Logica Relacional:
Formulas

form ::=
expr *in* expr (subset)
|!form (neg)
| form && form (conj)
| form || form (disj)
| *all v* : *type*/form (univ)
| *some v* : *type*/form (exist)

Logica Relacional:
Expresiones

$$
\begin{aligned}
&\text{expr} ::= \\
&\text{expr} + \text{expr (union)} \\
&\mid \text{expr \& expr (intersection)} \\
&\mid \text{expr} - \text{expr (difference)} \\
&\mid \sim \text{expr (transpose)} \\
&\mid \text{expr.expr (navigation)} \\
&\mid +\text{expr (transitive closure)} \\
&\mid \{v : t/\text{form}\} \text{ (set former)} \\
&\mid Var
\end{aligned}
$$

Logica Relacional:
Semantica de las Formulas

$$M : \text{form} \rightarrow env \rightarrow Boolean$$

$$env = (var + type) \rightarrow value$$
$$value = (atom \times \cdots \times atom) +$$
$$(atom \rightarrow value)$$

$$M[a\ in\ b]e = X[a]e \subseteq X[b]e$$
$$M[!F]e = \neg M[F]e$$
$$M[F\&\&G]e = M[F]e \wedge M[G]e$$
$$M[F\ ||\ G]e = M[F]e \vee M[G]e$$
$$M[all\ v : t/F] =$$
$$\bigwedge \{M[F](e \oplus v \mapsto \{\ x\ \})/x \in e(t)\}$$
$$M[some\ v : t/F] =$$
$$\bigvee \{M[F](e \oplus v \mapsto \{\ x\ \})/x \in e(t)\}$$

Logica Relacional:

Semantica de las Expresiones

$$X : \text{expr} \rightarrow env \rightarrow value$$

$$X[a+b]e = X[a]e \cup X[b]e$$
$$X[a\&b]e = X[a]e \cap X[b]e$$
$$X[a-b]e = X[a]e \setminus X[b]e$$
$$X[\sim a]e = \{\, \langle x, y\rangle : \langle y, x\rangle \in X[a]e \,\}$$
$$X[a.b]e = X[a]e ; X[b]e$$
$$X[+a]e = \text{the smallest } r \text{ such that}$$
$$\quad r ; r \subseteq r \text{ and } X[a]e \subseteq r$$
$$X[\{v : t / F\}]e =$$
$$\quad \{x \in e(t) / M[F](e \oplus v \mapsto \{\, x \,\})\}$$
$$X[v]e = e(v)$$
$$X[a[v]]e = \{\langle y_1, \ldots, y_n\rangle /$$
$$\quad \exists x.\, \langle x, y_1, \ldots, y_n\rangle \in e(a) \wedge \langle x\rangle \in e(v)\}$$

El Lenguaje de Especificacion Alloy

- Lenguaje de especificacion cuyo lenguaje y semantica estan basados en la logica relacional.

- Provee mecanismos para definir tipos similar a las clases de los lenguajes orientados a objetos.

- Permite introducir axiomas y operaciones que determinan el modelo formal.

Alloy: Ejemplos: Grafos

```
sig Nodo { }

sig Grafo {
  nodos : set Nodo
  arcos : nodos -> nodos
  }
```

Alloy: Ejemplos:
Grafos Conexos

```
module grafosconexos

sig Nodo { }

one sig Grafo {
	nodos : set Nodo,
	arcos : nodos -> nodos
}

fact conexo {
	all disj n1, n2 : Grafo.nodos | n2 in n1.(^(Grafo.arcos + ~(Grafo.arcos)))
}

assert noTieneAislado{all n : Grafo.nodos | some n.(Grafo.arcos) || some n.(~(Grafo.arcos))}

check noTieneAislado for 5
```

Alloy: Ejemplos:
Grafos Aciclicos

```
module grafoaciclico

sig Nodo { }

one sig Grafo {
	nodos : set Nodo,
	arcos : nodos -> nodos
}

fact acyclic {
	no ^(Grafo.arcos) & iden
}
```

Alloy: Ejemplos:
Grafos Aciclicos Dirigidos

```
module grafoaciclicodirigido

sig Nodo { }

one sig Grafo {
	nodos : set Nodo,
	arcos : nodos -> nodos
}

fact acyclic {
	no ^(Grafo.arcos) & iden
}


fact hasRoot {
	one n : Grafo.nodos | n.(*(Grafo.arcos)) = Grafo.nodos
}


pred conexo() {
	all disj n1, n2 : Grafo.nodos | n2 in n1.(^(Grafo.arcos + ~
(Grafo.arcos)))
}

assert DAGConexo {conexo()}

check DAGConexo for 8
```

Alloy: Ejemplos:
Listas encadenadas

```
module listaSimplementeEncadenada

sig Data {}


sig List {
	first : lone Data,
	next : lone List
}

one sig Empty extends List {}

fact emptyIsEmpty {no Empty.first && no Empty.next}

fact allToEmpty {all l : List | Empty in l.*next}

assert acyclic {all l : List | l not in  l.(^next)}

check acyclic for 7
```

Como obtener Alloy?

