The Clocks Are Ticking: No More Delays!

Reduction Semantics for Type Theory with Guarded Recursion

Patrick Bahr[1] Hans Bugge Grathwohl[2]
Rasmus Møgelberg[1]

[1]IT University of Copenhagen

[2]Aarhus University

What is guarded recursion?

- abstract form of step-indexing
- allows to add general recursive types without breaking consistency

What is it good for?

- For reasoning: construct models of programming languages and type systems.
- For programming: ensures productivity of coinductive definitions – in a modular way.

Goals

Reduction semantics

for dependent type theory with

- a universe
- guarded recursion
- multiple clocks & clock quantification

Motivation

- decide equality (confluence + normalisation)
  $\rightsquigarrow$ type checking
- establish productivity operationally (canonicity)

Overview

1. Guarded Recursion
2. Guarded Dependent Type Theory
3. Clocked Type Theory (CloTT)
   + Reduction Semantics

Guarded Recursive Types

Nakano. A modality for recursion. LICS 2000

Guarded Recursion

- type modality $\triangleright$ (pronounced "later")
- $\triangleright$ is an applicative functor[1]

$$\text{next} : A \to \triangleright A$$

$$\circledast : \triangleright(A \to B) \to \triangleright A \to \triangleright B$$

- guarded fixed-point operator

$\text{fix} : (\triangleright A \to A) \to A$

$$\text{fix}\, f = f(\text{next}(\text{fix}\, f))$$

[1] Atkey & McBride. Productive Coprogramming with Guarded Recursion, ICFP 2013

Guarded Recursive Types

Guarded streams:

$$\mathsf{Str}_{\mathsf{G}} \cong \mathsf{Nat} \times \triangleright \mathsf{Str}_{\mathsf{G}}$$

functions of types $\mathsf{Str}_{\mathsf{G}} \to \mathsf{Str}_{\mathsf{G}}$ are causal.

Guarded Recursive Types

Guarded streams:

$$\mathsf{Str}_\mathsf{G} \cong \mathsf{Nat} \times \triangleright \mathsf{Str}_\mathsf{G}$$

functions of types $\mathsf{Str}_\mathsf{G} \to \mathsf{Str}_\mathsf{G}$ are causal.

Example

We can write a function that increments each element:

$$\mathsf{incr} : \mathsf{Str}_\mathsf{G} \to \mathsf{Str}_\mathsf{G}$$
$$\mathsf{incr} := \mathsf{fix}\, \lambda g.\lambda x : \mathsf{Str}_\mathsf{G}.\, \langle \mathsf{suc}\,(\pi_1 x), g \circledast (\pi_2\, x) \rangle$$

but not a function that skips every other element

$$\mathsf{skipEven} : \mathsf{Str}_\mathsf{G} \to \mathsf{Str}_\mathsf{G}$$

Coinductive types via clock quantification

- $\triangleright$ annotated with clock variables κ
- quantification over clocks: $\forall \kappa.A$
- force : $(\forall \kappa.\triangleright^{\kappa} A) \to \forall \kappa.A$

Example

$$\mathsf{Str}_{\mathsf{G}} \cong \mathsf{Nat} \times \triangleright\ \mathsf{Str}_{\mathsf{G}}$$

Atkey & McBride. Productive Coprogramming with Guarded Recursion, ICFP 2013

Coinductive types via clock quantification

- $\triangleright$ annotated with clock variables κ
- quantification over clocks: $\forall\kappa.A$
- force : $(\forall\kappa.\triangleright^{\kappa}A) \to \forall\kappa.A$

Example

$$\mathsf{Str}_G \cong \mathsf{Nat} \times \triangleright^{\kappa}\mathsf{Str}_G$$

Atkey & McBride. Productive Coprogramming with Guarded Recursion, ICFP 2013

Coinductive types via clock quantification

- $\triangleright$ annotated with clock variables κ
- quantification over clocks: $\forall \kappa.A$
- force : $(\forall \kappa.\triangleright^{\kappa} A) \to \forall \kappa.A$

Example

$$\mathsf{Str}_{\mathsf{G}}^{\kappa} \cong \mathsf{Nat} \times \triangleright^{\kappa} \mathsf{Str}_{\mathsf{G}}^{\kappa}$$

