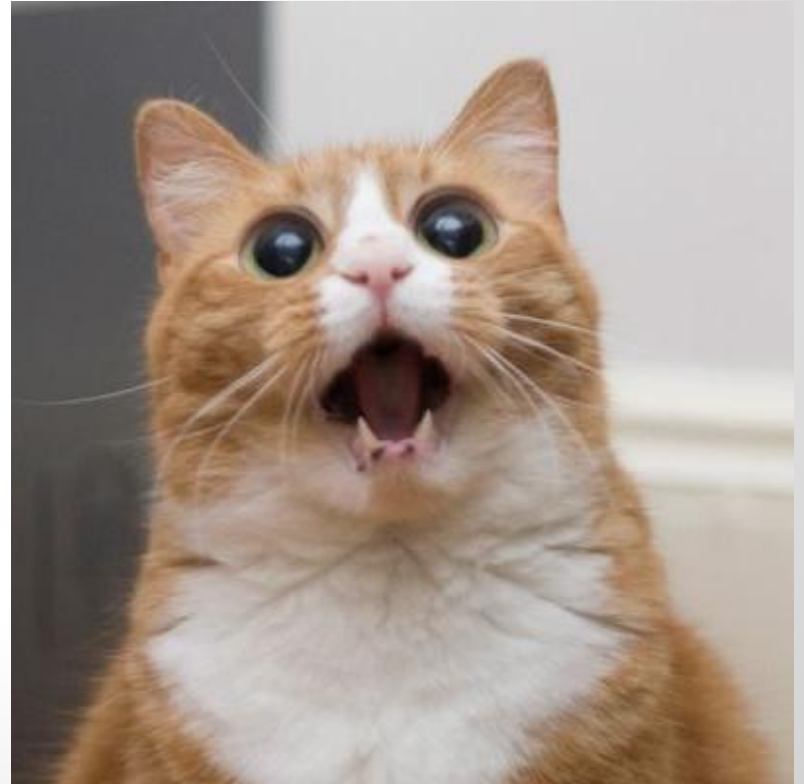


Anubis: Especificando un lenguaje de programación

Gorka Suárez García

La premisa del trabajo...

Cómo especificar un
lenguaje de
programación (como
Java) usando un
lenguaje de
especificación (como
Maude).



El objetivo final.

"¡Pues ahora pienso
montar mi propio
lenguaje! ¡Con
casinos y furcias!"



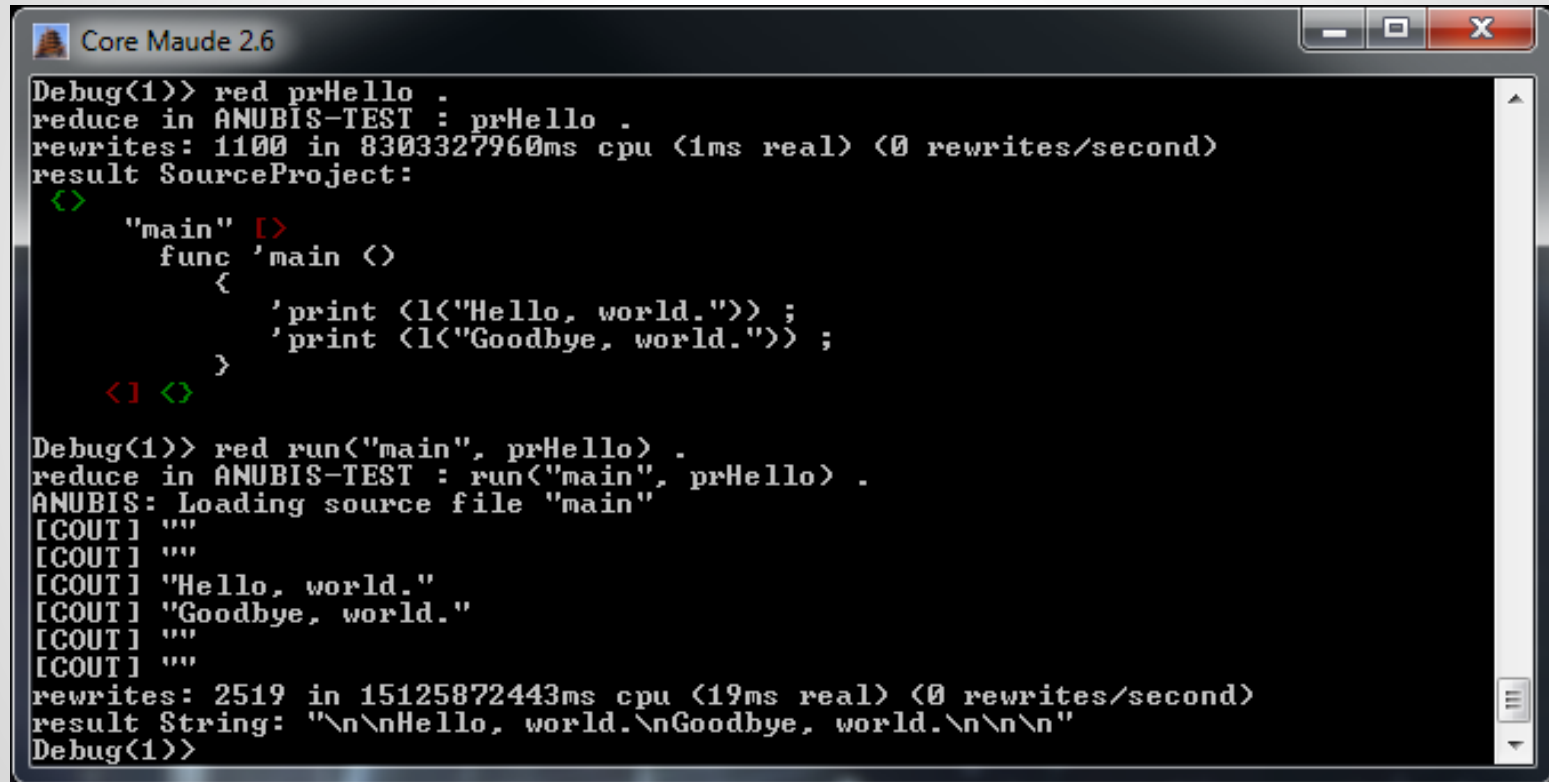
¿Qué ocurrió?

