The webALT Application Grammar

Jordi Saludes, jordi.saludes@upc.edu

20090824

Outline

http://www.webalt.net

Description and Goal

- eContent EU Programme 2004–2007

Description and Goal

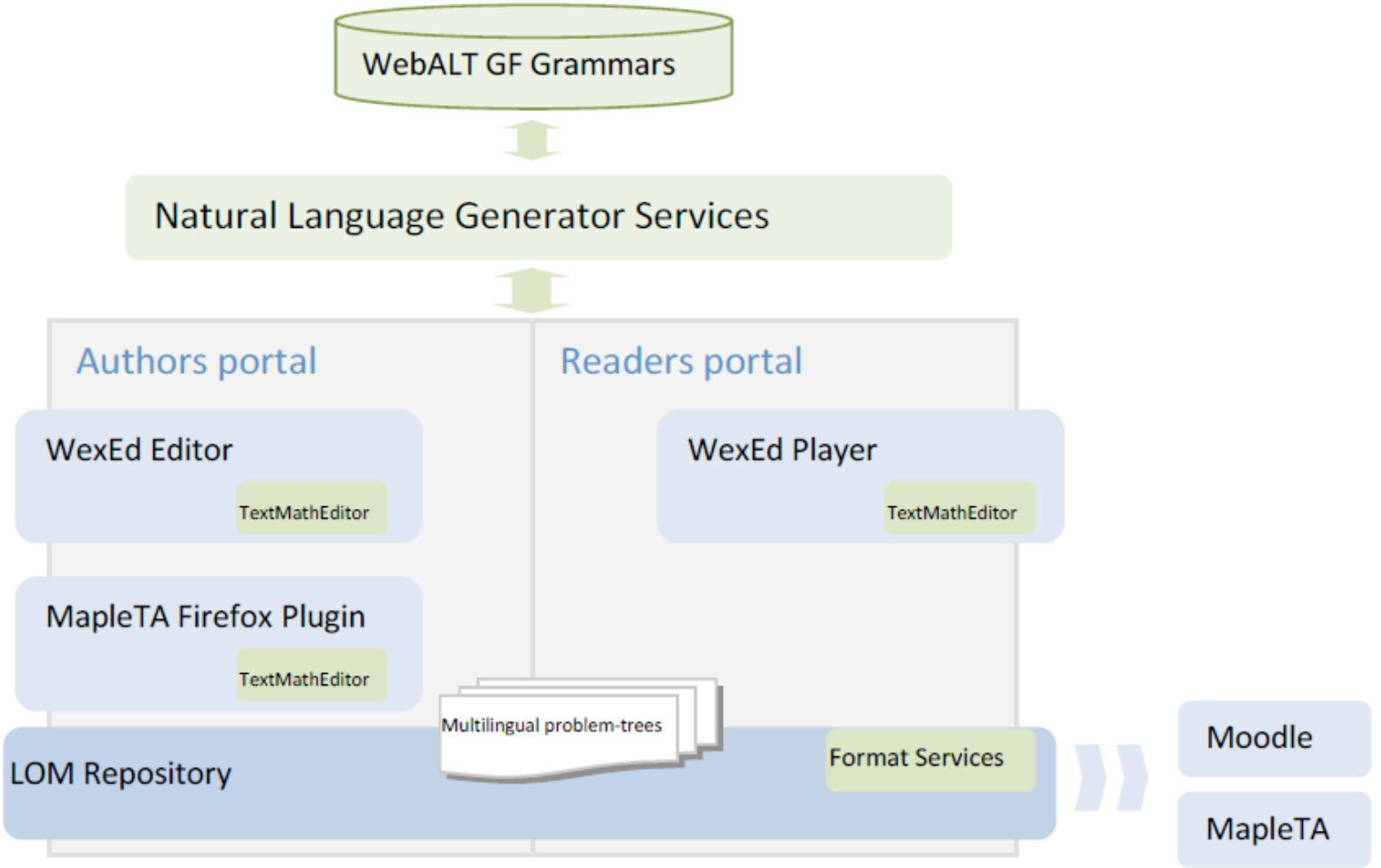
- eContent EU Programme 2004–2007
- Provide a repository of math exercises available in several languages.

