

- **La lógica**, proporciona una formulación **simbólica** e **independiente del dominio** de las leyes del pensamiento humano. Este doble carácter hace posible mecanizar sus técnicas y métodos.

- PROBLEMA:

- Tiene un carácter **estático**, se desarrolló para estudiar objetos matemáticos bien definidos y consistentes, se requieren formas más dinámicas (y menos perfectas) de lógica.
- Los métodos de la lógica resultan más **caros** en términos computacionales, es necesario reducir sus costes.

- SOLUCIÓN: **Lógica computacional**, modela el conocimiento impreciso, incompleto, dinámico, distribuido. Soporta el razonamiento aproximado, temporal, no monótono,...

Lógicas para Aplicaciones software:

Restricciones: - *Lógica ecuacional*

Extensiones de la lógica: - *Lógica difusa* - *Lógica Lineal* - *Lógica Modales*

- Lógica ecuacional:

- Subconjunto de la lógica de primer orden, se eliminan todas las conectivas lógicas. El símbolo de predicado es la (=).
- $s=t$, dos términos son semánticamente iguales, aunque son diferentes sintácticamente.
Ej: $ascii('0') = 48$
- Un conjunto E de ecuaciones con el mismo símbolo "raíz" f en las partes izquierdas de las ecuaciones se describe como "la definición" de f. *$even(0)=true$ - $even(X)=even(X-2)$*

Lógica para la programación:

Restricciones - Estilo de Lenguaje:

- *Lógica ecuacional* - *Funcional (Haskell)*
- *Lógica clasual* - *Relacional (Prolog)*

Extensiones:

- *Lógica many-sorted +tipos*
- *Lógica order-sorted +herencia*
- *Lógica (modal) temporal +concurrency*
- *Lógica (modal) dinámica +objetos*

- **Unificación de lógicas: Lógica de Reescritura (RWL)**, lógica del cambio que permite especificar la dinámica de un sistema.

- Integración "sin costuras" de distintas características: funciones, y tipos, indeterminismo, concurrency, reflexión y genericidad.
- Marco unificado en el que pueden definir distintas lógicas: ecuacional, clasual, lineal.
- Existe una lógica temporal, asociada a la RWL, estrictamente más potente que CTL/LTL: la temporal logic of rewriting (LTR).

- **Maude**:, implementa eficientemente la lógica RWL.

- Soporta de forma natural la especificación formal / modelado / programación en un estilo funcional.
- Distingue entre la parte concurrente y funcional.
- Reescritura módulo listas, conjuntos, multiconjuntos,... mediante atributos ecuacionales.
- Genericidad y tipos ordenados de datos.
- Infraestructura para análisis y verificación formal (alcanzabilidad, model-checking, theorem proving).
- Reflexión como soporte al meta-modelado, ejecución simbólica y construcción rápida de herramientas de soporte.

- Lenguaje Maude

- Sintaxis: Basada en ecuaciones y reglas de reescritura (estilo Haskell, ML, Scheme o Lisp)
- Semántica: Basada en la Lógica de Reescritura (RWL), que modela funciones, concurrencia y objetos.

- Fundamentos de Maude, consta de tres tipos de módulos:

- Módulos funcionales *fmod <conjunto de ecuaciones>endfm*
- Módulos de sistema *mod <conjunto de ecuaciones/reglas de escritura>endm*
- Módulos O2 *omod ... endom*

- E (Ecuaciones):

- Definen funciones confluentes y terminantes.
- Se definen dentro de módulos funcionales.
- E puede incluir un conjunto Ax de Axiomas algebraicos.
- Representan la parte estática del sistema.
- Se aplican de forma determinista.

- R (Reglas de reescritura):

- Definen funciones que pueden ser no confluentes y/o no terminantes.
- Se definen dentro de módulos de sistema.
- Especifican la dinámica del sistema, es decir, acciones que puedan producir transiciones del mismo.

- Paso de reescritura (Maude step), dado un término (o estado) s, un paso de reescritura de t a t' se consigue aplicando una regla de R módulo las ecuaciones de E.

El estado t se simplifica usando las ecuaciones de E hasta alcanzar su forma irreducible (tE) con respecto E.

Una traza de ejecución es una secuencia de Maude steps. Se representa como: $t \longrightarrow^* t'$

- **Reescritura módulos axiomas**, dado $E=E_0 \cup Ax$, las ecuaciones de Ax no se usan para reescribir sino para hacer un pattern-matching especializado. Cuando el conjunto de axiomas algebraicos no es vacío, cada paso de reescritura se hace sustituyendo el pattern-matching convencional por pattern-matching módulo axiomas.

- **Árbol de ejecución**, un árbol de computación es un conjunto de secuencias de Maude steps que se organizan en forma de árbol.

- **Optimización**, Maude mantiene en memoria versiones normalizadas de las reglas y ecuaciones del programa, donde sus lhs están normalizadas respecto de Ax.

En cada Maude step, la fase inicial se descompone en dos normalizaciones, la primera respecto a los axiomas de Ax y la segunda respecto de las ecuaciones de E_0 .