El entusiasmo y la ilusión
están bien... pero tampoco
hay que pasarse...

Vamos, que en mi cabeza
sonaba mejor...



Un pequeño resultado.



```
Core Maude 2.6
Debug(1)>> red prHello .
reduce in ANUBIS-TEST : prHello .
rewrites: 1100 in 8303327960ms cpu (1ms real) (0 rewrites/second)
result SourceProject:
<>
  "main" [>
    func 'main <>
      <
        'print <1<"Hello, world.">> ;
        'print <1<"Goodbye, world.">> ;
      >
    <] <>

Debug(1)>> red run<"main", prHello> .
reduce in ANUBIS-TEST : run<"main", prHello> .
ANUBIS: Loading source file "main"
[COU1] ""
[COU1] ""
[COU1] "Hello, world."
[COU1] "Goodbye, world."
[COU1] ""
[COU1] ""
rewrites: 2519 in 15125872443ms cpu (19ms real) (0 rewrites/second)
result String: "\n\nHello, world.\nGoodbye, world.\n\n\n"
Debug(1)>>
```

¿Qué es un lenguaje?

Declaraciones...

(con)

Instrucciones...

(con)

Expresiones...

(con)

Sangre, sudor y lágrimas...

¡Quiero expresarme!

sort Expression .

op $_+_$: Expression Expression \rightarrow
Expression [ctor assoc] .

¿Asunto resuelto? ^_^U

Problemas de expresión...

¿Tiene sentido hacer?

"Arquímedes" * 3.14

Quizás no...

(Salvo que realmente desees hacer
eso y revolucionar el mundo...)

¡Quiero expresarme MEJOR!

Expression

- + SequenceOfExpressions
- + ValueExpression
 - + BasicValueExpression: BooleanExpression, CharacterExpression, NumericExpression.
 - + ReferenceValueExpression: NullExpression, FunctionExpression, ContainerExpression.

(Y más subtipos que hay...)

¿Cuánto más puede haber?

Expression SequenceOfExpressions ValueExpression BasicValueExpression BooleanExpression
CharacterExpression NumericExpression IntegerExpression RealExpression
ReferenceValueExpression NullExpression FunctionExpression ContainerExpression
ObjectExpression TupleExpression ListExpression StringExpression VariableExpression
ContainerAccessExpression CallFunctionExpression MemberAccessExpression MatchExpression
AssignMatchExpression VOTMatchExpression VariableMatchExpression TypedMatchExpression
TupleOfVariableExpressions ListMatchExpression IdentifierWithType ListMatchExpressionWithType
TupleOfVariableExpressionsWithType SequenceOf2Expressions SequenceOf2IntegerExpressions
SequenceOfMatchExpressions SequenceOfVariableExpressions SequenceOfTypedExpressions
SequenceOfVOTMatchExpressions SequenceOf2VariableExpressions
SequenceWithMatchExpressions SequenceOfValueExpressions GenericFunctionExpression
FunctionParameter ByValFunctionParameter ByRefFunctionParameter ParametersExpression
EmptyParametersExpression CallParametersExpression CallV1VMEParamExpression
CallV1BoolParamExpression CallV1VarParamExpression FunctionParametersExpression
FunctionSignatureExpression

Y esos no eran todos los tipos...