Atkey & McBride. Productive Coprogramming with Guarded Recursion, ICFP 2013

Coinductive types via clock quantification

- $\triangleright$ annotated with clock variables κ
- quantification over clocks: $\forall\kappa.A$
- force : $(\forall\kappa.\triangleright^{\kappa}A) \to \forall\kappa.A$

Example

$$\mathsf{Str}_{\mathsf{G}}^{\kappa} \cong \mathsf{Nat} \times \triangleright^{\kappa}\mathsf{Str}_{\mathsf{G}}^{\kappa}$$
$$\mathsf{Str} = \forall\kappa.\mathsf{Str}_{\mathsf{G}}^{\kappa}$$

Atkey & McBride. Productive Coprogramming with Guarded Recursion, ICFP 2013

Coinductive types via clock quantification

- $\triangleright$ annotated with clock variables κ
- quantification over clocks: $\forall\kappa.A$
- force : $(\forall\kappa.\triangleright^{\kappa}A) \to \forall\kappa.A$

Example

$$\mathsf{Str}_{\mathsf{G}}^{\kappa} \cong \mathsf{Nat} \times \triangleright^{\kappa}\mathsf{Str}_{\mathsf{G}}^{\kappa}$$
$$\mathsf{Str} = \forall\kappa.\mathsf{Str}_{\mathsf{G}}^{\kappa}$$

Functions of type $\mathsf{Str} \to \mathsf{Str}$ are productive.

e.g. $\quad \mathsf{skipEven} : \mathsf{Str}_{\mathsf{G}} \to \mathsf{Str}_{\mathsf{G}}$

Atkey & McBride. Productive Coprogramming with Guarded Recursion, ICFP 2013

Guarded Recursion
+
Dependent Type Theory

A. Bizjak, H. B. Grathwohl, R. Clouston, R. E. Møgelberg, and L. Birkedal. Guarded dependent type theory with coinductive types. FoSSaCS 2016.

Guarded Recursion

+

Dependent Type Theory

Guarded Dependent Type Theory (GDTT)

A. Bizjak, H. B. Grathwohl, R. Clouston, R. E. Møgelberg, and L. Birkedal. Guarded dependent type theory with coinductive types. FoSSaCS 2016.

Combining Π and $\triangleright^{\kappa}$

$$\frac{\Gamma \vdash s : \Pi x : A.B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B\,[t/x]}$$

Combining Π and $\triangleright^\kappa$

$$\frac{\Gamma \vdash s : \Pi x : A.B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B\,[t/x]}$$

$$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : ???}$$

Combining Π and $\triangleright^\kappa$

$$\frac{\Gamma \vdash s : \Pi x : A.B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B\,[t/x]}$$

$$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : \triangleright^\kappa B\,[t/x]}$$

Combining Π and $\triangleright^\kappa$

$$\frac{\Gamma \vdash s : \Pi x : A.B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B\,[t/x]}$$

$$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : \triangleright^\kappa B\,[t/x]}$$

- Problem: $t : \triangleright^\kappa A$, but $x : A$

Combining Π and $\triangleright^\kappa$

$$\frac{\Gamma \vdash s : \Pi x : A.B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B\,[t/x]}$$

$$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : \triangleright^\kappa B\,[t/x]}$$

- Problem: $t : \triangleright^\kappa A$, but $x : A$
- needed: getting rid of $\triangleright^\kappa$ in a controlled way

Delayed Substitutions

[Bizjak et al. FoSSaCS 2016]

Instead of $$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : \triangleright^\kappa B\,[t/x]}$$

GDTT has $$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : \triangleright^\kappa \quad [x \leftarrow t]\,.B}$$

Delayed Substitutions

[Bizjak et al. FoSSaCS 2016]

Instead of
$$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : \triangleright^\kappa B\,[t/x]}$$

GDTT has
$$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : \triangleright^\kappa \underbrace{[x \leftarrow t]\,.B}_{\text{"let next}^\kappa x = t \text{ in } B\text{"}}}$$

Delayed Substitutions

[Bizjak et al. FoSSaCS 2016]

Instead of $$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : \triangleright^\kappa B\,[t/x]}$$

GDTT has $$\frac{\Gamma \vdash s : \triangleright^\kappa(\Pi x : A.B) \qquad \Gamma \vdash t : \triangleright^\kappa A}{\Gamma \vdash s \circledast^\kappa t : \triangleright^\kappa \underbrace{[x \leftarrow t]\,.B}_{\text{"let next}^\kappa x = t \text{ in } B\text{"}}}$$

