funzionali $f(t_1, t_2, \dots, t_n)$)
- **termini speciali** come tuple, collezioni (insiemi, sequenze, bag, mappe), ecc.

Linguaggio delle regole

- **regole di base** come skip, update, parallel block, ecc.
- **turbo regole** come seq, iterate, turbo submachine

Una ASM è una tupla:

(name, header, body, main rule, initialization)

name è:

[*asynchr*] asm ASM_name

- **ASM_name** è il nome della ASM e deve coincidere con il nome del file .asm che la contiene
- la parola chiave **asynchr** è opzionale (perchè racchiusa in []). Se presente, denota una ASM *asincrona* multi-agent. Se omessa, l'ASM è considerata *sincrona* multi-agent.
- Un **modulo ASM** è come una ASM senza main rule e senza inizializzazione. Nel definire un modulo ASM si usa la parola chiave **module** anzicchè **asm**.

Il Linguaggio Strutturale

header è:

```
[ import m1 [ ( id11,...,id1h1 ) ]  
  ...  
  import mk [ ( idk1,...,idkhk ) ]  
]  
[ export id1,...,ide ] or [ export * ]  
signature :  
  [ dom_declarations ]  
  [ fun_declarations ]
```

import/export di simboli (id) di domini, funzioni (e loro domini e codomini), e regole da/verso altre ASM

export * per esportare tutto

Ricordare: la segnatura contiene dichiarazioni (non definizioni) di domini e funzioni!

Il Linguaggio Strutturale

body è:

definitions :

domain D1 = Dterm1

...

function F1[(p11 in d11,...,p1k1 in d1k1)] = Fterm1

...

[rule_declarations]

[axiom_declarations]

Solo **domini concreti statici** possono essere definiti

Solo **funzioni statiche** possono essere definite

per una regola o assioma, dichiarazione e definizione sono la stessa cosa

Il Linguaggio Strutturale

main rule è:

main rule $R = \text{rule}$

R è il nome della main rule

la main rule è sempre una (macro-)regola **chiusa**,
cioè senza parametri

rule è proprio il corpo della regola di transizione

Se l'ASM è multi-agent, la main rule deve far partire
in parallelo i programmi degli agenti

I programmi degli agenti sono specificati nello stato
iniziale (vedi **initialization**)

Il Linguaggio Strutturale

Initialization è una sequenza di stati iniziali:

[**default**] **init** Id :

domain Dd1 = Dterm11

...

function Fd1[(p11 in d11,...,p1s1 in d1s1)] = Ftermd1

...

Uno stato iniziale deve essere denotato come *default*.

Solo **domini concreti dinamici** possono essere inizializzate

Solo **funzioni dinamiche**, non monitorate, possono essere inizializzate

Un primo esempio di programma ASMETAL

Un FLIP-FLOP device

Header section

*Clause di Import/export ,
vocabolario ASM*

```
asm flip_flop
import StandardLibrary
signature:
  domain State subsetof Natural
  dynamic controlled ctl_state : State
  dynamic monitored high : Boolean
  dynamic monitored low : Boolean
```

Body section

*Static domains/functions
defs., rule decls., e
assiomi*

```
definitions:
domain State = {0n,1n}

rule r_Fsm($i in State, $cond in Boolean,$rule in Rule, $j in State) =
  if ctl_state=$i and $cond
  then par $rule
    $ctl_state := $j
  endpar
  endif

axiom inv_neverBoth over high(), low(): not(high and low)
```

Main rule

```
main rule r_flip_flop = par
  r_Fsm[0n,high,<<skip>>,1n]
  r_Fsm[1n,low,<<skip>>,0n]
endpar
```

Initialization section

*Inizializzazioni per
domini e funzioni dinamici*

```
default init initial_state:
function ctl_state = 0n
```

Il Linguaggio delle definizioni

Dichiarazioni di funzioni (`fun_declarations`)

static f: [D ->] C

[dynamic] monitored f: [D ->] C

[dynamic] controlled f: [D ->] C

[dynamic] shared f: [D ->] C

[dynamic] out f: [D ->] C

[dynamic] local f: [D ->] C

- D e C sono risp. Il dominio ed il codominio di f
- D è opzionale; non va messo se f è 0-aria (cioè una variabile)

