The Clocks Are Ticking: No More Delays!
Reduction Semantics for Type Theory with Guarded Recursion*

Patrick Bahr[1], Hans Bugge Grathwohl[2], and Rasmus Ejlers Møgelberg[1]

[1] IT University of Copenhagen
[2] Aarhus University

Abstract

Guarded recursion in the sense of Nakano allows general recursive types and terms to be added to type theory without breaking consistency. Recent work has demonstrated applications of guarded recursion such as programming with codata, reasoning about coinductive types, as well as constructing and reasoning about denotational models of general recursive types. As a step towards an implementation of a type theory with guarded recursion, we present Clocked Type Theory, a new type theory for guarded recursion that is more suitable for reduction semantics than the existing ones. We prove confluence, strong normalisation and canonicity for its reduction semantics, constructing the theoretical basis for a future implementation.

1 Introduction

Guarded recursion [4] allows recursion to be added to type theory without breaking consistency by introducing time steps in the form of a delay type modality $\triangleright$ (pronounced 'later'). Elements of type $\triangleright A$ are to be thought of as elements of type A only available one time step from now. Recursion arises from a fixed point operator mapping each productive endofunction (i.e. a function of type $\triangleright A \to A$) to its unique fixed point.

The most advanced type theory with guarded recursion to date is Guarded Dependent Type Theory (GDTT) [2], an extensional type theory with a notion of clocks each of which has a delay modality. Coinductive types can be encoded using guarded recursive types and universal quantification over clocks [1], which allows productivity to be expressed in types. In addition, GDTT has a notion of *delayed substitutions* allowing for coinductive, type theoretic reasoning about coinductive data and functions manipulating coinductive data. Delayed substitutions make it difficult to define a reduction semantics directly on GDTT. To solve this problem, we introduce a new type theory called Clocked Type Theory (CloTT), which can be seen as a refinement of GDTT.

2 Clocked Type Theory

Clocked Type Theory is an extension of dependent type theory with a special collection of sorts called *clocks*, which are inhabited by *ticks*. An assumption of the form $\alpha : \kappa$ states that α is assumed to be a tick on clock κ. In a context $\Gamma, \alpha : \kappa, \Gamma' \vdash$, tick variable α represents the assumption that a tick of time occurs on clock κ between the time when the values represented by the variables in Γ and those in Γ' are received. The delay modality $\triangleright$ is replaced by a form of dependent function type over clocks: The type $\triangleright(\alpha : \kappa).A$ is a type of suspended computations

*This research was supported by The Danish Council for Independent Research for the Natural Sciences (FNU) (grant no. 4002-00442) and research grant 13156 from Villum Fonden.

requiring a tick α on the clock κ to compute elements of type A. This can be understood as a dependent function type, with introduction and elimination rules given by abstraction over ticks, written $\lambda(\alpha : \kappa).t$, and application to ticks, written $t\,[\alpha]$. A term t can only be applied to a tick α' if all of the variables that t depend on are available before α', which corresponds to the intuition that t computes to a value of type $\triangleright(\alpha : \kappa).A$ before α'. We obtain the ordinary delay type modality by writing $\triangleright^\kappa A$ for $\triangleright(\alpha : \kappa).A$ if α is not free in A. The applicative functor laws for $\triangleright^\kappa$ follow from standard β and η rules for tick abstraction and application:

$$(\lambda(\alpha' : \kappa).t)\,[\alpha] \to t\,[\alpha/\alpha'] \qquad \lambda(\alpha : \kappa).(t\,[\alpha]) \to t \quad \text{if } \alpha \notin \mathsf{fv}(t)$$

A suspended computation represented by a closed term of type $\triangleright(\alpha : \kappa).A$ can be forced by applying it to the tick constant $\diamond$. In general, this is unsafe for open terms, since it breaks the productivity guarantees that the typing system should provide. For example, the term $\lambda(x : \triangleright^\kappa A).x\,[\diamond]$ should not be typeable, because it is not productive. It is safe, however, to force an open term if the clock κ does not appear free in the context Γ.

A term $f : \triangleright^\kappa A \to A$ is a productive function taking suspended computations of type A and returning values of type A. The *delayed fixed point* $\mathsf{dfix}^\kappa f$ of f is an element of type $\triangleright^\kappa A$ which, when given a tick, applies f to itself. Crucially, $\mathsf{dfix}^\kappa f$ only unfolds in the reduction semantics if applied to the tick constant, i.e. we have the reduction rule $(\mathsf{dfix}^\kappa\, t)\,[\diamond] \to t\,(\mathsf{dfix}^\kappa\, t)$. In particular, any term $(\mathsf{dfix}^\kappa\, t)\,[\alpha]$ remains stuck if α is a tick variable.

3 Results

We argue that CloTT is at least as expressive as the fragment of GDTT without identity types, by giving a translation of the latter into the former. The translation maps most of the equational rules of GDTT to equalities that follow from the reduction semantics of CloTT. In particular, most of the rules for delayed substitutions follow in fact from the β and η rules for tick abstraction and standard rules for substitutions. Some of the equational rules of GDTT do not follow from the reduction semantics, but we argue that these are most naturally expressed in CloTT as propositional equalities stating that 'all ticks are equal' and 'all clocks are equal'. We argue informally why these can be added to a future extension of CloTT with path types (in the sense of Cubical Type Theory [3]) without breaking canonicity.

Our main results concern the reduction semantics of CloTT, which we show is confluent and strongly normalising. As a consequence of this, equality of terms and types can be decided by reducing these to their unique normal forms. This decision procedure is a major step towards a type checking algorithm for CloTT. We also prove a canonicity theorem stating that every closed term of type Nat reduces to a natural number. As a consequence of this we derive the statement of productivity: Given a well typed closed term of stream type, its n'th element can be computed in finite time. This is a formal statement of the fact that guarded recursion captures productivity of coinductive definitions in types.

References

[1] R. Atkey and C. McBride. Productive coprogramming with guarded recursion. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, pages 197–208. ACM, 2013.

[2] A. Bizjak, H. B. Grathwohl, R. Clouston, R. E. Møgelberg, and L. Birkedal. Guarded dependent type theory with coinductive types. In *FOSSACS*, 2016.

[3] C. Cohen, T. Coquand, S. Huber, and A. Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. *CoRR*, abs/1611.02108, 2016.

[4] Hiroshi Nakano. A modality for recursion. In *LICS*, pages 255–266, 2000.