★★★ UniFormal ★★★

Towards a Unified Framework

Lambert Meertens

IFIP WG2.1 Meeting 70, Schloss Reisensburg, July 2013

Proposal

- I believe that it is both technically feasible and worthwhile to define a unified open framework for representing collections of definitions and declarations of formal objects that is practically usable in a variety of areas in the formal sciences (formalized mathematics, declarative programming, program calculation).
- The proposal is that the Group commit itself to undertaking a multi-year effort to define a formalism providing such a unified framework.

Why make this a WG2.1 project?

- This is an ambitious project, but together we have the breadth and depth of expertise required to address it.
- We are not encumbered by private interests.
- After the initial release regular maintenance will be needed, both in response to demands for increased expressiveness and to fix errors and deal with other importunities; the Working Group can provide the necessary continuity and authority for this task.

Technical role of the Working Group

- To define a unified formalism (“UniFormal”) for the internal representation of terms that is usable for a wide variety of tools (proving/proof checking; computer algebra; program construction; …) and can be used as an interchange format between tools.
- To construct (or encourage the construction of) libraries for handling UniFormal terms (parsing/unparsing; type checking; transforming; …), thereby facilitating the construction of tools.

Terminology

- *Term*: a symbolically represented formal object (think “abstract syntax tree”).

```
plus(2,2)
```

- *Presentation*: what an end user sees (concrete syntax).

$$2 + 2$$

- *Tool*: any program using UniFormal for the internal representation of terms.
- *End user*: a person using such a tool.
- *Developer*: a person creating such a tool.
- *Context*: a collection of documents needed for the interpretation (semantics) of terms.

Initial scope limitations

It may be wise to impose some initial limitations on the scope of the formalism; for example:

- UniFormal has no preferred semantics; the core of the formalism is agnostic as to whether, say, `divide(1,0)` contains a domain error or denotes a well-defined specific value.
- UniFormal also has no unique prescribed type discipline; while it is certainly possible to express type judgements, interpreting them requires additional information.

Initial scope limitations (continued)

- UniFormal terms are not self-contained; tools may need additional context in order to be able to deal with them, and different tools may have different default interpretations. (For example, a computer algebra system may assume by default that variables are real-valued.)
- Depending on the tool and context, the same term may variously be presented as $f(x)$, $f.x$, or `f x`. UniFormal does not itself prescribe the presentations of terms.