Description and Goal

- eContent EU Programme 2004–2007
- Provide a repository of math exercises available in several languages.
  - Retrieving

Description and Goal

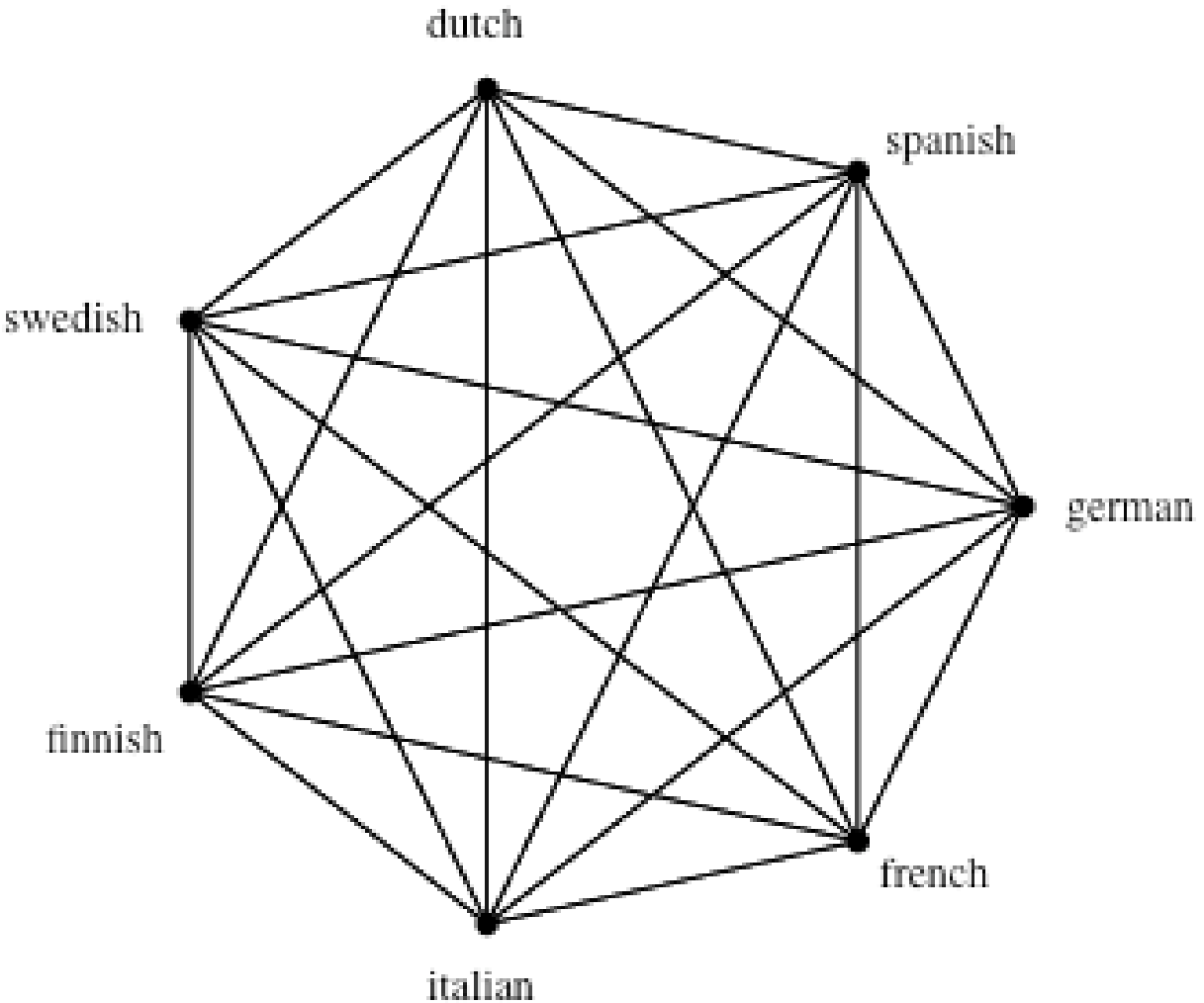
- eContent EU Programme 2004–2007
- Provide a repository of math exercises available in several languages.
  - Retrieving
  - Authoring


