Guarded Recursion in Agda via Sized Types

Niccolò Veltri[1] and Niels van der Weide[2]

[1] Department of Computer Science, IT University of Copenhagen, Denmark
`nive@itu.dk`

[2] Institute for Computation and Information Sciences, Radboud University, Nijmegen, The Netherlands
`nweide@cs.ru.nl`

Sized types and guarded recursion are two techniques used to ensure the productivity of recursively defined elements of coinductive types. Productivity means that each finite part of the output only depends on a finite part of the input, and it is necessary to ensure consistency of the type system. Programming with coinductive types becomes less convoluted when employing these techniques instead of traditional approaches relying on strict syntactic checks. In this work, we show how guarded recursion can be simulated in Agda using sized types.

Sized types [5] are types annotated with an abstract ordinal indicating the number of possible unfoldings that can be performed on elements of the type. Sized types are implemented in Agda and can be used in combination with coinductive records [1] to specify coinductive types.

Guarded recursion [7] is a different approach where the type system is enhanced with a modality, called "later" and written $\triangleright$, encoding time delay in types. The later modality comes with a general fixpoint combinator for programming with productive recursive functions and allows the specification of guarded recursive types. These types can be used in combination with clock quantification to define coinductive types [2].

To simulate guarded recursion via sized types, we formalize the syntax of a simple type theory for guarded recursion in Agda, which we call **GTT**. Then we prove the syntax sound w.r.t. a categorical semantics specified in terms of sized types[1].

The Syntax of GTT

The language **GTT** is a variant of Atkey and McBride's type system for productive coprogramming [2]. In Atkey and McBrides calculus, all judgments are indexed by a clock context, which may contain several different clocks. They extend simply typed lambda calculus with two additional type formers: a modality $\triangleright$ for encoding time delay into types and universal quantification over clock variables $\forall$, which is used in combination with $\triangleright$ to specify coinductive types.

The judgements of **GTT** depend on a clock context which can only be empty or contain a single clock. The types of **GTT** include the $\triangleright$ modality and a modality $\Box$, which is a nameless analogue of Atkey and McBride's universal clock quantification. The $\Box$ type former maps a type in the singleton clock context to one in the empty clock context. Guarded recursive types are defined using a least fixpoint type former μ. We also have a weakening operation on types, which maps a type in the empty clock context to one in the singleton clock context, and a similar operations on contexts. Clouston *et al.*[4] also studied a guarded variant of lambda calculus extended with a $\Box$ operation, which they call "constant". **GTT** differs from their calculus in that our judgments are indexed by a clock context and it has the benefit of allowing a more appealing introduction rule for the $\Box$ modality.

We only allow clock contexts in **GTT** to contain at most one clock variable, because Agda's support for sized types is tailored to types depending on exactly one size, or on a finite but

[1]The full Agda formalization can be found at https://github.com/niccoloveltri/agda-gtt.

precise number of sizes, which makes it cumbersome to work with types depending on clock contexts containing an indefinite number of clocks.

The terms of **GTT** include operations box and unbox, which are the introduction and elimination rules for the $\Box$ modality. In Atkey and McBride's system, these rules correspond to clock quantification and clock application respectively. In the rule for clock quantification, they add a side condition requiring the universally quantified clock to not appear free in the variable context. In **GTT**, we achieve this by requiring box to be applicable only to terms over a weakened context.

GTT also has a delayed fixpoint combinator dfix. This term takes as input a productive recursive definition, represented by a function of type $\triangleright A \to A$, and returns an element of $\triangleright A$. Using dfix we can define the usual fixpoint operator, returning a term of type A instead of $\triangleright A$.

Categorical Semantics of GTT

For the denotational semantics, we use a variation of the topos of trees [3]. Instead of natural numbers, we take the preorder of sizes as the indexing category of the presheaves. Types and contexts of **GTT** in the empty clock contexts are interpreted as sets, while types and contexts in the singleton clock contexts are interpreted as antitone sized types. The simple type and term formers are interpreted using the standard Kripke semantics.

Following Møgelberg's interpretation of universal clock quantification [6], we model the $\Box$ modality by taking limits. For the semantic later modality, we adapt the definition in the topos of trees to our setting. Given a presheaf A, the action of $\blacktriangleright A$ on a size i is given by taking the limit of A on all sizes strictly smaller than i. The semantic fixpoint operator is defined using self-application. The productivity of this construction relies on sizes being a well-ordered set. This gives rise to a consistent interpretation of **GTT**.

References

[1] Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: Programming Infinite Structures by Observations. In *POPL*, pages 27–38, 2013.

[2] Robert Atkey and Conor McBride. Productive Coprogramming with Guarded Recursion. In *ICFP*, pages 197–208, 2013.

[3] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees. *Logical Methods in Computer Science*, 8(4), 2012.

[4] Ranald Clouston, Ales Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. Programming and Reasoning with Guarded Recursion for Coinductive Types. In *FoSSaCS*, pages 407–421, 2015.

[5] John Hughes, Lars Pareto, and Amr Sabry. Proving the Correctness of Reactive Systems Using Sized Types. In *POPL*, pages 410–423, 1996.

[6] Rasmus Ejlers Møgelberg. A Type Theory for Productive Coprogramming via Guarded Recursion. In *CSL-LICS*, pages 71:1–71:10, 2014.

[7] Hiroshi Nakano. A Modality for Recursion. In *LICS*, pages 255–266, 2000.