Efficient Ambiguous Parsing of Mathematical Formulae

Claudio Sacerdoti Coen
<sacerdot@cs.unibo.it>

Stefano Zacchiroli
<zacchiro@cs.unibo.it>

University of Bologna

December 16, 2004 - Jouy-en-Josas (France)

Project PCRI, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

MoWGLI: Math on the Web, Get it by Logic and Interfaces, EU IST-2001-33562.

Summary

1. Overloading in formal mathematics
2. Type based parsing: the inefficient way
3. Exploiting metavariables
4. Benchmarks and comparisons

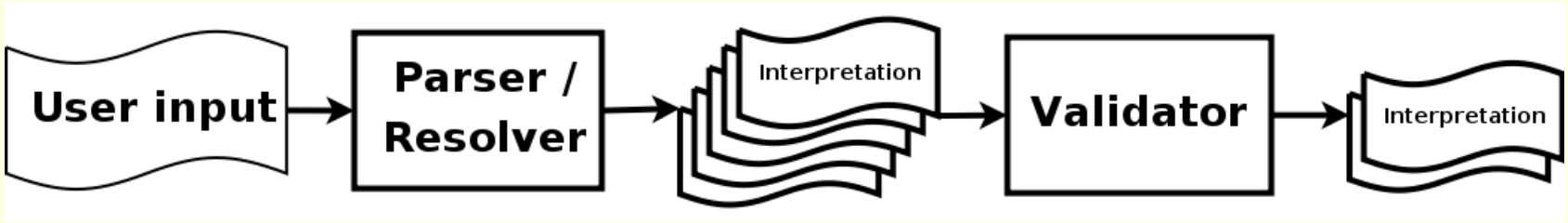
Forms of Ambiguity

$$(1 + 2) * 3 = 9$$

- natural, integer, rational, real, complex, . . . ?
- which representation?
  Peano numbers, Church numerals, binary integers, . . .
- which equality?
  Leibniz's polymorphic equality, decidable monomorphic equality for natural numbers, undecidable equality for real numbers, . . . ?
- coercions? implicit arguments?

Only a few combinations of choices are well-typed

A naive solution



Desired properties

1. compositionality: the semantic analysis phase should be as separate as possible from the parsing phase
2. efficiency: semantic errors must be detected as early as possible, avoiding duplicate efforts. $(1_? +_? 2_?) *_{\mathbb{R}} 3_? =_{\mathbb{N}} 9_?$

Laziness is not enough

$$\dfrac{\bigvee_{g : B \to T'} \qquad \dfrac{\bigvee_{\Gamma \vdash f : \Pi x : A.T(x)} \quad \bigvee_{\Gamma \vdash M : A'} \quad \bigvee_{\Gamma \vdash A \equiv A'}}{\Gamma \vdash (f\ M) : (T\ M)} \qquad \bigvee_{\Gamma \vdash (T\ M) \equiv B}}{\Gamma \vdash (g\ (f\ M)) : T'}$$

- a mismatch between the output type of f and the input type of g is detected by $\Gamma \vdash (T\ M) \equiv B$ ...
- ... but only after the recursion $\Gamma \vdash M : A'$
- $\Gamma \vdash M : A'$ cannot be postponed (otherwise $\Gamma \vdash (T\ M) \equiv B$ can diverge)

Exploiting metavariables

$$\dfrac{\bigvee_{g : B \to T'} \qquad \dfrac{\bigvee_{\Gamma \vdash f : \Pi x : A.T(x)} \quad \Gamma \vdash ?_1 :?_2 \quad \bigvee_{\Gamma \vdash A \equiv ?_2}}{\Gamma \vdash (f\ ?_1) : (T\ ?_1)} \qquad \bigvee_{\Gamma \vdash (T\ ?_1) \equiv B}}{\Gamma \vdash (g\ (f\ ?_1)) : T'}$$

- introduce metavariable in the calculus
- replace conversion with unification and type-checking with refinement
- the unification and refinement judgements are three valued:
  σ (Keep) ϵ (Prune) $\bot$ (Keep)

An example [1/2]