WebALT GF Grammars
Natural Language Generator Services
Authors portal
Readers portal
WexEd Editor
TextMathEditor
WexEd Player
TextMathEditor
MapleTA Firefox Plugin
TextMathEditor
Multilingual problem-trees
LOM Repository
Format Services
Moodle
MapleTA

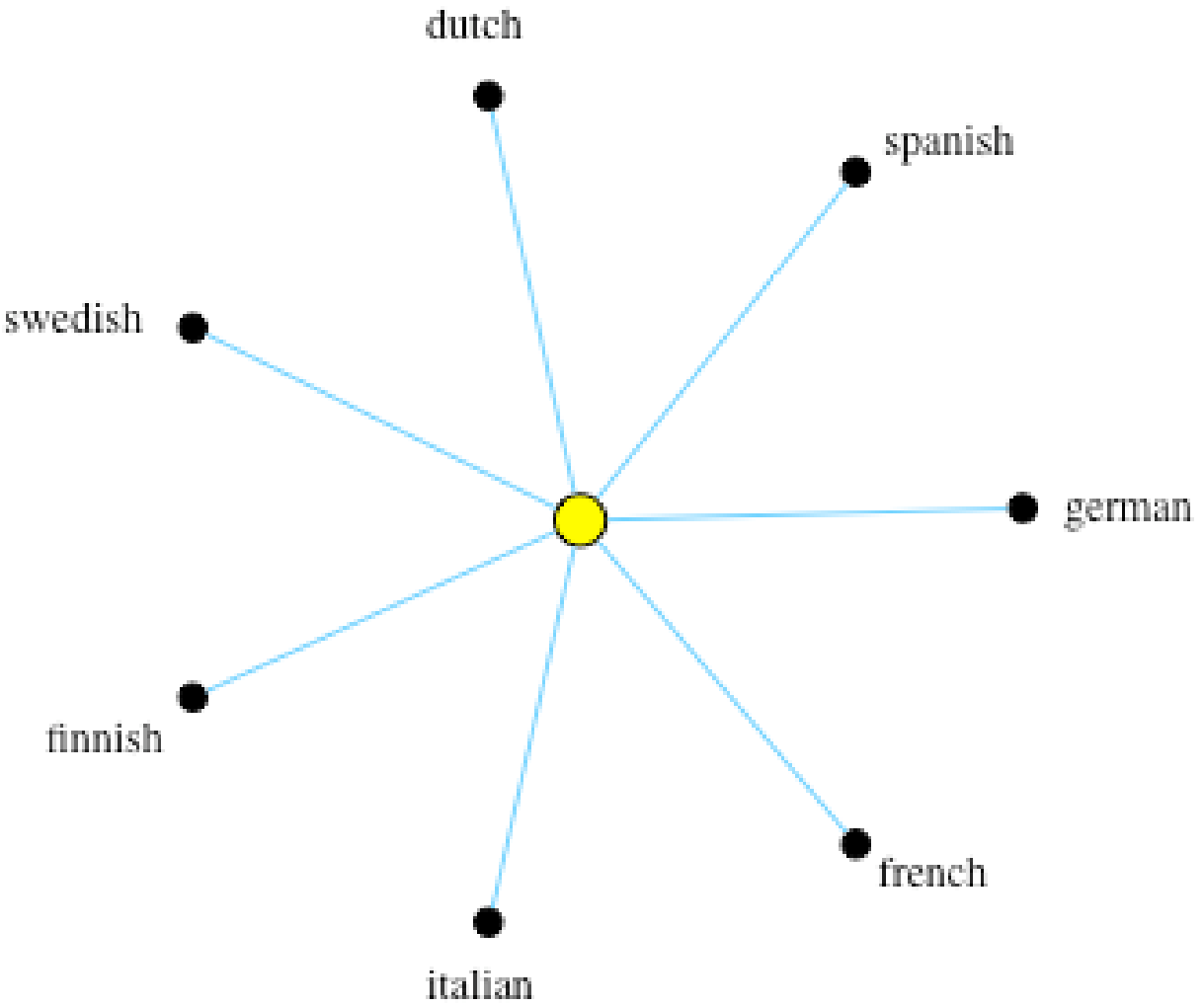
Some results

TextMath Editor

How to do it?