In general

$$\triangleright^\kappa\,[x_1 \leftarrow t_1, \ldots x_n \leftarrow t_n]\,.A$$

$$\text{next}\,[x_1 \leftarrow t_1, \ldots x_n \leftarrow t_n]\,.t$$

Equalities

$$
\begin{aligned}
\triangleright^{\kappa}\xi\,[x \leftarrow \mathsf{next}\xi.u]\,.A &= \triangleright^{\kappa}\xi.A\,[u/x] \\
\triangleright^{\kappa}\xi\,[x \leftarrow u]\,.A &= \triangleright^{\kappa}\xi.A && \text{if } x \notin \mathsf{fv}(A) \\
\triangleright^{\kappa}\xi\,[x \leftarrow u, y \leftarrow v]\,\xi'.A &= \triangleright^{\kappa}\xi\,[y \leftarrow v, x \leftarrow u]\,\xi'.A && \text{if } \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{next}\xi\,[x \leftarrow \mathsf{next}\xi.u]\,.t &= \mathsf{next}\xi.t\,[u/x] \\
\mathsf{next}\xi\,[x \leftarrow u]\,.t &= \mathsf{next}\xi.t && \text{if } x \notin \mathsf{fv}(t) \\
\mathsf{next}\xi\,[x \leftarrow u, y \leftarrow v]\,\xi'.t &= \mathsf{next}\xi\,[y \leftarrow v, x \leftarrow u]\,\xi'.t && \text{if } \ldots \\
\mathsf{next}\xi\,[x \leftarrow t]\,.x &= t
\end{aligned}
$$

Equalities

$$
\begin{aligned}
\triangleright^{\kappa}\xi\,[x \leftarrow \mathsf{next}\xi.u]\,.A &= \triangleright^{\kappa}\xi.A\,[u/x] \\
\triangleright^{\kappa}\xi\,[x \leftarrow u]\,.A &= \triangleright^{\kappa}\xi.A && \text{if } x \notin \mathsf{fv}(A) \\
\triangleright^{\kappa}\xi\,[x \leftarrow u, y \leftarrow v]\,\xi'.A &= \triangleright^{\kappa}\xi\,[y \leftarrow v, x \leftarrow u]\,\xi'.A && \text{if } \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{next}\xi\,[x \leftarrow \mathsf{next}\xi.u]\,.t &= \mathsf{next}\xi.t\,[u/x] \\
\mathsf{next}\xi\,[x \leftarrow u]\,.t &= \mathsf{next}\xi.t && \text{if } x \notin \mathsf{fv}(t) \\
\mathsf{next}\xi\,[x \leftarrow u, y \leftarrow v]\,\xi'.t &= \mathsf{next}\xi\,[y \leftarrow v, x \leftarrow u]\,\xi'.t && \text{if } \ldots \\
\mathsf{next}\xi\,[x \leftarrow t]\,.x &= t
\end{aligned}
$$

Not clear how to devise a confluent & normalising reduction semantics that verify these equalities.

Clocked Type Theory (CloTT)

"The clocks are ticking: No more delays!"

The clocks are ticking

- Treat $\triangleright^\kappa A$ as function type "$\kappa \to A$"
- generalise to dependent function type: $\triangleright(\alpha : \kappa).A$

The clocks are ticking

- Treat $\triangleright^{\kappa} A$ as function type "$\kappa \to A$"
- generalise to dependent function type: $\triangleright(\alpha : \kappa).A$

$$\frac{\Gamma, \alpha : \kappa \vdash_{\Delta} t : A}{\Gamma \vdash_{\Delta} \lambda(\alpha : \kappa).t : \triangleright(\alpha : \kappa).A}$$

The clocks are ticking

- Treat $\triangleright^{\kappa} A$ as function type "$\kappa \to A$"
- generalise to dependent function type: $\triangleright\,(\alpha : \kappa).A$

$$\frac{\Gamma, \alpha : \kappa \vdash_{\Delta} t : A}{\Gamma \vdash_{\Delta} \lambda(\alpha : \kappa).t : \triangleright\,(\alpha : \kappa).A}$$

$$\frac{\Gamma \vdash_{\Delta} t : \triangleright\,(\alpha : \kappa).A \qquad \Gamma, \alpha' : \kappa, \Gamma' \vdash_{\Delta}}{\Gamma\ , \alpha' : \kappa,\ \Gamma'\ \vdash_{\Delta} t\,[\alpha'] : A\,[\alpha'/\alpha]}$$