E.g.: parsing $(5/2)!$ where $/ \in \{/_{\mathbb{N}}, /_{\mathbb{R}}\}$ and $2, 5 \in \mathbb{N} \cup \mathbb{R}$

$$D = \begin{cases} / & \mapsto & [/_{\mathbb{N}} : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \ ; \ /_{\mathbb{R}} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}] \\ 2 & \mapsto & [2 : \mathbb{N} \ ; \ 2 : \mathbb{R}] \\ 5 & \mapsto & [5 : \mathbb{N} \ ; \ 5 : \mathbb{R}] \end{cases}$$

$$\phi_0 = \phi_{???} \begin{cases} / & \mapsto & PlaceHolder \\ 2 & \mapsto & PlaceHolder \\ 5 & \mapsto & PlaceHolder \end{cases}$$

$$\Phi_0 = \{\phi_0\}$$

$$refine(t_{\uparrow}(\phi_0)) =$$
$$refine(?_1!) = [?_1 : \mathbb{N}]$$

An example [2/2]

$$\phi_{1??} = \begin{cases} / & \mapsto & /_{\mathbb{N}} \\ 2 & \mapsto & PlaceHolder \\ 5 & \mapsto & PlaceHolder \end{cases}$$

$$refine(t_{\uparrow}(\phi_1)) = refine((?_1 /_{\mathbb{N}} ?_2)!) = [?_1 : \mathbb{N} \; ; \; ?_2 : \mathbb{N}]$$

$$\phi_{2??} = \begin{cases} / & \mapsto & /_{\mathbb{R}} \\ 2 & \mapsto & PlaceHolder \\ 5 & \mapsto & PlaceHolder \end{cases}$$

$$refine(t_{\uparrow}(\phi_1)) = refine((?_1 /_{\mathbb{R}} ?_2)!) = \epsilon$$

$\Phi_1 = \{\phi_{1??}\}$ 4 ASTs pruned at once

...

$\Phi_2 = \{\phi_{11?}\}$ 2 ASTs pruned at once; $t_{\uparrow}(\phi_{11?}) = (2_{\mathbb{N}} \; /_{\mathbb{N}} \; ?_1)!$

...

$\Phi_3 = \{\phi_{111}\}$ 1 AST pruned; $t_{\uparrow}(\phi_{111}) = (2_{\mathbb{N}} \; /_{\mathbb{N}} \; 5_{\mathbb{N}})!$ is well typed and closed

Benchmarks

NA	PA	I	NA	PA	I	NA	PA	I	NA	PA	I
1	2	-1	32	33	-1	128	38	90	1280	43	1237
1	4	-3	32	33	-1	128	38	90	1536	54	1482
1	6	-5	32	33	-1	160	38	122	1536	48	1488
3	5	-2	32	33	-1	192	108	84	1680	35	1645
4	6	-2	32	33	-1	192	111	81	1792	51	1741
4	7	-3	32	39	-7	224	46	178	2688	54	2634
8	7	1	32	33	-1	224	45	179	3584	55	3529
14	24	-10	42	13	29	224	47	177	3584	54	3530
21	13	8	63	83	-20	256	40	216	7168	63	7105
32	32	0	63	83	-20	320	20	300	8192	60	8132
32	33	-1	63	19	44	480	41	439	14336	62	14274
32	38	-6	96	37	59	512	45	467	21504	65	21439
32	33	-1	128	40	88	512	41	471	21504	60	21444
32	33	-1	128	43	85	896	51	845	36864	79	36785
32	33	-1	128	42	86	896	51	845	53760	65	53695
32	33	-1	128	42	86	896	51	845	53760	67	53693
32	40	-8	128	39	89	896	49	847			
32	39	-7	128	38	90	896	47	849			
32	33	-1	128	38	90	1024	42	982			
32	33	-1	128	38	90	1280	44	1236			
32	33	-1	128	38	90	1280	43	1237			

Comparison to Coq notational scopes [1/3]

Coq notational scopes:

- one active scope at a time (e.g. nat, R, $types$)
- no overloading inside one scope
- a concrete syntax to change the scope (e.g. $(\pi + (5/2)\%nat)\%R$)
- notational scopes associated to operator arguments
  (e.g. $length : \forall A\%types, (listA)\%list \to nat$)
  Cfr. Weak Type Theory

Comparison to Coq notational scopes [2/3]

Pros:

- Since the weak type system is unrelated to the Coq type system it is more general.
- Maximal modularity.

Comparison to Coq notational scopes [3/3]

Cons:

- Serious problem of consistency between the two type systems (the weak types are assigned by the user).
- Puts burden on the user.
- Does not really exploits much of the environment even if available.
- Weaker then our approach. E.g.$id : \forall T : \mathbf{Set}.T \to T$ No scope is associated to the second argument T. Thus $(id\ \mathbb{N}\ 0)$ must be written as $(id\ \mathbb{N}\ 0\%nat)$ (even if $(id\ \mathbb{N})$ is known to have type $\mathbb{N} \to \mathbb{N}$).