How to do it?



GF enters the stage

Use GF abstract trees formalism as interlingua for:

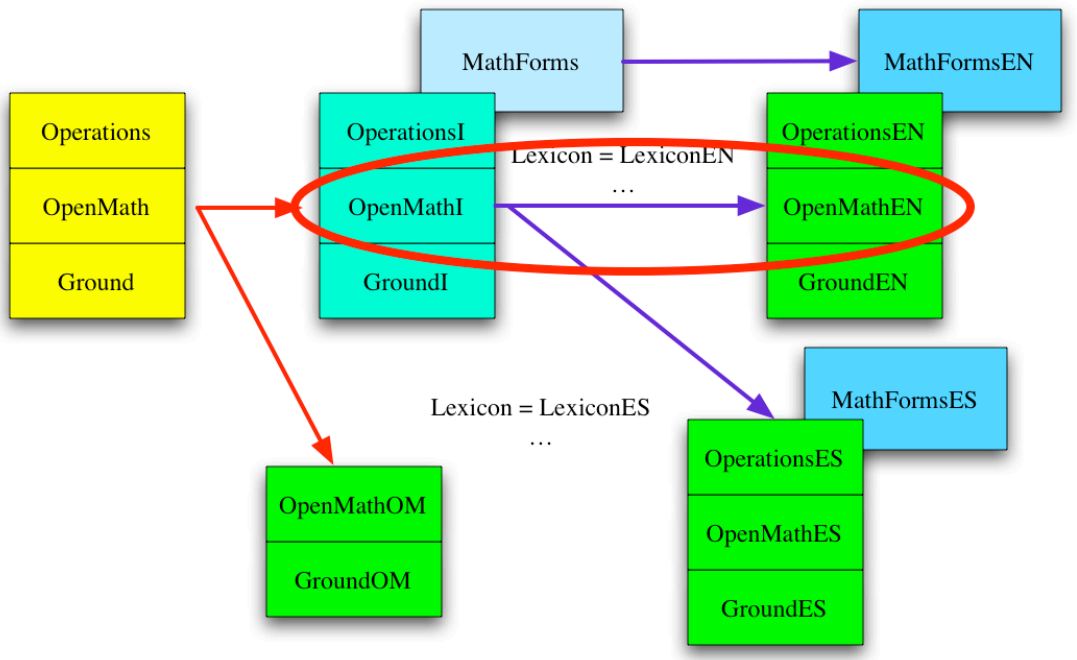
- Catalan
- English
- Finnish
- French
- Italian
- Spanish
- Swedish

Outline

Structure

- Three layers:
  - Operations
  - OpenMath
  - Ground

Structure (2)


MathForms
MathFormsEN
Operations
OpenMath
Ground
OperationsI
OpenMathI
GroundI
Lexicon = LexiconEN
…
OperationsEN
OpenMathEN
GroundEN
Lexicon = LexiconES
…
MathFormsES
OperationsES
OpenMathES
GroundES
OpenMathOM
GroundOM

Atomic constructions

- Variables
  - x, y. They can enter in function definitions.
  - Some variables are defined.
  - Variables of type `t` are promoted to Values of type `t`
- Values
- Literals

Use of the **Symbolic** module.

`Index` A natural number used as index.

`NamedSet` A set with a proper name.

`ValNum`, `ValFun`, `ValSet`,`ValTensor` Values for numbers, functions and tensors (vectors, matrices)

`ValFun3` A function of 2 o 3 variables.