The clocks are ticking

- Treat $\triangleright^{\kappa} A$ as function type "$\kappa \to A$"
- generalise to dependent function type: $\triangleright(\alpha : \kappa).A$

$$\frac{\Gamma, \alpha : \kappa \vdash_{\Delta} t : A}{\Gamma \vdash_{\Delta} \lambda(\alpha : \kappa).t : \triangleright(\alpha : \kappa).A}$$

$$\frac{\Gamma \vdash_{\Delta} t : \triangleright(\alpha : \kappa).A \qquad \Gamma, \alpha' : \kappa, \Gamma' \vdash_{\Delta}}{\Gamma, \alpha' : \kappa, \Gamma' \vdash_{\Delta} t\,[\alpha'] : A\,[\alpha'/\alpha]}$$

available before
tick α' on
clock κ occurred

The clocks are ticking

- Treat $\triangleright^{\kappa} A$ as function type "$\kappa \to A$"
- generalise to dependent function type: $\triangleright(\alpha : \kappa).A$

$$\frac{\Gamma, \alpha : \kappa \vdash_{\Delta} t : A}{\Gamma \vdash_{\Delta} \lambda(\alpha : \kappa).t : \triangleright(\alpha : \kappa).A}$$

$$\frac{\Gamma \vdash_{\Delta} t : \triangleright(\alpha : \kappa).A \qquad \Gamma, \alpha' : \kappa, \Gamma' \vdash_{\Delta}}{\Gamma, \alpha' : \kappa, \Gamma' \vdash_{\Delta} t\,[\alpha'] : A\,[\alpha'/\alpha]}$$

available before tick α' on clock κ occurred

available after tick α' on clock κ occurred

The clocks are ticking

- Treat $\triangleright^{\kappa} A$ as function type "$\kappa \to A$"
- generalise to dependent function type: $\triangleright\,(\alpha : \kappa).A$

$$\frac{\Gamma, \alpha : \kappa \vdash_{\Delta} t : A}{\Gamma \vdash_{\Delta} \lambda(\alpha : \kappa).t : \triangleright\,(\alpha : \kappa).A}$$

$$\frac{\Gamma \vdash_{\Delta} t : \triangleright\,(\alpha : \kappa).A \qquad \Gamma, \alpha' : \kappa, \Gamma' \vdash_{\Delta}}{\Gamma\ , \alpha' : \kappa,\ \Gamma'\ \vdash_{\Delta} t\,[\alpha'] : A\,[\alpha'/\alpha]}$$

The clocks are ticking

- Treat $\triangleright^\kappa A$ as function type "$\kappa \to A$"
- generalise to dependent function type: $\triangleright(\alpha : \kappa).A$

$$\frac{\Gamma, \alpha : \kappa \vdash_\Delta t : A}{\Gamma \vdash_\Delta \lambda(\alpha : \kappa).t : \triangleright(\alpha : \kappa).A}$$

$$\frac{\Gamma \vdash_\Delta t : \triangleright(\alpha : \kappa).A \qquad \Gamma, \alpha' : \kappa, \Gamma' \vdash_\Delta}{\Gamma\ , \alpha' : \kappa,\ \Gamma'\ \vdash_\Delta t\,[\alpha'] : A\,[\alpha'/\alpha]}$$

No more delays!

$$\begin{aligned} \mathsf{next}^\kappa\,[x \leftarrow t]\,.s &\rightsquigarrow \lambda(\alpha : \kappa).s\,[t\,[\alpha]/x] \\ \triangleright^\kappa\,[x \leftarrow t]\,.s &\rightsquigarrow \triangleright(\alpha : \kappa).s\,[t\,[\alpha]/x] \end{aligned}$$

Reduction Semantics of Ticks

$$(\lambda(\alpha' : \kappa).t)\,[\alpha] \quad \rightarrow \quad t\,[\alpha/\alpha']$$

$$\lambda(\alpha : \kappa).(t\,[\alpha]) \quad \rightarrow \quad t \qquad \text{if } \alpha \notin \mathsf{fv}(t)$$

Guarded fixed points

Fixed point combinator

$\mathsf{fix}^\kappa : (\triangleright^\kappa A \to A) \to A$

$$\mathsf{fix}^\kappa f = f(\mathsf{next}^\kappa(\mathsf{fix}^\kappa f))$$

We need to restrict fixed point unfolding to obtain strong normalisation (while retaining canonicity).