- **Atributos ecuacionales**, cuando aparecen operadores con axiomas asociativos, conmutativos, etc, la reescritura no es capaz de computar de forma efectiva con ellos (espacio de búsqueda infinito). Para ello, dichas propiedades se tratan aparte, como **atributos**.

pares no ordenados — $>comm \quad XY=YX$

listas — $>assoc \quad X(YZ) = (XY)Z$ — $id:null \quad Xnull = nullX = X$

multiconjuntos — $>assoc, comm, id:null$

conjuntos — $>assoc, comm, id:null, idem \quad XX = X$

[$assoc \ idem$] **PROHIBIDO EN MAUDE**

- Otros atributos ecuacionales

- [ctor] — $>símbolo \ constructor$
- [gather (e E)] — $>assoc$ por la derecha
- [item] — $>abrevia \ s(s(..(0)))$ como $sn(0)$
- [ditto] — $>mismos \ axiomas \ que \ el \ anterior$

- **Ecuaciones**, se usan para normalizar los estados. **Reglas**, se usan para hacer evolucionar el sistema produciendo cambios de estado.

- **Estrategias de evaluación**, a diferencia de otros lenguajes funcionales como Haskell, que son perezosos, Maude es un lenguaje impaciente (sólo para las ecuaciones).

Para garantizar la convegenia y terminación de los cómputos, es necesario que las ecuaciones satisfagan buenas condiciones.

- **Terminación**, un programa es terminante si no existen cadenas infinitas de pasos de reducción. Cuando un programa es terminante, la forma normal de cualquier término siempre existe.

La terminación es una propiedad indecidible de los programas.

- **Confluencia**, un programa es confluente si, siempre que un término se puede reescribir a dos términos distintos, éstos, a su vez, convergen a un mismo término.

- **Ortogonalidad** — **>Confluencia**, cuando un programa es confluente, la forma normal de cualquier término, si existe, es única.

- La confluencia es indecidible.
- Hay condiciones suficientes. Ej: la ortogonalidad donde Ortogonalidad = no solapamiento de lh's (ninguna instancia en común) + linealidad por la izquierda (no variables repetidas)

- **Reescritura condicional**, las reglas/ecuaciones condicionales tienen la forma general:

- $\text{crl } l = >r \text{ if } C$
- $\text{ceq } l = >r \text{ if } C$

Donde C es una conjunción de expresiones del tipo: *equation*, *matching equation*, *sort expression*, *rule*.

- **Lógica Fuzzy (borrosa, difusa) - Extensiones de la lógica**, soporta el razonamiento aproximado (útil en robótica, sistemas expertos,...).

- Multivaluada, en vez de binaria, que se deriva -o es una aplicación- de la teoría de "conjuntos Fuzzy", (valores entre 0 y 1)
- Ejemplos: Lukasiewicz, Goedel, Pavelka.

- **Lógica Lineal (Girard, 1987) - Extensiones de la lógica**, permite actualizar la cantidad de recursos tras cada cambio de estado (útil en control de recursos)

- La implicación en lógica lineal se escribe \multimap y se llama "loli"
- En dicha implicación, se consume el recurso A para producir B, las condiciones se modifican tras su uso, es decir, los recursos se consumen al usarlos.
- Nuevas conectivas lógicas: ! of course (replicación), ? why not (borrado)
- Separación en dos clases de las conectivas estándar: "multiplicativa" o simultánea, y otra "aditiva".
- La conjunción $\&$ es una conjunción ya que se puede probar simultáneamente y no es una disyunción.

- **Lógica Modal - Extensiones de la lógica**, nuevos cuantificadores (modales):

- Temporales: UNIVERSAL always, EXISTENCIAL sometimes para formalizar el tiempo, creencias,...
- Dinámicas: Lógicas de la acción: lógica modal para razonar acerca de las acciones y procesos.
- Epistémicas: Lógicas del Conocimiento y de la Creencia/Ignorancia.

- **Lógica del Tiempo**, se desarrollaron por los filósofos para investigar cómo el tiempo se usa en el lenguaje natural de las personas.

Superan la aproximación de primer orden, donde lo habitual es introducir un simple argumento extra para el tiempo.

- **Aproximaciones rivales**, las dos aproximaciones, clásica y temporal, son rivales.

En terminología de BDs:

- *tensers*, son partidarios de la aproximación modal temporal.
- *detenser*, partidarios de la aproximación de primer orden.

- **La Trilogía del Software**, consta de los siguientes componentes de software:

- **Propiedades:** Especificaciones, requerimientos, tipos, modelos (grafos, fórmulas).
- **Juegos de datos:** Seq: trazas, Out: Escenarios, In: Ejemplos
- **Programas:** Código, documentación, Proofs.

- **Procesos Formales de la Trilogía:**

Propiedades a Programas: Derivación formal

Programas a Propiedades: Model checking. Análisis y verificación automática del Software.

Propiedades a Propiedades: Transformación de modelos. Programación automática.

Programas a Programas: Transformación de programas. Especialización, Fragmentación, Refactorización.

Datos a Datos: Minería de datos.

Programas/Propiedades a Datos: Generación automática de casos de prueba.

Datos a Programas/Propiedades: Aprendizaje automático de programas/ propiedades a partir de Ejemplos.