`ValGeo` A geometric value (A plane, a point)

Categories

```
  Index   =                  Ord
NamedSet  =                  CN
ValNum, ... =                NP
 ValFun   ={t:FuncForm; s2:MathVar} ** NP
```

Values vs. Variables

- Values are `ValNum`, `ValFun`, `ValTensor`

Values vs. Variables

- Values are `ValNum`, `ValFun`, `ValTensor`
- Variables are `VarNum`, `VarFun`, `VarTensor`

Values vs. Variables

- Values are `ValNum`, `ValFun`, `ValTensor`
- Variables are `VarNum`, `VarFun`, `VarTensor`
- Better: Coerce Variables to Values as dependent types: `(t: MathType) -> Variable t -> Value t` where `t` is one of `Number`, `Function`, `Set`,`Tensor`.

. . . and lists of these

- Example:
  - `[ValNum]`: A list of two o more numerical values.

. . . and lists of these

- Example:
  - `[ValNum]`: A list of two o more numerical values.
  - `[VarNum]`: A list of two o more numerical variables.

Other categories

`Boxes` These are for the TextMath Editor to display formulas. Boxes are numbered

`Formula` A string to be taken as a formula verbatim.

OpenMath

http://www.openmath.org/
OpenMath defines a *Small Type System* (STS)

Lexicon

Lexicon files use mainly linguistic categories.

Lexicon

Lexicon files use mainly linguistic categories.

The atomic lexicon N and A that enter into the:

Lexicon

Lexicon files use mainly linguistic categories.

The atomic lexicon N and A that enter into the:

Lexicon

Lexicon files use mainly linguistic categories.

The atomic lexicon N and A that enter into the:

The webALT Lexicon

- `N -> CN`

`A -> N -> CN`

Lexicon

Lexicon files use mainly linguistic categories.

The atomic lexicon N and A that enter into the:

The webALT Lexicon

- `N -> CN`

`A -> N -> CN`

More complicated expressions:

Lexicon

Lexicon files use mainly linguistic categories.

The atomic lexicon N and A that enter into the:

The webALT Lexicon

- `N -> CN`

  `A -> N -> CN`

  More complicated expressions:

  - `least_common_multiple_CN`

Lexicon

Lexicon files use mainly linguistic categories.

The atomic lexicon N and A that enter into the:

The webALT Lexicon

- `N -> CN`

  `A -> N -> CN`

  More complicated expressions:

  - `least_common_multiple_CN`
  - `left_composition_CN`

Lexicon

Lexicon files use mainly linguistic categories.

The atomic lexicon N and A that enter into the:

The webALT Lexicon

- `N -> CN`

  `A -> N -> CN`

  More complicated expressions:

  - `least_common_multiple_CN`
  - `left_composition_CN`
  - `sub` **oper** uses + (but is inlined)

The top layer: Operations

- Complete (but simple) exercises.

The top layer: Operations

- Complete (but simple) exercises.
- Hints.

The top layer: Operations

- Complete (but simple) exercises.
- Hints.
- Feedback.

Categories

Asking a student to do something: `Operation`

Compute-like verbs: `CompV`

Question verbs e.g. decide, determine: `QuestV`

Sentence verbs e.g. prove, show: `SentV`

Categories

```
Operation   =  Text (MathOper)
Prop        =  S
SimpleProp  =  Cl ** {p:Polarity} (PCl)
QProp       =  QS
```

Calculate-like verbs

```
ComputeV, CalculateV, SolveV, DefineV, EvalV, FindV,
SimplifyV, DetermineV' : CompV
```

Compute Calculate Solve Define Evaluate Find Simplify Determine	the absolute value of x.

. . . that get combined with a Value

- Meaning: *Compute this value*

```
DoComputeN : CompV -> ValNum -> Operation
```

. . . that get combined with a Value

- Meaning: *Compute this value*

```
DoComputeN : CompV -> ValNum -> Operation
```

- Similarly for functions, sets and tensors.

determine as *decide*

```
DetermineV, DecideV : QuestV ;
DoQuestV : QuestV -> QProp -> Operation ;
```

Determine / Decide | if 2 is even.

`QProp` are for questioning on propositions.

Prove and disprove

```
ProveV, DisproveV, ShowV : SentV ;
DoSentV : SentV -> Prop -> Operation ;
```

Show Prove Disprove	that two is even.

"What?" questions

- `WhatIsN : ValNum -> Operation`
  What is the determinant of M?