Guarded fixed points

Fixed point combinator

$\mathsf{fix}^\kappa : (\triangleright^\kappa A \to A) \to A$

$$\mathsf{fix}^\kappa f = f(\mathsf{next}^\kappa(\mathsf{fix}^\kappa f))$$

We need to restrict fixed point unfolding to obtain strong normalisation (while retaining canonicity).

Delayed fixed point

- $\mathsf{dfix}^\kappa : (\triangleright^\kappa A \to A) \to \triangleright^\kappa A$
- only unfolds if applied to tick constant $\diamond$

$$\begin{array}{lcll} (\mathsf{dfix}^\kappa f)\,[\alpha] & \not\to & f(\mathsf{dfix}^\kappa f) & \text{if } \alpha \text{ is tick variable} \\ (\mathsf{dfix}^\kappa f)\,[\diamond] & \to & f(\mathsf{dfix}^\kappa f) & \end{array}$$

Tick constant

$\diamond$ can only be used in a context without free occurrences of κ.

$$\frac{\Gamma \vdash_{\Delta,\kappa} t : \triangleright\,(\alpha : \kappa).A \qquad\qquad \Gamma \vdash_{\Delta}}{\Gamma \vdash_{\Delta,\kappa} t\,[\diamond] \qquad\qquad : A\,[\diamond/\alpha]}$$

$\diamond$ is used to implement force

$$\mathsf{force} : \forall\kappa.\triangleright^{\kappa} A \to \forall\kappa.A$$
$$\mathsf{force}\ x = \Lambda\kappa.(x\,[\kappa])\,[\diamond]$$

Tick constant

$\diamond$ can only be used in a context without free occurrences of κ.

$$\frac{\Gamma \vdash_{\Delta,\kappa} t : \triangleright\,(\alpha : \kappa).A \qquad \kappa' \in \Delta \qquad \Gamma \vdash_{\Delta}}{\Gamma \vdash_{\Delta} \quad t\,[\diamond]\,[\kappa'/\kappa] \;:\; A\,[\diamond/\alpha]\;[\kappa'/\kappa]}$$

$\diamond$ is used to implement force

$$\mathsf{force} : \forall\kappa.\triangleright^{\kappa} A \to \forall\kappa.A$$
$$\mathsf{force}\ x = \Lambda\kappa.(x\,[\kappa])\,[\diamond]$$

Results

Theorem (Decidable equality)

- *Reduction relation* $\rightarrow$ *is confluent.*
- *Well-typed terms are strongly normalising.*

Results

Theorem (Decidable equality)

- *Reduction relation* $\rightarrow$ *is confluent.*
- *Well-typed terms are strongly normalising.*

Theorem (Canonicity)

If $\vdash_\Delta t : \mathsf{Nat}$, *then* $t \rightarrow^* \mathsf{suc}^n\, 0$ *for some* $n \in \mathbb{N}$.

Results

Theorem (Decidable equality)

- *Reduction relation* $\to$ *is confluent.*
- *Well-typed terms are strongly normalising.*

Theorem (Canonicity)

If $\vdash_\Delta t : \mathsf{Nat}$, *then* $t \to^* \mathsf{suc}^n\, 0$ *for some* $n \in \mathbb{N}$.

Corollary (Productivity)

Given $\vdash_\Delta t : \mathsf{Str}$, *any element of the stream* t *can be computed with a finite number of reduction steps.*

Summary

Reduction semantics for dependent type theory with

- a universe
- guarded recursion
- multiple clocks & clock quantification

[2] Birkedal et al. Guarded cubical type theory: Path equality for guarded recursion. CSL 2016

Summary

Reduction semantics for dependent type theory with

- a universe
- guarded recursion
- multiple clocks & clock quantification

Future work

- identity types $\rightsquigarrow$ cubical type theory[2]
- add propositional equalities (fixed point unfolding, clock/tick irrelevance)

[2] Birkedal et al. Guarded cubical type theory: Path equality for guarded recursion. CSL 2016

The Clocks Are Ticking: No More Delays!

