# Epigram

## MPS-based editor for Epigram programming language

### What is Epigram?

Epigram system, being invented by Conor McBride, is an implementation of dependently typed functional programming language Epigram, designed by Conor McBride himself and James McKinna. (See their paper The View From The Left). Good Epigram tutorial you can find here. Also see Wikipedia.

### What are dependent types?

Dependent type systems, as invented by Per Martin-Löf, generalize the simple function space S -> T (the type of functions from type S to type T), familiar from traditional functional languages, to the dependent function space x:S -> T, where T may mention - hence depend on x.

Though universal quantification is often used in many type systems to capture polymorphism, the quantification is usually over types. Dependent type systems allow quantification over all values - which now include the types. Simple function space S -> T is a particlular case of dependent function space x:S -> T, when T doesn't depend on x.
(From Epigram tutorial)

### Difficulties with programming in Epigram

When programming in currently available version of Epigram, a.k.a. Durham Epigram, you should use Xemacs as a user interface. Epigram receives all Xemacs events represented in Elisp, and responds with some other Elisp. Xemacs interprets that Elisp and shows result. Hence Epigram doesn't use any abilities of Xemacs but typing letters in different colors and receiving key and mouse events.

Then, when you have succesfully sent some text to Epigram, you can't edit it anymore. It becomes typechecked, elaborated and memoized, and coloured for user, but completely uneditable. There's no local undo in current Epigram - such that could "unsend" some part of program from Epigram and make it editable. Currently, Undo command can only undo last action.

Hence, though Epigram supports some interactivity and just-in-tyme type-driven smart completion, which highly decreases an amount of manually written code, I'd rather have more convenient editor to write programs in Epigram.

## MPS - language development environment

For those who do not have any idea about what MPS is, it's recommended to read Sergey Dmitriev's article. I can't add more. Also visit MPS homepage.
Maybe the best way to start playing with MPS is to read MPS tutorial.

## MPgram - MPS-based Epigram editor

The two-dimensional syntax of Epigram fits an idea of cell-based editing quite well. Why not to create Epigram language in MPS ?
The main purpose for me was (and remains) to develop a kind of IDE for Epigram language, (using all the power of MPS, of course) which should provide convenient user interface for editing programs and interacting with Epigram elaborator itself.

### MPgram abilities

#### First, just editing.

#### MPgram supports scope checking and scope-driven autocompletion

In fact, scope control.
When a programmer uses an identifier, that identifier usually must be already defined. In MPgram, when a programmer wants to use an identifier, he has to use already defined identifier - because MPgram knows which identifiers are visible in that point of program and does not allow to use identifiers which are unknown or out of scope. It looks like autocompletion: when a programmer starts typing an identifier, the system gives him a popup list with known identifiers' names starting with prefix he

has just typed. He can choose one of them or ask for autocompletion or continue typing, but anyway, if his identifier is not in scope - hence not in the popup list - he'll fail entering that identifier.

## Automatic generation of some evident constructions, and automatic banning of some evidently wrong constructions.

Some expressions or subexpressions, though having many ways to be written syntactically correct, have only one possibility to be semantically good. Why not to leave a job of writing the most evident constructions to the editor?
Another expressions, though syntactically correct, are definitely wrong. Why not to let the editor check the most evident such cases?

- When a programmer enters a known identifier, MPgram finds out an arity of that identifier and generates the appropriate quantity of empty cells - placeholders - for parameters.
- In a data-where declaration, all data constructors defined in where clause should return type defined in data clause. When a programmer creates a new data constructor definition, MPgram generates an appropriate identifier with appropriate number of parameter placeholders as a type of that data constructor to be defined.
- MPgram does not allow usages of some function within its own body, if not within block started with rec eliminator.

## Indroducing variables

Sometimes one is lazy to define all required identifiers before using them. In MPgram, one can introduce new identifier just in time. It is marked as unknown and becomes visible everywhere, like a globally defined identifier (i.e. defined in data-where or l et declaration). One can define such identifier later, by creating a declaration for the identifier with the same name and asking the editor to replace all usages of unknown identifiers with this name with the identifier just defined.