"What?" questions

- `WhatIsN : ValNum -> Operation`
  What is the determinant of M?
- Similarly for functions, sets and tensors.

"What?" questions

- `WhatIsN : ValNum -> Operation`
  What is the determinant of M?
- Similarly for functions, sets and tensors.
- Better:
  `WhatIs: (t:MathType) -> Value t -> Operation`

Select operations

- Select something to fulfill a proposition:
  - `DoSelectN : ValNum -> Prop -> Operation`
    Select x such that x is even.
  - `DoSelectFromN : ValNum -> ValSet -> Prop -> Operation`
    Select x from $\mathbb{N}$ such that x is prime.
  - `DoSelectSet : VarNum -> NamedSet -> Prop -> Operation`
    Select an integer x such that x is prime.
- Similarly for functions, sets and tensors

Verbal form for entities

- the sum of x and y versus Add y to x

Verbal form for entities

- the sum of x and y versus Add y to x
- Values are lifted to operations
  `VerbalizeN : ValNum -> Operation`

Verbal form for entities

- the sum of x and y versus Add y to x
- Values are lifted to operations
  `VerbalizeN : ValNum -> Operation`
- This is just a placeholder!

Verbal forms of OpenMath symbols

```
arith1_divide_v, arith1_minus_v,
arith1_plus_v : (x,y:ValNum) -> Operation
arith1_times_v,
arith1_power_v : (x,y:ValNum) -> Operation
arith1_unary_minus_v : ValNum -> Operation
```

Example

```
def
    VerbalizeN (arith1_plus x y)  = arith1_plus_v x y
```

Example

```
def
    VerbalizeN (arith1_plus x y)  = arith1_plus_v x y
```

- From: *(Verb) The sum of x and y*

Example

```
def
    VerbalizeN (arith1_plus x y)  = arith1_plus_v x y
```

- From: *(Verb) The sum of x and y*
- To: *Add y to x*

Example

- `calculus1_diff_v : ValFun -> Operation`
  - From: *`(verb.)` the derivative of f*
  - To: *Differentiate f*

Example

- `calculus1_diff_v : ValFun -> Operation`
  - From: *(verb.) the derivative of f*
  - To: *Differentiate f*
- `calculus1_diff_at_v : ValFun -> ValNum -> Operation`
  - From: *(verb.) the derivative of f at x*
  - To: *Differentiate f at x*:

Example

- Invert f:

  ```
  fns1_inverse_v : ValFun -> Operation
  ```

- Transpose M:

  ```
  linalg1_transpose_v : ValTensor -> Operation
  ```

Combining operations

Twenty is even. Divide twenty by 2:

```
Declare : Prop -> Operation -> Operation
```

Append operations to operations

```
Combine : (t1,t2:Operation) -> Operation
```

Arithmetic operators

Corresponds to **Arith1** OpenMath Content Dictionary

- `arith1_gcd : [ValNum] -> ValNum ;`
- `arith1_power : ValNum -> ValNum -> ValNum ;`
- `webalt_power2, webalt_power3 : ValNum -> ValNum ;`
- `arith1_abs : ValNum -> ValNum ;`
- `arith1_times : [ValNum] -> ValNum ;`
- `arith1_unary_minus : ValNum -> ValNum ;`
- `arith1_plus : [ValNum] -> ValNum ;`
- `arith1_root : ValNum -> Index -> ValNum ;`
- `webalt_root2, webalt_root3 : ValNum -> ValNum ;`
- `arith1_divide : ValNum -> ValNum -> ValNum ;`
- `arith1_sum, arith1_product  ValSet -> ValNum -> ValNum`

Left-associative mathematical operations are usually acting on `[ValNum]`.

Left-associative mathematical operations are usually acting on `[ValNum]`.

- `arith1_times : [ValNum] -> ValNum ;`

Left-associative mathematical operations are usually acting on `[ValNum]`.

