Recursive types

Marco Kuhlmann

2003-03-05
Recursive types are ubiquitous

Lists of natural numbers:

\[
\text{NatList} = \text{nil} : \text{Unit} \mid \text{cons} : \text{Nat} \times \text{NatList} \\
\mu T. \text{Unit} + \text{Nat} \times T
\]

Dependency trees:

\[
\text{DTree} = \text{t} : \text{Lex} \mid \text{n} : \text{Lex} \times (\text{Role} \times \text{DTree}) \text{List} \\
\mu T. \text{Lex} + \text{Lex} \times (\text{Role} \times T) \text{List}
\]

Functional counters:

\[
\text{Counter} = \text{get} : \text{Nat} \mid \text{inc} : \text{Unit} \to \text{Counter} \\
\mu T. \text{Nat} + (\text{Unit} \to T)
\]
Recursive types as infinite trees

Recursive type definitions = specifications of infinite regular trees

Example:

NatList = nil : Unit | cons : Nat × NatList
Introduction

- Introducing recursive types
  - Intuition
  - Expressive power
  - Formalities
- Reasoning about infinite trees
- Membership tests
- Recursive types and subtyping
- Conclusions
Typing the fixed-point combinator

\[ \text{fix} = \lambda f. \ (\lambda x. f (x x)) \ (\lambda x. f (x x)) \]

How would we type the fixed-point combinator?

- \( x \) needs to have an arrow type whose domain is the type of \( x \) itself
- property is satisfied by the recursive type \( \mu S. S \rightarrow T \)

A well-typed fixed-point combinator

\[ \text{fix}_T = \lambda f : T \rightarrow T. \ (\lambda x : (\mu S. S \rightarrow T). f (x x)) \ (\lambda x : (\mu S. S \rightarrow T). f (x x)) \]
\[ \text{fix}_T : (T \rightarrow T) \rightarrow T \]
Typing divergence

Infinitely many well-typed diverging functions

\[ \text{diverge}_T = \lambda \_ : \text{Unit}. \ \text{fix}_T (\lambda x : T. x) \]
\[ \text{diverge}_T : \text{Unit} \to T \]

Consequences: Systems with recursive types . . .

. . . do not have the strong normalisation property

. . . have at least one value of every type

. . . are useless as logics (every proposition is provable)
Two approaches towards formalising recursive types

What is the relation between a recursive type and its one-step unfolding?

\[ \mu T. \text{Unit} + \text{Nat} \times T \sim \text{Unit} + \text{Nat} \times (\mu T. \text{Unit} + \text{Nat} \times T) \]

Two approaches:

- equi-recursive approach
- iso-recursive approach
Equi-recursive approach

What is the relation between a recursive type and its one-step unfolding?

interchangeable in all contexts

Consequences:

- conceptually clean
- infinite type expressions
- implementation can be tricky
- may interfere with other advanced typing features
Iso-recursive approach

What is the relation between a recursive type and its one-step unfolding?

different but isomorphic

Consequences:

- conceptually awkward
- finite type expressions + fold/unfold operations
- implementation rather straightforward
- implementation can often be “piggybacked”
Folding and unfolding

unfold$_T$ and fold$_T$ witness the isomorphism

Unfold:

$\text{unfold}_{\mu X. T} : \mu X. T \rightarrow [X \rightarrow \mu X. T]^*T$

$$\begin{align*}
U &= \mu X. T_1 \\
\Gamma \vdash t_1 : U \\
\Gamma \vdash \text{unfold}_U(t_1) : [X \rightarrow U]^*T_1
\end{align*}$$

TUnfold

Fold:

$\text{fold}_{\mu X. T} : [X \rightarrow \mu X. T]^*T \rightarrow \mu X. T$

$$\begin{align*}
U &= \mu X. T_1 \\
\Gamma \vdash t_1 : [X \rightarrow U]^*T_1 \\
\Gamma \vdash \text{fold}_U(t_1) : U
\end{align*}$$