- http://alloy.mit.edu
- Disponible para numerosas plataformas.

Fundamentos de DynAlloy: Logica Dinamica

- Logica que permite modelar evolucion de los estados.
- Permite modelar propiedades de la ejecucion de programas secuenciales no-deterministicos.

Sintaxis de la Logica Dinamica

action ::= $a_1, \ldots a_k$ (atomic actions)
| *skip*
| action+action (nondeterministic choice)
| action;action (sequential composition)
| action* (finite iteration)
| dform? (test)

expr ::= *var*
| $f(\text{expr}_1, \ldots, \text{expr}_k)$ ($f \in F$ with arity k)

dform ::= $p(\text{expr}_1, \ldots, \text{expr}_n)$ ($p \in P$ with arity n)
| !dform (negation)
| dform && dform (conjunction)
| dform || dform (disjunction)
| *all* v : *type* | dform (universal)
| *some* v : *type* | dform (existential)
| [action]dform (box)

Semantica de la Logica Dinamica

Q : form $\rightarrow ST \rightarrow Boolean$
P : action $\rightarrow \mathcal{P}\,(ST \times ST)$
Z : expr $\rightarrow ST \rightarrow \mathbf{s}$

$Q[p(t_1, \ldots, t_n)]\mu = (Z[t_1]\mu, \ldots, Z[t_n]\mu) \in env(p)$ (atomic formula)
$Q[!F]\mu = \neg Q[F]\mu$
$Q[F \&\& G]\mu = Q[F]\mu \wedge Q[G]\mu$
$Q[F \mid\mid G]\mu = Q[F]\mu \vee Q[G]\mu$
$Q[all\ v : t \ \mid \ F]\mu = \bigwedge \{Q[F](\mu \oplus v{\mapsto}x) \mid x \in env(t)\}$
$Q[some\ v : t \ \mid \ F]\mu = \bigvee \{Q[F](\mu \oplus v{\mapsto}x) \mid x \in env(t)\}$
$Q[\ [a]F\ ]\mu = \bigwedge \{Q[F]\nu \mid \langle \mu, \nu \rangle \in P(a)\}$

$P[a] = env(a)$ (atomic action)
$P[skip] = \{\ \langle \mu, \mu \rangle : \mu \in ST\ \}$
$P[a + b] = P[a] \cup P[b]$
$P[a;b] = P[a] \circ P[b]$
$P[a^*] = (P[a])^*$
$P[\alpha?] = \{\ \langle \mu, \mu \rangle : Q[\alpha]\mu\ \}$

$Z[v]\mu = \mu(v)$
$Z[f(t_1, \ldots, t_k)]\mu = env(f)(Z[t_1]\mu, \ldots, Z[t_k]\mu)$

Ejemplo de Especificacion en Logica Dinamica

all x : Nat | x = x0 => [A(x)](x = x0+1)

(x=x0 && y=y0) => [Swap(x,y)](x=y0 && y=x0)

(x=x0 && y=y0) => [Swap(x,y) ; Swap(x,y)](x=x0 && y=y0)

Programas via Acciones

Programa atomico via accion atomica (por ejemplo +1, o Swap)

P1 ; P2 ---> T(P1) ; T(P2)

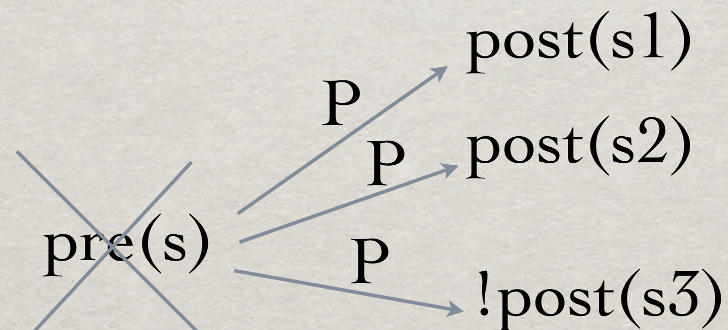
if C then P1 else P2 fi ---> C? ; T(P1) + (!C)? ; T(P2)

while C do P ---> (C? ; T(P))* ; (!C)?