- `arith1_times : [ValNum] -> ValNum ;`
- `arith1_plus : [ValNum] -> ValNum ;`

Tree transformations for clarity

```
def
    arith1_root x two = webalt_root2 x
    arith1_root x three = webalt_root3 x
```

Calculus

```
calculus1_diff, calculus1_int : ValFun -> ValFun ;
calculus1_defint : ValSet -> ValFun -> ValNum ;
calculus1_defint_interval : ValFun -> ValNum -> ValNum -> V
calculus1_nthdiff : Index -> ValFun -> ValFun ;
calculus1_partialdiff : Index -> ValFun -> ValFun ;
```

How a function is defined

- Category `ValFun` = `MathFunc` in resource grammar.

How a function is defined

- Category `ValFun` = `MathFunc` in resource grammar.
- ```
  oper
     MathFunc : Type = {t:FuncForm; s2: MathVar} ** NP ;
  param
     FuncForm = FNoVar | FNamed | FVar | FGral ;
  ```
  ```

How a function is defined

- Category `ValFun` = `MathFunc` in resource grammar.
- ```
  oper
      MathFunc : Type = {t:FuncForm; s2: MathVar} ** NP ;
  param
      FuncForm = FNoVar | FNamed | FVar | FGral ;
  ```
- It is a NP with intrinsic attributes:
  ```

How a function is defined

- Category `ValFun` = `MathFunc` in resource grammar.
- oper

```
    MathFunc : Type = {t:FuncForm; s2: MathVar} ** NP ;
```

  param

```
    FuncForm = FNoVar | FNamed | FVar | FGral ;
```

- It is a NP with intrinsic attributes:
  - A function form (`FuncForm`)

How a function is defined

- Category `ValFun` = `MathFunc` in resource grammar.
- oper

```
    MathFunc : Type = {t:FuncForm; s2: MathVar} ** NP ;
```

  param

```
    FuncForm = FNoVar | FNamed | FVar | FGral ;
```

- It is a NP with intrinsic attributes:
  - A function form (`FuncForm`)
  - A variable

Value of a function at a point

```
fun
    At : ValFun -> ValNum -> ValNum
```

implemented by `at_fn` at resources.

Value of a function at a point (2)

```
at_fn : MathFunc -> MathObj -> MathObj = \f,x ->
 case f.t of {
  FNoVar => variants {} ; --TODO
  FGral => NPfn (adverbNP (atAdv x) f) ;
  FNamed => NPfn (adverbNP (possessAdv x) f) ;
  FVar => NPfn (adverbNP (whereIs (useVar f.s2) x) f) }
```