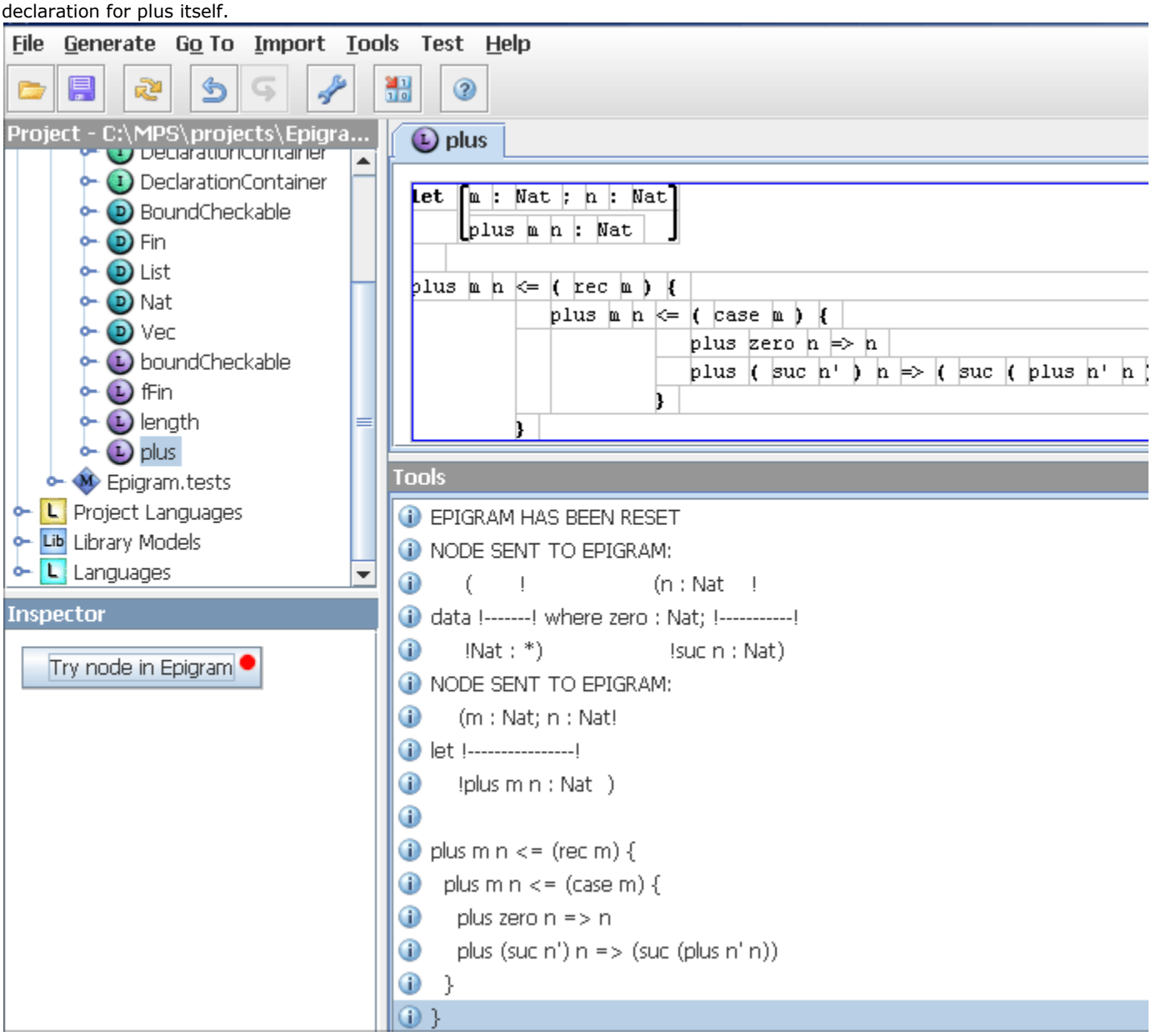
## Sending programs to elaborator

Well, you've written a program. You want to launch your program, don't you? MPgram can run programs using Epigram elaborator. You may send every kind of declaration (i.e., data-where, let or inspect declarations) to the elaborator.

NB: you do not need to think about what another declarations you need to send before this one. Editor does that work for you and sends all required declarations in the right order.

