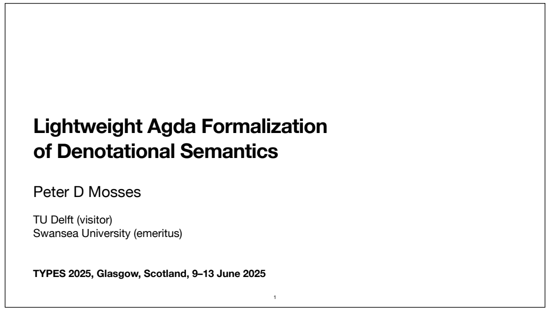
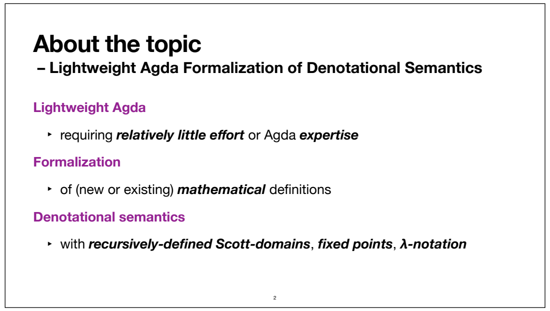


1

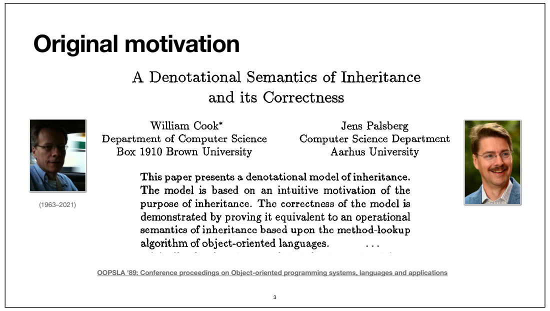
Thank you for still being here!

This is a lightweight talk about work in progress



2

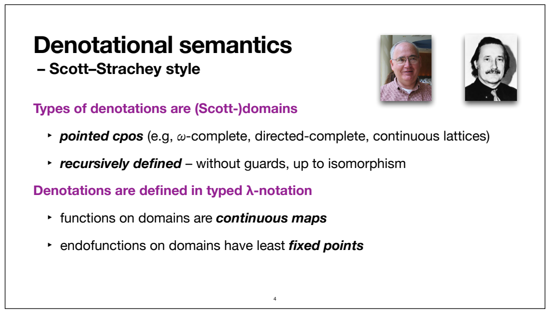
Let me start by clarifying the topic of the talk



3

This denotational semantics was published in 1989, and hadn't been mechanically checked.

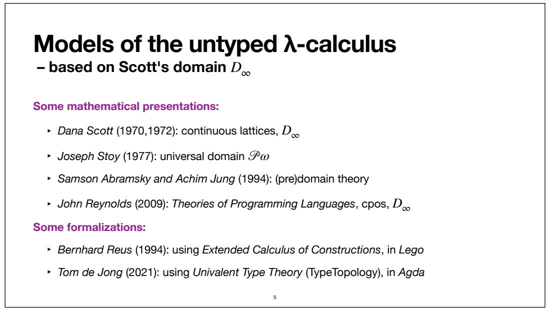
I expected it to be quite straightforward to formulate the definitions and proofs in Agda...



4

Denotational semantics isn't as popular these days as it was in the 70s and 80s

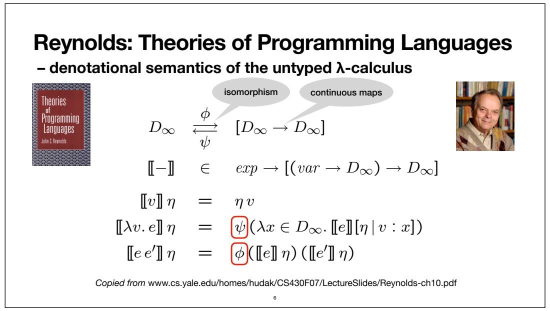
Let me briefly recall the main features



5

Dana Scott initially tried to prove that the untyped lambda-calculus has no set-theoretic models
but then he discovered the bilimit construction of the D-infinity model…

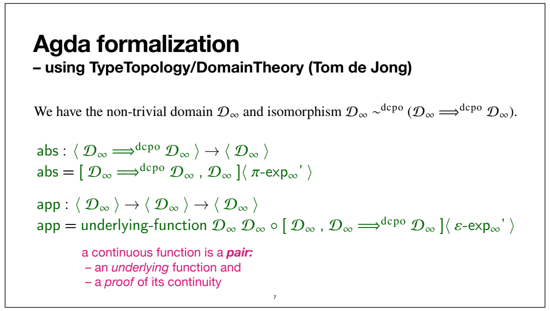
Apart from mathematical presentations of the model, some formalizations have been developed



6

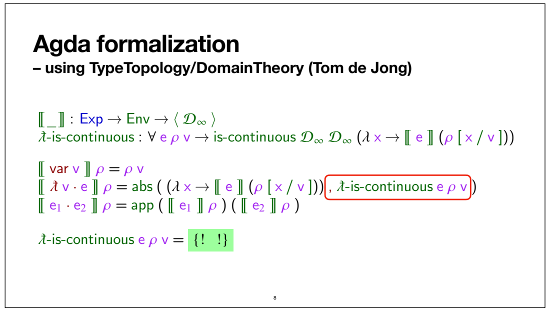
Here is a mathematical presentation of a denotational semantics of the untyped lambda-calculus from 2009