TFold
Piggybacking

\[
\text{unfold}_{\mu X. T} : \mu X. T \rightarrow [X \rightarrow \mu X. T]^* T \\
\text{fold}_{\mu X. T} : [X \rightarrow \mu X. T]^* T \rightarrow \mu X. T
\]

Lists of natural numbers:

\[
\text{unfold}_{\text{NatList}} : \text{NatList} \rightarrow \text{NatListBody} \\
\text{fold}_{\text{NatList}} : \text{NatListBody} \rightarrow \text{NatList}
\]

\[
\text{nil} = \text{fold}_{\text{NatList}}(\langle 1, \text{Unit} \rangle) \\
\text{cons} = \lambda n : \text{Nat}. \lambda l : \text{NatList}. \text{fold}_{\text{NatList}}(\langle 2, (n, l) \rangle)
\]
Overview

- Introducing recursive types
- **Reasoning about infinite trees**
  - Infinite trees
  - Regular trees and $\mu$-types
  - Induction and co-induction
- Membership tests
- Recursive types and subtyping
- Conclusions
Infinite trees

Let \((\Sigma, \text{ar} : \Sigma \to \mathbb{N})\) be a signature.

A tree is a partial function \(T \in \mathbb{N}^* \to \Sigma\) where

- \(T(\varepsilon)\) is defined,
- if \(T(\pi\sigma)\) is defined then \(T(\pi)\) is defined,
- if \((\text{ar} \circ T)(\pi) = k\), then \(T(\pi i)\) is defined for and only for \(1 \leq i \leq k\).

Terminology:

- **nodes**: \(\text{dom}(T)\)
- **root node**: \(\varepsilon \in \text{dom}(T)\)
- **labels**: \(\text{codom}(T)\)
- **daughter relation**: \(\sigma \in \text{daughters}_T(\pi) \iff T(\pi\sigma)\downarrow\)
Regular trees and $\mu$-types

$\mu$-types are compact representations of regular trees:

- $S$ is a **subtree** of $T$ if $S = \lambda \sigma. T(\pi \sigma)$ for some $\pi$.
- $T$ is **regular** if the set of its subtrees is finite.

Set of $\mu$-types:

$$T ::= X \in \mathcal{V} \mid T_1 \times T_2 \mid T_1 \rightarrow T_2 \mid \mu X. T'$$

Contractive $\mu$-types:

- $\mu X. X$ cannot reasonably be interpreted as a tree.
- allow only **contractive** $\mu$-types
- $T$ is contractive if it does not have the form $\mu X. \mu X_1 \ldots \mu X_n. X$
Review: Induction

Inductive definitions:

- start with a universe $U$ of values
- want to define $X \subseteq U$
- monotone generator function $F : \mathcal{P}(U) \to \mathcal{P}(U)$
- consider $\mu X. F(X)$

Example:

\[
N_0 = \emptyset \\
N_{k+1} = \{0\} \cup \{\text{succ}(n) \mid n \in N_k\} \\
N = \bigcup_{k=0}^{\infty} N_k = \mu k. N_k
\]

Inductively defined objects are finite.
Proof techniques for infinite trees

Co-induction can deal with infinite objects.

Co-inductive definitions:

- start with a universe $U$ of values
- want to define $X \subseteq U$
- monotone generator function $F : \mathcal{P}(U) \to \mathcal{P}(U)$
- consider $\nu X. F(X)$

Example: Infinite trees

- same generating function as for finite trees
- consider greatest instead of least fixed point
Induction and co-induction: Basics

Definition: Let $X$ be a subset of $\mathcal{U}$.

- $X$ is $F$-closed if $F(X) \subseteq X$.
- $X$ is $F$-consistent if $X \subseteq F(X)$.
- $X$ is a fixed point of $F$ if $F(X) = X$.

Theorem: Let $F \in \mathcal{P}(U) \to \mathcal{P}(U)$ be monotone.