Quiero identificarme...

- + ¿Alguien ha dicho Qid?
- + ¿Los tipos no son identificadores también?
- + ¿Nos vale cualquier cosa?
- + ¿Cómo restringimos los identificadores?
- + Ya puestos, ¿deberíamos tener una jerarquía para los identificadores de tipos?
- + ¿Tiene algo de lo que digo sentido?

¿Y eso se traduce en...?

sorts Identifier TypeIdentifier .

subsort Identifier TypeIdentifier < Qid .

---(Duda existencial: Un identificador de tipo es un subtipo de identificador, pero hay identificadores que son un subtipo de identificador de tipo... MADNESS!)

var ID : Qid .

cmb ID : Identifier if isValidIdentifier(ID) .

cmb ID : TypeIdentifier if isValidTypeIdentifier(ID) .

Todavía quiero expresarme.

op `_+_` : NumericExpression NumericExpression ->
NumericExpression [ctor prec 8 gather(E e)] .

op `_--` : NumericExpression NumericExpression ->
NumericExpression [ctor prec 8 gather(E e)] .

op `_&&_` : BooleanExpression BooleanExpression ->
BooleanExpression [ctor prec 24 gather(E e)] .

op `_||_` : BooleanExpression BooleanExpression ->
BooleanExpression [ctor prec 28 gather(E e)] .

*** Etcétera...

Una gran expresión conlleva una gran responsabilidad...

- + Pensar primero en las reglas que definen nuestros identificadores.
- + Pensar después en el orden de precedencia y la asociatividad de los operadores.
- + Pensar durante días y semanas en las restricciones que queremos imponer en nuestro lenguaje "querido".

Me matas literalmente hablando...

- + Menos mal que Maude tiene literales y de eso no me tengo que preocupar... ¿VERDAD?
- + Probemos expresar: $1 + 1.0$
- + ¡OH DIOS MÍO! ¿POR QUÉ!?
- + Maude tiene sus "trucos" para especificar algunas cosas y operadores definidos e incompatibilidades entre tipos.

¿Y cómo soluciono yo esto?



¡Qid al rescate otra vez!

- + Porque nada da tanto gusto como tener que poner: '1, '2.3, "d', ""eugenio", 'null
- + ¿Y qué pasa con "goodbye horses"?
- + Oh mierda... los espacios en blanco... (>_<)
- + No hay problema, creamos otro tipo más que es gratis y creamos un contenedor:
l("goodbye horses")

Era alguien muy variable...

- + ¿Qué es una variable?
- + ¿A qué huelen los punteros?
- + ¿Cómo se relaciona con todo lo anterior?
- + Digamos que una variable "encaja" en diferentes tipos de expresión.
- + ¿Eso no hace una jerarquía de tipos en Maude demasiado complicada?

Una relación complicada y variable...

subsort Identifier CallFunctionExpression
ContainerAccessExpression MemberAccessExpression <
VariableExpression < BooleanExpression
CharacterExpression IntegerExpression RealExpression
NullExpression FunctionExpression ObjectExpression
TupleExpression ListExpression StringExpression .

¿Pero qué... carajo es esto? (>_<')

Ejemplos algo variables...

// Mi lenguaje fantástico de estar terminado:

var a, b = new Foo();

a = [1, 2, 3, 8]; // Identifier

a[2] = 4; // ContainerAccessExpression

b.x = 1.23; // MemberAccessExpression

var c = b.foo[1]; // CallFunctionExpression

Quedas sentenciado a los isocubos...

- + Jerarquía de tipos de sentencias, por si acaso nos puedan ser útiles.
- + Pensar qué queremos ofrecer en el lenguaje de estructuras de control.
- + Problemas con los paréntesis al haberlos utilizado en las expresiones.
- + Problemas de precedencia y locura.

Declaro que estoy perdido...

- + Las declaraciones son un tipo de sentencia.
- + Pero no toda sentencia podría ser válida como declaración en medio del código.
- + ¿Solución? Usar axiomas para "limpiar" los subtipos "lista" derivados del tipo lista de sentencias (diversión garantizada).