Some design desiderata

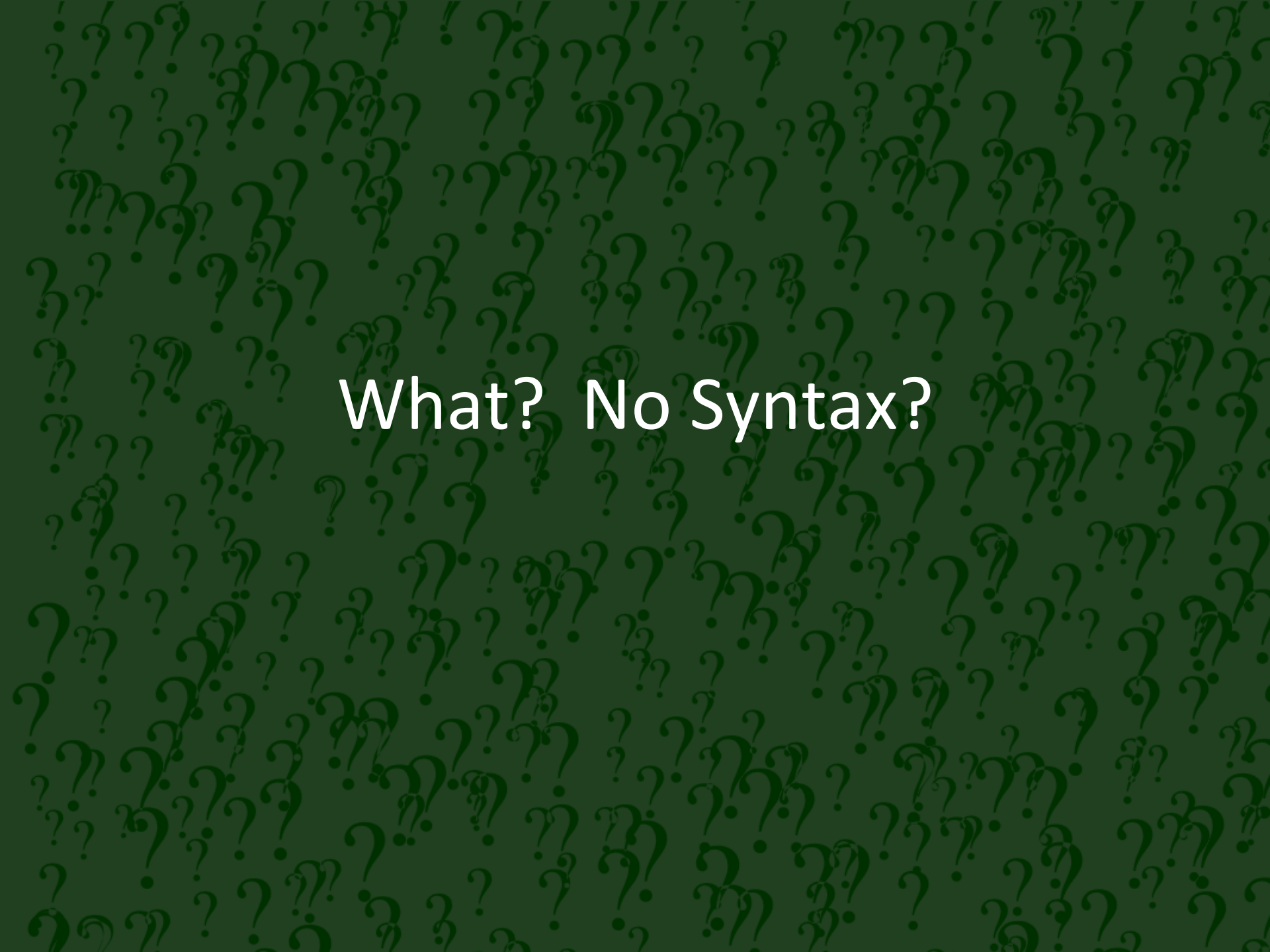
- The formalism is open-ended in the sense that developers are free to define restrictions or extensions as needed – provided that they are prepared to deal with ensuing compatibility issues.
- The formalism has a basic standardized core, to be supplemented by a forest of auxiliary extensions created in a need-driven process (e.g., a Unicode-based human-readable interchange-format standard).

Some design desiderata (continued)

- Simple things can be expressed in a simple way: no need for arcane preambles.
- Terms can carry annotations (e.g., for indicating presentation preferences).
- Links (URLs) are supported.
- There is a native (predefined) macro mechanism for allowing abbreviations (so that `plus(2,2)` might expand to `apply(plus-operator, tuple(2,literal($[2]),literal ($[2])))`).