1. The intersection of all $F$-closed sets is the least fixed point of $F$.
2. The union of all $F$-consistent sets is the greatest fixed point of $F$. 
**Principle of induction**

$$\mu X. F(X) := \bigcap \{ X \mid F(X) \subseteq X \}$$ is the least fixed point of $F$.

**Principle of induction:** $F(X) \subseteq X \Rightarrow \mu F \subseteq X$

**Proof technique:** To show that $\mu F \subseteq P$, show that $P$ is $F$-closed.

**Example:** Let $P$ be any property on natural numbers, which are taken to be defined by the generating function

$$F(N) = \{0\} \cup \{ \text{succ}(n) \mid n \in N \}.$$  

To show that all $n \in N$ satisfy the property $P$, show that $P$ is $F$-closed, i.e., that $\{0\} \subseteq P$ and that $\{ \text{succ}(p) \mid p \in P \} \subseteq P$. 
**Principle of co-induction**

\[ \nu X. F(X) := \bigcup \{ X \mid X \subseteq F(X) \} \] is the greatest fixed point of \( F \).

**Principle of co-induction:** \( X \subseteq F(X) \Rightarrow X \subseteq \nu F \)

**Proof technique:** To show that \( P \subseteq \nu F \), show that \( P \) is \( F \)-consistent.

**Example:** Let \( \leadsto \) be the reduction relation on functional programs, and let the set of diverging programs be defined by the generating function

\[ F(\uparrow) = \{ a \mid \exists b: (a \leadsto b \land b \in \uparrow) \} . \]

Consider an expression \( \Omega \) that reduces to itself \((\Omega \leadsto \Omega)\), and let \( P = \{ \Omega \} \). \( P \) is \( F \)-consistent, as \( \{ \Omega \} = P \subseteq F(P) \). Therefore, \( P \subseteq \uparrow \).
Overview

- Introducing recursive types
- Reasoning about infinite trees
- **Membership tests for infinite types**
  - Generic algorithm
  - Correctness and completeness
- Recursive types and subtyping
- Conclusions
Generating sets

When does an element \( x \in \mathcal{U} \) fall into the greatest (least) fixed point of \( F \)?

**Idea for an algorithm:** Start from \( \nu F \) (\( \mu F \)) and follow \( F \) backwards.

- problem: \( x \in \mathcal{U} \) can be generated by \( F \) in different ways
- danger of combinatorial explosion
- no problem if there is just one path backwards

**Generating sets:**

- \( G_X = \{ X \subseteq \mathcal{U} \mid x \in F(X) \} \)
- Any superset of a generating set for \( x \) is also a generating set for \( x \).
- \( F \) is called **invertible** iff \( \forall x \in \mathcal{U}: 0 \leq |G_x| \leq 1 \).
Support graph

Support set: Let $F$ be invertible.

$$\text{support}_F(x) = \begin{cases} X & \text{if } X \in G_x \text{ and } \forall X' \in G_x: X \subseteq X', \\ \uparrow & \text{if } G_x = \emptyset. \end{cases}$$

Support graph:

- nodes: supported and unsupported elements of $\mathcal{U}$
- edge $(x, y)$ whenever $y \in \text{support}(x)$
**Generic algorithm**

$X \subseteq \mathcal{U}$ is in the greatest fixed point of an invertible generating function $F$ if no unsupported elements are reachable from $x$ in the support graph of $F$:

$$
gfp_F(X) = \text{support}_F(X) \downarrow \land (\text{support}_F(X) \subseteq X \lor gfp_F(\text{support}_F(X) \cup X))$$

Reduction to a reachability problem in graphs
Partial correctness (1)

Let $F$ be invertible.