Il Linguaggio delle definizioni

Dichiarazioni di funzioni (`fun_declarations`)

Esempi (Flip_Flop): **variabili**

dynamic controlled `ctl_state : State`

dynamic monitored `high : Boolean`

dynamic monitored `low : Boolean`

Altri esempi funzioni:

// una funzione che associa un intero ad ogni intero

controlled `votoByID: Integer -> Integer`

// una funzione che dice quali interi sono scelti

monitored `interoscelto: Integer -> Boolean`

Il Linguaggio delle definizioni

Caratterizzazione dei domini

type-domain: caratterizzano il superuniverso

basic type-domains: **Complex, Real, Integer, Natural, String, Char, Boolean, Rule, e Undef** definiti nella **standard library**!

basic domain Real

basic domain Integer

...

structured: per costruire insiemi finiti, sequenze, bag, mappe, e tuple a partire da altri domini

Agent e **Reserve** definiti nella **standard library**!

any domain: domini generici; **Any** della **standard library** è il più generico!

Il Linguaggio delle definizioni dei domini

- **abstract**: elementi di natura "astratta", non definiti se non attraverso funzioni definite su tale dominio
- **enum**: enumerazioni, ad esempio
Color = {RED, GREEN, BLUE}
- **concrete domain**: user-defined e subset dei type-domain

Il Linguaggio delle definizioni

Dichiarazioni di **domini concreti** (`dom_declarations`)

[**dynamic**] **domain D subsetof td**

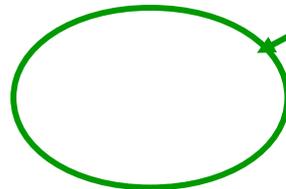
dove:

- D è il nome del dominio da dichiarare
- td è il type-domain di cui D è subset
- La parola chiave **dynamic** è opzionale e denota che l'insieme è *dinamico* (stesso concetto delle funzioni). Per default, un dominio è *statico* e va *definito* nella sezione **definitions**

Esempio:

signature:

domain State subsetof Natural



*Basic domain
della standard library*

definitions:

domain State = {0n,1n}

Il Linguaggio delle definizioni

Dichiarazioni di **type-domain** (`dom_declarations`)

anydomain D

dove D è il nome di un dominio generico e fissato a run-time con un dominio specifico

Abstract TD

abstract domain D

dove D è il nome del type domain

enum domain D = { EL1 | ... | ELn }

dove D è il nome e EL1, ..., ELn le costanti dell'enumerazione

Esempi:

Esempio Lift

- **Lift: un sistema che gestisce più ascensori**

Ogni ascensore può avere due direzioni, e può essere ferma o in movimento

abstract domain Lift

enum domain Dir = {UP | DOWN} //direction

domain Floor subsetof Integer

enum domain State = {HALTING | MOVING} //lift control states

//lift direction of travel, initially UP (see initial state s0)

dynamic controlled direction: Lift -> Dir

dynamic controlled ctrlState: Lift -> State

dynamic controlled floor: Lift -> Floor

Esempio 2

signature:

abstract domain BancomatCard

enum domain Pressure_type = {TOO_LOW | NORMAL |HIGH}

monitored currCard: BancomatCard //n. della carta presente nel bancomat

controlled pressure : Pressure_type

Il Linguaggio delle definizioni

Altri **type-domain** (non dichiarati nella segnatura)

Prod (d_1, d_2, \dots, d_n)

d_1, \dots, d_n sono i domini del prodotto cartesiano

Seq (d)

d è il dominio base delle possibili sequenze

Powerset (d)

d è il dominio base dell'insieme delle parti (l'insieme di tutti i possibili insiemi di elementi di d)

Bag (d)

d è il dominio base dei possibili bag (borsa)

Prod (d_1, d_2, \dots, d_n)

d_1, \dots, d_n sono i domini del prodotto cartesiano

Esempi:

Il Linguaggio delle definizioni

Dichiarazioni di funzioni (`fun_declarations`) da più domini

signature:

`monitored attracted: Prod(Dir, Lift) -> Boolean`

`static totalQuantity: Powerset(Orders) -> Quantity`

`static list:Seq(Integer) // es. list=[1,2,5,8]`

Altri esempi:

monitored `f1: Seq(Integer) -> Boolean`

monitored `f2: Seq(Prod(Integer, Boolean))`

`// es. f2=[(1,true),(5,false)]`

controlled `f3: Boolean -> Prod(Real,Real) //es. f3(true) = (3.0,4.5)`

Struttura e definizioni/inizializzazioni di una ASM

Un FLIP-FLOP device

```
asm flip_flop
import StandardLibrary
signature:
  domain State subsetof Natural
  dynamic controlled ctl_state : State
  dynamic monitored high : Boolean
  dynamic monitored low : Boolean

definitions:
domain State = {0n,1n}

default init initial_state:
  function ctl_state = 0n
```

N.B.: Lo stato del FLIP-FLOP è modellato.

Il Linguaggio delle definizioni

Dichiarazioni(definizioni) di regole (`rule_declarations`)

[**macro**] **rule** R [(x1 in D1,...,xn in Dn)] =
rule

- R è il nome della regola
- xi sono i possibili parametri (opzionali)
- Di sono i domini che fanno da *tipi* per i parametri
- rule è un costruttore di regola (vedi il *Linguaggio delle regole*)
- La parola chiave **macro** è opzionale
- La parola chiave **macro** può essere sostituita da **turbo** per indicare una Turbo regola (una categoria di ASMs non trattate in questo breve corso)

Il Linguaggio delle definizioni

Dichiarazioni(definizioni) di regole (`rule_declarations`)

**[macro] rule R [(x1 in D1,...,xn in Dn)] =
rule**

R = r_Fsm

Esempio:

rule r_Fsm(\$i in State, \$cond in Boolean,\$rule in Rule, \$j in State) =

```
if ctl_state=$i and $cond
then par $rule
      ctl_state := $j
endpar
```

endif

rule: una regola condizionale

Il Linguaggio delle definizioni

Dichiarazioni(definizioni) di assiomi (`axiom_declarations`)

axiom [ID] over id1,...,idn : term

- ID (opzionale) è il nome dell'assioma
- idi sono nomi di domini, funzioni* e regole (con nome) vincolati dall'assioma
- term è un termine che rappresenta l'espressione booleana del vincolo

*In caso di overloading di funzioni, occorre indicare anche il loro dominio, come in $f(D)$ (o $f()$ per funzioni 0-arie) con f nome di funzione e D nome del dominio di f .

Esempio:

axiom `inv_neverBoth` **over** `high(), low(): not(high and low)`

Il Linguaggio delle definizioni

Convenzioni sugli ID:

ID_VARIABLE una stringa che inizia con "\$".

Esempi **\$x** **\$abc** **\$pippo**

ID_ENUM una stringa di lunghezza ≥ 2 , fatta di sole lettere maiuscole. Esempi: **ON** **OFF** **RED**

ID_DOMAIN una stringa che inizia con una lettera maiuscola. Esempi: **Integer** **X** **SetOfBags**
Person

ID_RULE
una stringa che inizia con "r_". Esempi: **r_SetMyPerson**
r_update

ID_FUNCTION
una stringa che inizia con una lettera minuscola diversa da "r_" e da "inv_". Esempi: **plus** **minus** **re**

ID_AXIOM
una stringa che inizia con "inv_". Esempio: **inv_I1**

Il Linguaggio delle definizioni

Commenti (2 forme possibili):

```
// text to be commented
```

```
/* text to be commented*/
```

Il Linguaggio dei termini

$$x+iy \quad x-iy$$

dove X e y sono numeri reali e i l'immagine

Letterale numerico seguito da "n"

Il Linguaggio dei termini



Il Linguaggio dei termini

Variable Term

v

con V nome di variabile (preceduto da \$)

Function Term

$[id .]f [(t1, \dots, tn)]$

dove:

- f è il nome della funzione da applicare
- $(t1, \dots, tn)$ una tupla di termini
- id è il riferimento all'agente (se presente)
che detiene la funzione f

Il Linguaggio dei termini

Variable Term

Function Term `max(2,3)`
 `abs(-4)`
 `abs(max(-2,-8))`
 `self.f(5)` o `f(self,5)`

Il Linguaggio dei termini

$[t_1, \dots, t_n]$ con t_i termini della stessa natura

$[\]$ per la sequenza vuota

$\{t_1, \dots, t_n\}$ con t_i termini della stessa natura

$\{\ \}$ per l'insieme vuoto

$\langle t_1, \dots, t_n \rangle$ con t_i termini della stessa natura

$\langle \ \rangle$ per il bag vuoto

$\{t_1 \rightarrow s_1, \dots, t_n \rightarrow s_n\}$ con t_i termini della stessa natura, e S_i termini della stessa natura pure

$\{- \rightarrow \}$ per la mappa vuota

Il Linguaggio dei termini

["hello","bye"]

[[],[1,2]]

[1..4] \equiv [1,2,3,4]

{[],[1,2],[1]}

{'a','b'}

{1..2,0.5} \equiv {1.0,1.5,2.00}

<1,2,1>

<'a','b','a','b'>

<1..10,2> \equiv <1,3,5,7,9>

Il Linguaggio dei termini

D con **D ID** di un dominio concreto/type-domain,
o termine rappresentante un dominio strutturato

<<R>> con **R** regola

Il Linguaggio dei termini

Integer //ID

prod(Real,String) //prodotto cartesiano

<<skip>>

<<f:=1>>

Il Linguaggio dei termini

if G then tthen [else telse] endif

dove G è un termine booleano, $tthen$ e $telse$ sono termini della stessa natura

LetTerm

let(v1=t1,...,vn=tn)in tv1,...,vn endlet

dove v_i sono variabili e $t_1, \dots, t_n, tv_1, \dots, v_n$ sono termini

Il Linguaggio dei termini

```
if $x>0 then 1  
    else if $x=0 then 0  
        else 0-1  
    endif  
endif  
let ( $double_x = $x+$x )  
    in $double_x * $double_x  
endlet
```

Il Linguaggio dei termini

Comprehension
Term

$[v_1 \text{ in } S_1, \dots, v_n \text{ in } S_n \mid G_{v_1, \dots, v_n} : tv_1, \dots, v_n]$

$\{v_1 \text{ in } D_1, \dots, v_n \text{ in } D_n \mid G_{v_1, \dots, v_n} : tv_1, \dots, v_n\}$

$\langle v_1 \text{ in } B_1, \dots, v_n \text{ in } B_n \mid G_{v_1, \dots, v_n} : tv_1, \dots, v_n \rangle$

$\{v_1 \text{ in } D_1, \dots, v_n \text{ in } D_n \mid G_{v_1, \dots, v_n} : tv_1, \dots, v_n \rightarrow sv_1, \dots, v_n\}$

Comprehension
Term

Il Linguaggio dei termini

Comprehension
Term

[x in $[0..2*n-1]$ | $x \bmod 2 = 0$:
 $g(x)$]

{ x in $\{0..n\}$: $2+x$ }

< x in $\langle 0..n \rangle$: $g(x)$ >

Comprehension
Term

Il Linguaggio dei termini

Quantification
Term

(exist v_1 in D_1, \dots, v_n in D_n
with G_{v_1, \dots, v_n})

ExistUnique
Term

(exist unique v_1 in D_1, \dots, v_n in D_n
with G_{v_1, \dots, v_n})

(forall v_1 in D_1, \dots, v_n in D_n
with G_{v_1, \dots, v_n})

Quantification
Term

Il Linguaggio dei termini

Quantification
Term

(**exist** x in $\{2,5,7\}$ with $x=2$)

ExistUnique
Term

(**exist unique** x in X with $x=0$)