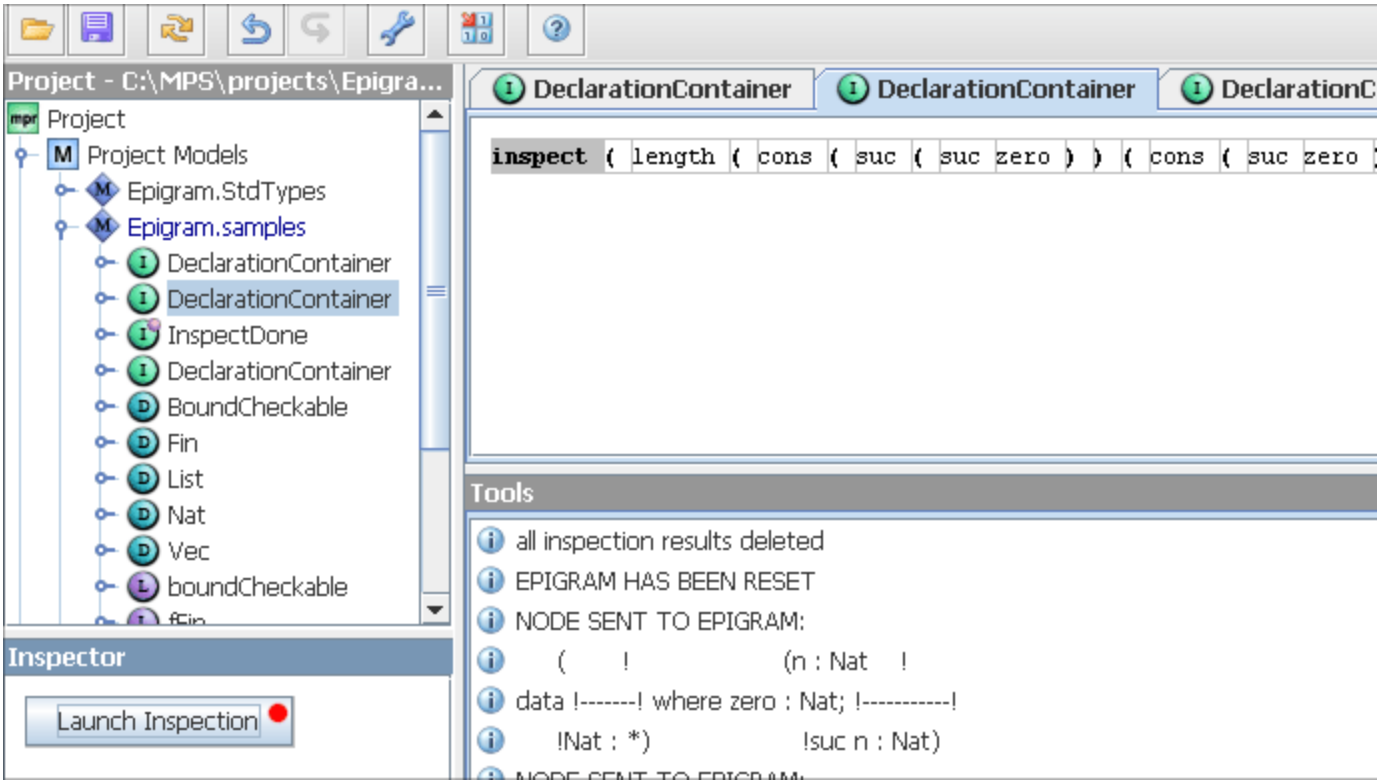
At this screenshot, you can see how I sent a program for adding two natural numbers to the elaborator. I pressed a button (marked here with a red spot) Try node in Epigram. First, MPgram had sent a declaration for type Nat, and only then sent a

declaration for plus itself.

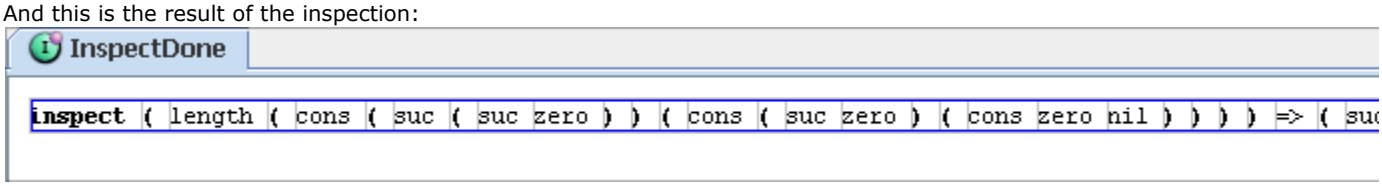



Currently, MPgram just tells you if your program is bad-typed or not. Also, when sending inspect declaration, MPgram returns you a result of inspection as a normal MPS Epigram structure node.

This screenshot shows how I sent this inspect declaration to calculate the length of my List of Nat.




And this is the result of the inspection:




The details of interacting with Epigram are discussed in the next paragraph.

## Interacting with Epigram

Since MPS stores programs as trees or graphs, the best way to interact with Epigram elaborator is just to send nodes of abstract syntax tree using special protocol. However, Durham Epigram does not provide such protocol. So I had to lose almost half of all MPS power and to decide rendering program into text before sending it to Epigram. Besides, I had to pretend being Xemacs and give Epigram Xemacs commands, and receive some. This pretending has made some difficulties, and this is the most unreliable part of my entire project.

If provided with interacting on abstract-syntax-tree level, I'll be able to use all the information about program collected when sending program to the elaborator.

Currently my editor is able to send programs to Durham Epigram, i.e. to distinguish well-typed programs from bad-typed ones by using Epigram; to receive inspection results and to create a MPS node from them, if result consists of identifiers and applications. Constructing a structure from abstractions is more difficult, because Epigram answers with abstractions which may contain implicitly typed variables, like in "lam x => x". Of course, may be it's not very difficult to infer those types, but I don't really think it's a job for editor. Besides, I consider receiving text a temporary workaround; and I hope that editor will finally receive a tree marked with nodes' types as a result – so I don't feel like developing things which are going to be unuseful soon.

# How to play with MPgram?

- First, download and install MPS from MPS Early Access Program page. Note: to play with MPgram, you do not need to create or change any languages, hence you do not have to use IntelliJ IDEA which is required, however, for some other use cases of MPS.
- Then, download and unzip an archive with Epigram language. It contains some Java classes and some MPS models. It also contains Epigram executable files for Windows and Linux which I had downloaded from Durham Epigram page.
- Launch MPS and open a file Epigram.mpr from where you've extracted an archive.
- Read MPgram tutorial.

- Enjoy!