**Lemma:** $X \subseteq F(Y)$ if and only if $\text{support}_F(X) \downarrow$ and $\text{support}_F(X) \subseteq Y$.

**Proof:** Show that $x \in F(Y)$ if and only if $\text{support}_F(x) \downarrow$ and $\text{support}_F(x) \subseteq Y$.

- Assume $x \in F(Y)$. Then $G_x$ is non-empty: at least $Y$ is a generating set for $x$. In particular, since $F$ is invertible, $\text{support}_F(x)$, the smallest generating set, exists, and $\text{support}_F(x) \subseteq Y$.

- If $\text{support}_F(x) \subseteq Y$, then $F(\text{support}_F(x)) \subseteq F(Y)$ due to the monotonicity of $F$. By the definition of support, $x \in F(\text{support}(x))$, so $x \in F(Y)$.

**Lemma:** Suppose that $P$ is a fixed point of $F$. Then $X \subseteq P$ if and only if $\text{support}_F(X) \downarrow$ and $\text{support}_F(X) \subseteq P$.

**Proof:** Recall that $P = F(P)$ and apply the previous lemma.
Partial correctness (2)

\[ \text{gfp}_F(X) = \text{support}_F(X) \downarrow \land (\text{support}_F(X) \subseteq X \lor \text{gfp}_F(\text{support}_F(X) \cup X)) \]

Theorem:

1. If \( \text{gfp}_F(X) = \text{true} \), then \( X \subseteq \nu F \).
2. If \( \text{gfp}_F(X) = \text{false} \), then \( X \not\subseteq \nu F \).

Proof: Induction on the recursive structure of \( \text{gfp}_F \).

1. Assume \( \text{support}_F(X) \subseteq X \). By a previous lemma, \( X \subseteq F(X) \), i.e., \( X \) is \( F \)-consistent; thus, \( X \subseteq \nu F \) by the coinduction principle.

   Assume \( \text{gfp}_F(\text{support}_F(X) \cup X) = \text{true} \). By the induction hypothesis, \( \text{support}_F(X) \cup X \subseteq \nu F \), and so \( X \subseteq \nu F \).

2. ...
Partial correctness (3)

\[ \text{gfp}_F(X) = \text{support}_F(X) \downarrow \land (\text{support}_F(X) \subseteq X \lor \text{gfp}_F(\text{support}_F(X) \cup X)) \]

Theorem:

1. If \( \text{gfp}_F(X) = \text{true} \), then \( X \subseteq \nu F \).
2. If \( \text{gfp}_F(X) = \text{false} \), then \( X \notin \nu F \).

Proof: Induction on the recursive structure of \( \text{gfp}_F \).

1. . . .

2. Assume \( \text{support}_F(X) \uparrow \). Then, by a previous lemma, \( X \notin \nu F \).

   Assume \( \text{gfp}_F(\text{support}_F(X) \cup X) = \text{false} \). Then \( \text{support}_F(X) \cup X \notin \nu F \), i.e., \( X \notin \nu F \) or \( \text{support}_F(X) \notin \nu F \). Either way, \( X \notin \nu F \) – in the latter case by using a previous lemma.
Reachable elements

Problem: $\text{gfp}_F$ will diverge if $\text{support}_F(x)$ is infinite for some $x \in \mathcal{U}$.

Set of reachable elements:

$$\text{reachable}_F(X) = \bigcup_{n \geq 0} \text{predecessors}^n(X)$$

Definition: An invertible function $F$ is said to be finite state if $\text{reachable}_F(x)$ is finite for all $x \in \mathcal{U}$. 
Termination condition

\[ \text{gfp}_F(X) = \text{support}_F(X) \downarrow \land (\text{support}_F(X) \subseteq X \lor \text{gfp}_F(\text{support}_F(X) \cup X)) \]