In denotational semantics of larger programming languages, isomorphisms are usually left implicit



7

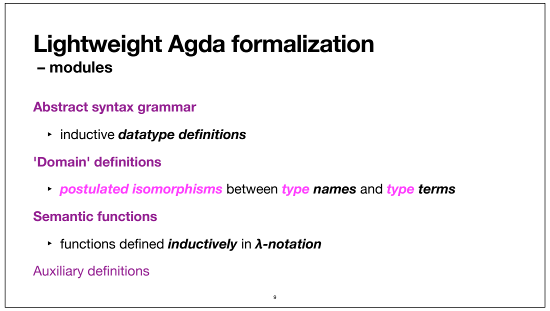
Tom de Jong's formalization is based on directed-complete posets.



8

The proof of the proposition that lambda is continuous isn't very deep, but takes 3 pages in John Reynolds book

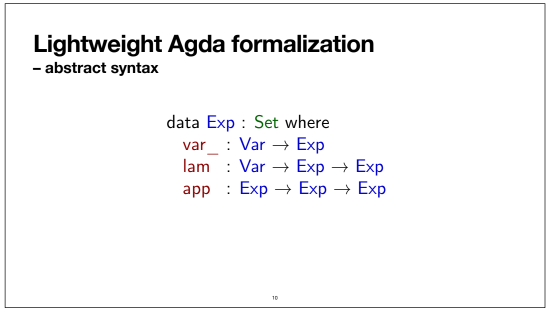
I would personally find it an excessive amount of work to formalize the proof in Agda...



9

... so I've developed a lightweight approach

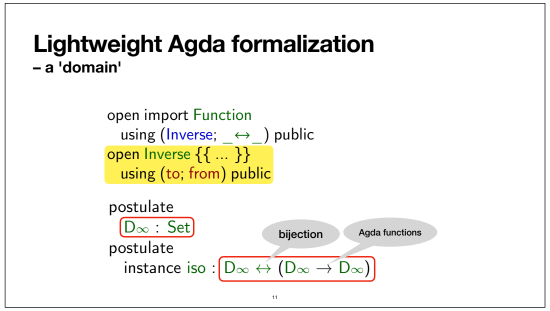
The next few slides illustrate the approach. The Agda definitions are included in the abstract, and available online



10

In Scott–Strachey style, abstract syntax is defined by a context-free grammar. Its formalization in Agda is a corresponding inductive datatype.

For simplicity, this datatype uses ordinary functional notation for the constructors.

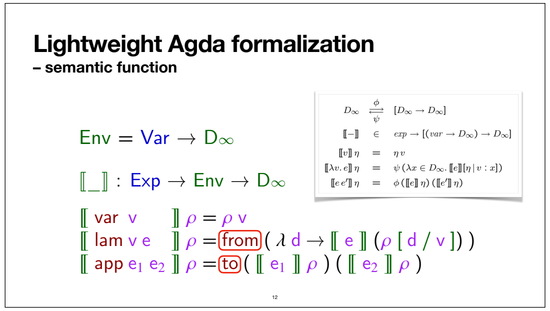


11

Here we assume D-infinity to be an ordinary Agda type with a bijection to the type of all Agda functions on D-infinity

In this example, the assumptions are satisfied when D-infinity has a single element.
In all other examples, the corresponding assumptions are unsatisfiable.

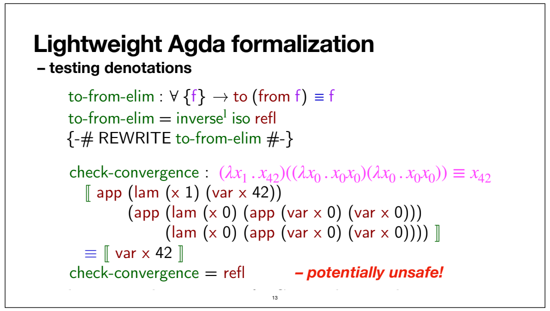
The highlighted Agda magic declares the inverse functions of the bijection, which are all we need...



12

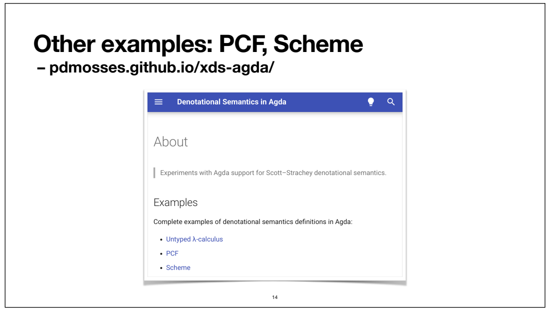
The semantic function corresponds directly to that defined by Reynolds!

The Agda type-checker insists on making the bijection explicit

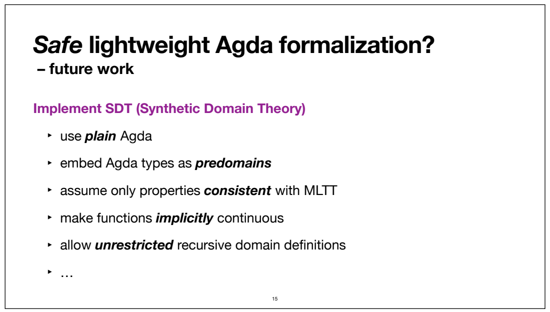


13

Agda can also check whether terms have equivalent denotations



14



15