Quantification
Term

Il linguaggio delle regole (alcune)

$$L := t$$

dove t è un termine e L (detta **locazione**) è o un termine funzionale $f(t_1, \dots, t_n)$ con f dinamica e non monitorata, o è una variabile

Il linguaggio delle regole (alcune)

```
let (v1 = t1, ..., vn = tn) in  
  Rv1,...,vn  
endlet
```

dove v_1, \dots, v_n sono variabili, t_1, \dots, t_n sono termini, e R_{v_1, \dots, v_n} è una regola

```
par R1 R2 ... Rn endpar
```

dove R_1, R_2, \dots, R_n sono regole

Il linguaggio delle regole (alcune)

forall v_1 **in** D_1, \dots, v_n **in** D_n
with G_{v_1, \dots, v_n} **do** R_{v_1, \dots, v_n}

dove v_i sono variabili, D_i termini che rappresentano domini, G_{v_1, \dots, v_n} termine booleano che rappresenta la condizione, e R_{v_1, \dots, v_n} è una regola

Il linguaggio delle regole (alcune)

$r [t_1, \dots, t_n]$

dove r è il nome della regola e t_i sono termini che rappresentano gli effettivi argomenti passati

$r[]$

per chiamare una regola che è senza parametri

Il linguaggio delle regole (derivate)



Regole e main rule di una ASM

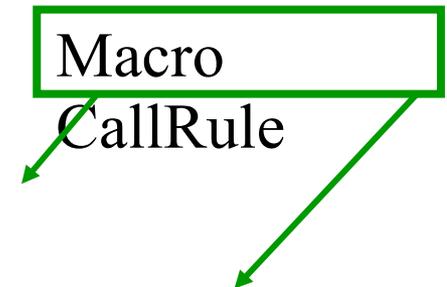
Un FLIP-FLOP device

Macro FSM:

```
rule r_Fsm($i in State, $cond in Boolean, $rule in Rule, $j in State)
= if ctl_state = $i and $cond
  then par
    $rule
    ctl_state := $j
  endpar
endif
```

Main rule:

```
main rule r_flip_flop = par
  r_Fsm[0n, high, <<skip>>,1n]
  r_Fsm[1n, low, <<skip>>,0n]
endpar
```



N.B.: Le transizioni di stato del FLIP-FLOP sono modellate.

A Flip-Flop Device in AsmetaL

Header section

*Clause di Import/export,
vocabolario ASM*

```
asm flip_flop
import StandardLibrary
signature:
  domain State subsetof Natural
  dynamic controlled ctl_state : State
  dynamic monitored high : Boolean
  dynamic monitored low : Boolean
```

Body section

*Static domains/functions
defs., rule decls., e
assiomi*

```
definitions:
domain State = {0n,1n}

rule r_Fsm($i in State, $cond in Boolean, $rule in Rule, $j in
State) =
  if ctl_state=$i and $cond
  then par $rule
    $ctl_state := $j
  endpar
endif

axiom inv_neverBoth over high(), low(): not(high and low)
```

Main rule

```
main rule r_flip_flop = par
  r_Fsm[0,high,<<skip>>,1n]
  r_Fsm[1,low,<<skip>>,0n]
endpar
```

Initialization section

*Inizializzazioni per
domini e funzioni dinamici*

```
default init initial_state:
  function ctl_state = 0n
```

Dining Philosophers in AsmetaL

From: E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Verlag, 2003.

agent program

```
DININGPHILOSOPHER =  
  if owner(resource) = none then owner(resource) := self  
  if owner(resource) = self then owner(resource) := none
```

formally $resource(self) = (leftFork(self), rightFork(self))$

```
rule r_DiningPhilosopher = par r_Eat[ ] r_Think[ ] endpar
```

```
macro rule r_Think =  
  if owner(left_fork(self) = self and owner(right_fork(self)) = self  
  none then par  
  owner(left_fork(self)) = self  
  owner(left_fork(self)) = none  
  endpar endif
```

```
macro rule r_Eat =  
  if owner(left_fork(self) = none and owner(right_fork(self)) = none  
  then par  
  owner(left_fork(self)) = self  
  owner(left_fork(self)) = self  
  endpar endif
```