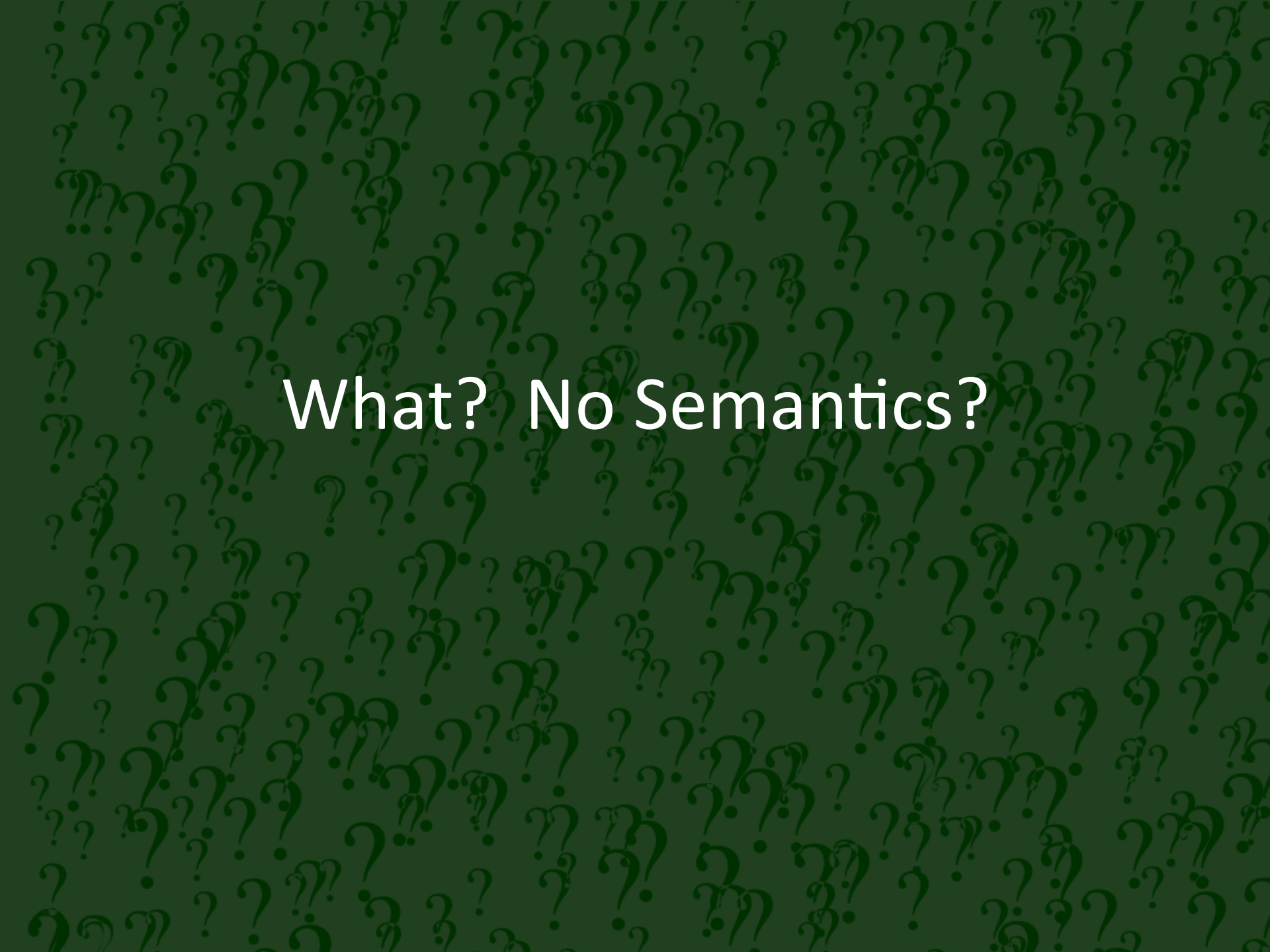
How to proceed with the discussion

- Before we have a discussion whether we choose to embark on such a project, I propose to first hold discussions to further reaching a shared understanding of some key issues; in particular:
  - to clarify to ourselves in more detail what we see as reasonable and valuable goals of such an undertaking;
  - to consider what mode of operation (*if* we undertake this) will help us best to achieve progress.


What? No Syntax?

No syntax?

- Each tool has its own concrete user-oriented presentation syntax. (For example, many systems use *term* : *type*, but Haskell uses *term* :: *type*.)
- When a term is imported into a tool, it will acquire the specific presentation syntax of that tool.
- Additionally, there will be a human-readable ASCII-style interchange format.


What? No Semantics?

No semantics?

There is already a (non-denotational) form of semantics:

- Given definitions (syntactic terms) of various symbols, unfolding plus standard substitution (β reduction) combined with α and η conversion gives us a basic form of semantics (interpreting equivalence as equality).
- This can be supplement with equational and predicate logic to take account of axioms for the domain of discourse.