Aserciones de Correccion Parcial

pre => [P]post

La formula es satisfecha for aquellos estados que satisfacen "pre", y para los cuales todo estado alacanzable por P satisface "post".



Como analizar aserciones de correccion parcial?

- Mediante el calculo de la precondicion mas debil de un programa.
- Permite caracterizar los estados que satisfacen la asercion en logica de primer orden.

Calculo de la Precondicion mas Debil

$$
\begin{array}{lcl}
wlp[a(\overline{y}), f] = pre|_{x}^{\overline{y'}} & \Longrightarrow & \text{all } \overline{n} \left(post|_{\overline{x'}}^{\overline{n}}|_{x}^{\overline{y'}} \Longrightarrow f|_{\overline{y'}}^{\overline{n}} \right) \\
wlp[g?, f] & = & g \Longrightarrow f \\
wlp[p_1 + p_2, f] & = & wlp[p_1, f] \wedge wlp[p_2, f] \\
wlp[p_1 ; p_2, f] & = & wlp[p_1, wlp[p_2, f]] \\
wlp[p^*, f] & = & \bigwedge_{i=0}^{\infty} wlp[p^i, f] \ .
\end{array}
$$

Entonces...

Una formula pre => [P]post es valida cuando

pre => wlp(P,post)

es valida.

DynAlloy

- Extension de Alloy para facilitar el analisis de propiedades de ejecuciones.
- Mejor separacion de intereses que utilizando trazas directamente en la especificacion.
- El “DynAlloy Analyzer” permite analizar especificaciones DynAlloy automaticamente.

Sintaxis de DynAlloy

$$formula ::= \ldots \mid \{formula\}\ program\ \{formula\}$$
"partial correctness"

$$\begin{aligned} program ::= &\ \langle formula, formula\rangle(\overline{x}) && \text{"atomic action"} \\ \mid &\ formula? && \text{"test"} \\ \mid &\ program + program && \text{"non-deterministic choice"} \\ \mid &\ program ; program && \text{"sequential composition"} \\ \mid &\ program^* && \text{"iteration"} \end{aligned}$$

Semantica de DynAlloy

$$M[\{\alpha\}p\{\beta\}]e = M[\alpha]e \implies \forall e' \left(\langle e, e' \rangle \in P[p] \implies M[\beta]e'\right)$$

$$P : program \to \mathcal{P}\,(env \times env)$$

$$\begin{array}{lcl}
P[\langle pre, post \rangle] & = & A(\langle pre, post \rangle) \\
P[\alpha?] & = & \{\, \langle e, e' \rangle : M[\alpha]e \wedge e = e' \,\} \\
P[p_1 + p_2] & = & P[p_1] \cup P[p_2] \\
P[p_1 \,; p_2] & = & P[p_1] \,; P[p_2] \\
P[p^*] & = & P[p]^*
\end{array}$$

Ejemplo:

```
sig Addr { }        sig Data { }

abstract sig Memory {
    addrs: set Addr,
    map: addrs -> lone Data
}

sig MainMemory extends Memory {}

sig Cache extends Memory {
    dirty: set addrs
}

sig System {
    cache: Cache,
    main: MainMemory
}
```

Ejemplo (continuacion)

$$\{true\}$$
$$Write(\mathrm{m} : Memory, \mathrm{d} : Data, \mathrm{a} : Addr)$$
$$\{\mathrm{m}'.\mathrm{map} = \mathrm{m}.\mathrm{map} \mathrel{++} (\mathrm{a} \ \rightarrow \ \mathrm{d})\} \ .$$

{ *true* }

SysWrite(s: *System*)

{ some d: *Data*, a: *Addr* |
s'.cache = s.cache ++ (a → d) and
s'.cache.dirty = s.cache.dirty + a and
s'.main = s.main }

{ *true* }

SysFlush(s: *System*)

{ some x: set s.cache.addrs |
s'.cache.map = s.cache.map - x→Data and
s'.cache.dirty = s.cache.dirty - x and
s'.main.map = s.main.map ++
{a: x, d: Data | d = s.cache.map[a]} }

Ejemplo (continuacion)

```
pred DirtyInv(s: System) {
    all a : !s.cache.dirty |
            s.cache.map[a] = s.main.map[a] }
```

{ DirtyInv(s) }
(SysWrite(s) + SysFlush(s))*
{ DirtyInv(s') }