Reduction Semantics for Type Theory with Guarded Recursion

Patrick Bahr[1] Hans Bugge Grathwohl[2]
Rasmus Møgelberg[1]

[1]IT University of Copenhagen

[2]Aarhus University

Bonus Slides

Typing Rules

$$\frac{\Gamma \vdash_{\Delta,\kappa} t : A \qquad \Gamma \vdash_{\Delta}}{\Gamma \vdash_{\Delta} \Lambda\kappa.t : \forall\kappa.A} \qquad \frac{\Gamma \vdash_{\Delta} t : \forall\kappa.A \qquad \kappa' \in \Delta}{\Gamma \vdash_{\Delta} t[\kappa'] : A\,[\kappa'/\kappa]}$$

$$\frac{\Gamma, \alpha : \kappa \vdash_{\Delta} t : A \qquad \kappa \in \Delta}{\Gamma \vdash_{\Delta} \lambda(\alpha : \kappa).t : \triangleright\,(\alpha : \kappa).A}$$

$$\frac{\Gamma \vdash_{\Delta} t : \triangleright\,(\alpha : \kappa).A \qquad \Gamma, \alpha' : \kappa, \Gamma' \vdash_{\Delta}}{\Gamma, \alpha' : \kappa, \Gamma' \vdash_{\Delta} t\,[\alpha'] : A\,[\alpha'/\alpha]}$$

$$\frac{\Gamma \vdash_{\Delta,\kappa} t : \triangleright\,(\alpha : \kappa).A \qquad \Gamma \vdash_{\Delta} \qquad \kappa' \in \Delta}{\Gamma \vdash_{\Delta} (t\ [\kappa'/\kappa])\,[\diamond] : A\,[\kappa'/\kappa]\,[\diamond/\alpha]}$$

$$\frac{\Gamma \vdash_{\Delta} t : \triangleright^{\kappa} A \to A}{\Gamma \vdash_{\Delta} \mathsf{dfix}^{\kappa}\, t : \triangleright^{\kappa} A}$$

Typing Rules (cont.)

$$\frac{\Gamma, x : A \vdash_\Delta t : B}{\Gamma \vdash_\Delta \lambda(x : A).t : \Pi(x : A).\, B}$$

$$\frac{\Gamma \vdash_\Delta t : \Pi(x : A).\, B \qquad \Gamma \vdash_\Delta u : A}{\Gamma \vdash_\Delta t\, u : B\,[u/x]}$$

$$\frac{\Gamma \vdash_\Delta F\,(\mathsf{dfix}^\kappa\, F)\, u : \mathcal{U} \qquad \Gamma \vdash_\Delta t : \mathsf{El}\,((\mathsf{dfix}^\kappa\, F)\,[\alpha]\, u)}{\Gamma \vdash_\Delta \mathsf{unfold}_\alpha\, t : \mathsf{El}\,(F\,(\mathsf{dfix}^\kappa\, F)\, u)}$$

$$\frac{\Gamma \vdash_\Delta ((\mathsf{dfix}^\kappa\, F)\,[\alpha])\, u : \mathcal{U} \qquad \Gamma \vdash_\Delta t : \mathsf{El}\,(F\,(\mathsf{dfix}^\kappa\, F)\, u)}{\Gamma \vdash_\Delta \mathsf{fold}_\alpha\, t : \mathsf{El}\,((\mathsf{dfix}^\kappa\, F)\,[\alpha]\, u)}$$

Reduction Semantics

$$(\lambda x : A.t)s \to t\,[s/x] \qquad (\Lambda\kappa.t)[\kappa'] \to t\,[\kappa'/\kappa]$$

$$(\lambda(\alpha' : \kappa).t)\,[\alpha] \to t\,[\alpha/\alpha'] \qquad \lambda(\alpha : \kappa).(t\,[\alpha]) \to t$$

$$(\Lambda\kappa.t[\kappa]) \to t \qquad \pi_i\,\langle t_1, t_2\rangle \to t_i$$

$$\mathsf{fold}_\diamond\, t \to t \qquad \mathsf{unfold}_\diamond\, t \to t$$

$$\mathsf{if\ true}\ t_1\ t_2 \to t_1 \qquad \mathsf{if\ false}\ t_1\ t_2 \to t_2$$

$$\mathsf{rec}\,(\mathsf{suc}\ t_1)\ t_2\ t_3 \to t_3\ t_1\,(\mathsf{rec}\ t_1\ t_2\ t_3) \qquad \mathsf{rec}\,0\ t\ s \to t$$

$$(\mathsf{dfix}^\kappa\ t)\,[\diamond] \to t\,(\mathsf{dfix}^\kappa\ t)$$

$$\frac{t \to u}{C[t] \to C[u]}$$