FNoVar f at 3

FGral the primitive of the exponential at 3

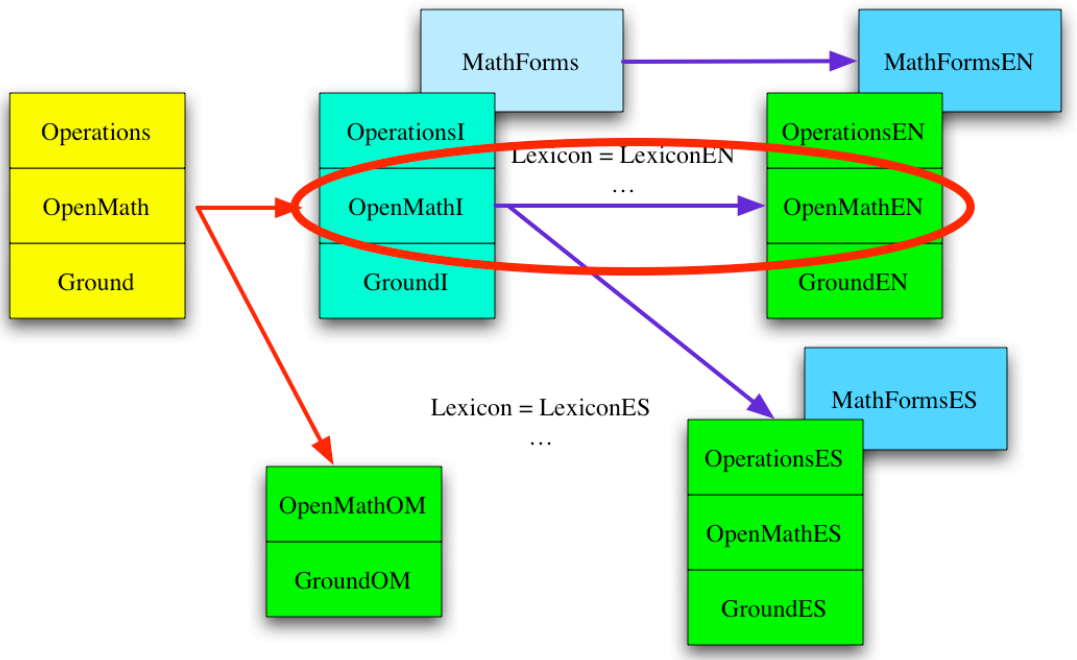
FNamed the exponential of 3

FVar the sum of y and pi where y is 3

Resource files

Shared operations live in `Mathforms*.gf` files

Structure (2)


MathForms
MathFormsEN
Operations
OpenMath
Ground
OperationsI
OpenMathI
GroundI
Lexicon = LexiconEN
…
OperationsEN
OpenMathEN
GroundEN
Lexicon = LexiconES
…
MathFormsES
OperationsES
OpenMathES
GroundES
OpenMathOM
GroundOM

Some examples: Parameters

```
param
        NVars = One | Many ;
        Tends = TNone | TAbove | TBelow | TBothSides ;
```

Tends

```
limit1_limit :
  ValNum -> TendsTo -> VarNum -> ValNum -> ValNum;
limit1_both_sides, limit1_above,
limit1_below, limit1_null : TendsTo

limit1_limit = limitTendNP limitAdv
limit1_null = mkTend TNone ;
limit1_above = mkTend TAbove ;
limit1_below = mkTend TBelow ;
limit1_both_sides = mkTend TBothSides ;
```

Example

(and error!)

```
limit1_limit (int2Num 2) limit1_null x (At transc1_exp (Va
```

Eng the limit of the exponential of x when x tends to 2

Spa * el límite de la exponencial de x cuando x tienda a 2

Explanation

```
limitCN : MathObj -> MathVar -> MathObj -> CN = \x0,x,fx ->
        let
                tendS : MathObj -> MathObj -> FullProp = \v
                  posCl (mkCl v (tend_to c))
in modCN (modCN limit_CN (possessAdv fx))
                  (mkAdv as_Subj (tendS (useVar x) x0)) ;
```

Why is a String needed in STend?

```
Test> p -cat=ValNum "the limit of the exponential of x when

limit1_limit (int2Num 1) ?2371 x (At transc1_exp (Var2Num x
limit1_limit (int2Num 1) ?2381 x (At transc1_exp (Var2Num x
```

Kinds of functions

```
Test> p -cat=ValNum -lang=TestEng "the limit of the right
```

```
el límit de la composició per la dreta de la tangent i de
```

x is less than or equal to y vs. x is less than y or equal to y

Outline

past (and present) shortcomings

- Slow parsing

past (and present) shortcomings

- Slow parsing
- Fragile parsing

past (and present) shortcomings

- Slow parsing
- Fragile parsing
- Not enough coverage

Future work: MOLTO

- Statistical MT approach

Future work: MOLTO

- Statistical MT approach
- Incremental compilation

Future work: MOLTO

- Statistical MT approach
- Incremental compilation
- Grammarian/Translator tools

Future of webALT grammar

Grow horizontally More content dictionaries.

Future of webALT grammar

Grow horizontally More content dictionaries.

Grow vertically More complicated expressions.

future of webALT grammar (2)

Possible application:

future of webALT grammar (2)

Possible application:

- webALT company

future of webALT grammar (2)

Possible application:

- webALT company
- MOLTO study case

future of webALT grammar (2)

Possible application:

- webALT company
- MOLTO study case
- Dialog managers, proof assistants,...

Why should we use the webALT grammar?

- Do not reinvent the wheel

Why should we use the webALT grammar?

- Do not reinvent the wheel
- It is free (LGPL).

Why should we use the webALT grammar?

- Do not reinvent the wheel
- It is free (LGPL).
- It is on a boat that has forward momentum.

Thanks!