**Theorem:** If \( F \) is finite state, \( \text{gfp}_F(X) \) terminates for any finite \( X \subseteq \mathcal{U} \).

**Proof:** Define \( \text{next}_F(X) := \text{support}_F(X) \cup X \).

For each call of \( \text{gfp}_F \), \( \text{next}_F(X) \subseteq \text{reachable}_F(X) \). Moreover, \( \text{next}_F(X) \) strictly increases on each recursive call. Since \( \text{reachable}_F(X) \) is finite, the following serves as a decreasing termination measure:

\[ m(\text{gfp}_F(X)) := |\text{reachable}_F(X)| - |\text{next}_F(X)| \]
Variants of the algorithms

Adding an accumulator:

- $\text{support}_F(X)$ is recomputed for every recursive call
- distinguish between goals ($X$) and assumptions ($A$)
- $\text{gfp}_F(X) = \text{true}$ if $\text{gfp}_F^a(\emptyset, X) = \text{true}$

Threading the assumptions:

- share support assumptions among calls at the same level of recursion
- return the set of assumptions, not $\text{true}/\text{false}$
- $\text{gfp}_F(\{x\}) = \text{true}$ if $\text{gfp}_F^t(\emptyset, x) \downarrow$
Overview

- Introducing recursive types
- Reasoning about infinite trees
- Membership tests
- **Recursive types and subtyping**
- Conclusions
Subtyping

Goal: Instantiating the generic algorithm with the subtyping relation.

Generating function for the subtyping relation:

\[ S(R) = \{ (S, \text{Top}) \mid S \in \mathcal{T}_{\mu} \} \]
\[ \quad \cup \{ (S_1 \times S_2, T_1 \times T_2) \mid (S_1, T_1) \in R, (S_2, T_2) \in R \} \]
\[ \quad \cup \{ (S_1 \to S_2, T_1 \to T_2) \mid (T_1, S_1) \in R, (S_2, T_2) \in R \} \]
\[ \quad \cup \{ (S, \mu X. T) \mid (S, [X \to \mu X. T]^* T) \in R \} \]
\[ \quad \cup \{ (\mu X. S, T) \mid ([X \to \mu X. S]^* S, T) \in R \} \]

Properties:

- monotone
- not invertible (but can be made so)
Proving termination (1)

Show that \( \text{reachable}_{\mu}(S, T) \) is finite for any pair \((S, T)\) of \(\mu\)-types.

**Bottom-up subexpressions** of \(\mu\)-types:

\[
\text{Sub}_B(R) = \{(T, T) \mid T \in \mathcal{T}_{\mu}\} \\
\cup \{(S, T_1 \times T_2) \mid (S, T_1) \in R\} \\
\cup \{(S, T_1 \times T_2) \mid (S, T_2) \in R\} \\
\cup \{(S, T_1 \rightarrow T_2) \mid (S, T_1) \in R\} \\
\cup \{(S, T_1 \rightarrow T_2) \mid (S, T_2) \in R\} \\
\cup \{([X \rightarrow \mu X. T]^*S, \mu X. T) \mid (S, T) \in R\}
\]

**Lemma:** \(\{S \mid (S, T) \in \mu \text{Sub}_B\} \) is finite.

**Proof:** Structural induction on \(T\).
Proving termination (2)

Top-down subexpressions of $\mu$-types:

$$\text{Sub}_T(R) = \{ (T, T) \mid T \in \mathcal{T}_\mu \}$$
$$\cup \{ (S, T_1 \times T_2) \mid (S, T_1) \in R \}$$
$$\cup \{ (S, T_1 \times T_2) \mid (S, T_2) \in R \}$$
$$\cup \{ (S, T_1 \rightarrow T_2) \mid (S, T_1) \in R \}$$
$$\cup \{ (S, T_1 \rightarrow T_2) \mid (S, T_2) \in R \}$$
$$\cup \{ (S, \mu X. T) \mid (S, [X \rightarrow \mu X. T]^* T) \in R \}$$

Lemma: $\mu \text{Sub}_T \subseteq \mu \text{Sub}_B$

Proof: requires some work
Conclusions

- recursive types = infinite trees
- proof technique: co-induction
- checking membership in greatest fixed points
- application to subtyping