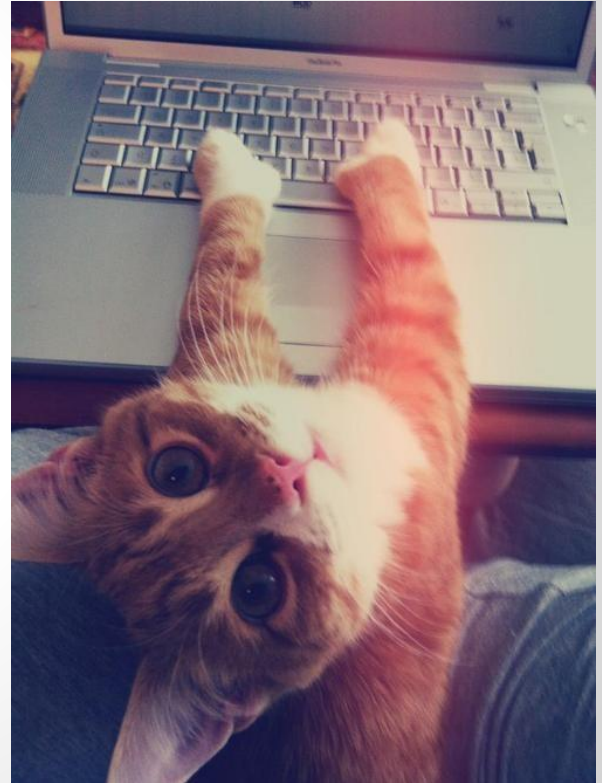
Una declaración sin clase...

- + Las declaraciones dentro de una clase no valen fuera de ella.
- + Tenemos que pensar de nuevo qué queremos permitir, por ejemplo:
 - ¿Queremos tener clases dentro de clases?
- + ¿Problemas? No tener soltura con Maude y liarla parda con el grafo de tipos que tienes.

Y tras la representación...

Un "código fuente" lo forma una lista de declaraciones, ya tenemos nuestro código representado con Maude...

¿Y ahora qué? ^_^



Parseando que es gerundio...

- + Como hemos aplicado restricciones a nuestra sintáxis, Maude nos ayudará a encontrar los errores... ¿VERDAD?
- + Cuando nuestro código está mal tenemos que buscar los fallos y analizar la estructura, para poder dar errores específicos sobre qué está equivocado.

Un mundo ideal...

- + En un programa el código no está de forma lineal para permitir su ejecución secuencial.
- + Almacenar los elementos del código fuente, para no preocuparse del orden.
- + Con tiempo la solución habría sido convertir todo en algún lenguaje "simplificado" para su "ejecución" con Maude.

¿Y qué tiene mi programa?

- + Una lista de alias de tipos.
- + Una lista de tipos (clases).
 - + Una lista de padres.
 - + Una lista de constructores y el destructor.
 - + Una lista de propiedades.
 - + Una lista de campos.
 - + Una lista de métodos.
- + Una lista de funciones.
- + Una lista de lambdas encontradas en la ejecución.

¿Y qué hago con eso?

El "programa" nos sirve para construir la "aplicación", donde tenemos una memoria donde almacenar los tipos de referencia, un ámbito para las variables globales, un contexto para la función actual en ejecución, una pila de contextos y como mínimo un buffer de salida por pantalla.

¿Contexto?

Cuando llamamos a una función creamos una "ruptura" en el código, para ejecutar otro en su lugar. Dentro tenemos una pila de ámbitos, así como otra de instrucciones para salvar por donde íbamos al salir de un ámbito (como ocurriría con un break dentro de un bucle).

¿Ámbito?

Almacena los datos de aquellos tipos que irían en la "pila" y no en el montículo de la aplicación. Su necesidad es debida a que dentro de una función podemos tener bloques de código que generan un ámbito donde podríamos declarar datos con nombre idéntico a otro que está fuera del ámbito actual.

¿Tiene sentido todo esto?

Posiblemente haya formas más sencillas de enfocar la representación de la información, hasta que te planteas hacer objetos con conteo de referencias (como en Java) y ves que el tiempo se te va y que también se te va la pinza en el proceso.

El estado actual del lenguaje...

- + Una especificación de la sintaxis para representar en Maude un código fuente.
- + Algunas funciones de parser para convertir el código fuente en una estructura que pensaba que sería más fácil de utilizar.
- + El inicio del intérprete para poder al menos ejecutar el "hola mundo".

Una cuestión de tiempo y esfuerzo...

- + Ficheros: 36
 - + Líneas con código: 3.632
 - + Total de líneas: 7.176
-
- + Inicio: 9 de diciembre del 2014.
 - + Tiempo: 24 días durante 6 semanas.

Pensé que sería más fácil... ' _ '`



A toro pasado...

- + De primeras lo suyo habría sido hacer algo "como java" y no "como java y más".
- + Aprender primero a hacer un lenguaje con tipos básicos y no meterse en camisa de once varas antes de tiempo.
- + No coger la gripe también estaría chulo para una próxima vez.

¿Con qué me quedo al final?

- + Resulta relativamente fácil y sencillo representar lenguajes con Maude.
- + Teniendo un lenguaje especificado, con su comportamiento en reglas, podríamos realizar model checking y analizar nuestros códigos de forma lógica y automática.
- + Y además ha sido...

Una gran experiencia a pesar de todo...



Referencias

- + Maude
- + Maude Manual
- + Programming Language Design (CS422)
- + Rewriting Logic Semantics of Java

¿Preguntas?

