Seminar on Types and Programming Languages
Programming Systems Lab, Saarland University

Recursive types

Marco Kuhlmann

2003-03-05

Recursive types are ubiquitous

Lists of natural numbers:

$$\mathsf{NatList} = \mathbf{nil} : \mathsf{Unit} \mid \mathbf{cons} : \mathsf{Nat} \times \mathsf{NatList}$$

$$\mu T.\, \mathsf{Unit} + \mathsf{Nat} \times T$$

Dependency trees:

$$\mathsf{DTree} = \mathbf{t} : \mathsf{Lex} \mid \mathbf{n} : \mathsf{Lex} \times (\mathsf{Role} \times \mathsf{DTree})\ \mathsf{List}$$

$$\mu T.\, \mathsf{Lex} + \mathsf{Lex} \times (\mathsf{Role} \times T)\ \mathsf{List}$$

Functional counters:

$$\mathsf{Counter} = \mathbf{get} : \mathsf{Nat} \mid \mathbf{inc} : \mathsf{Unit} \to \mathsf{Counter}$$

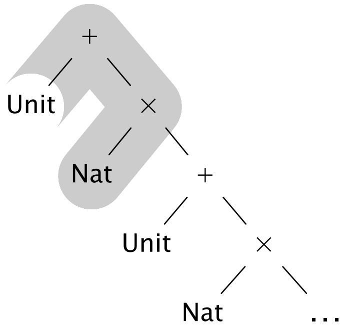
$$\mu T.\, \mathsf{Nat} + (\mathsf{Unit} \to T)$$

Recursive types as infinite trees

Recursive type definitions = specifications of infinite regular trees

Example:

$$\mathsf{NatList} = \mathbf{nil} : \mathsf{Unit} \mid \mathbf{cons} : \mathsf{Nat} \times \mathsf{NatList}$$



Introduction

- **Introducing recursive types**
  - Intuition
  - Expressive power
  - Formalities
- Reasoning about infinite trees
- Membership tests
- Recursive types and subtyping
- Conclusions

Typing the fixed-point combinator

$$\mathsf{fix} = \lambda f.\ (\lambda x. f(xx))\,(\lambda x. f(xx))$$

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 $\mu S.\, S \to T$

A well-typed fixed-point combinator

$$\mathsf{fix}_T = \lambda f : T \to T.\ (\lambda x : (\mu S.\, S \to T). f(xx))\,(\lambda x : (\mu S.\, S \to T). f(xx))$$
$$\mathsf{fix}_T\ :\ (T \to T) \to T$$

Typing divergence

Infinitely many well-typed diverging functions

$$\mathsf{diverge}_T = \lambda\,\_ : \mathsf{Unit}.\ \mathsf{fix}_T\,(\lambda\,x : T.\ x)$$
$$\mathsf{diverge}_T\ :\ \mathsf{Unit} \to T$$

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?

$$\mu T.\,\mathsf{Unit} + \mathsf{Nat} \times T \quad \sim \quad \mathsf{Unit} + \mathsf{Nat} \times (\mu T.\,\mathsf{Unit} + \mathsf{Nat} \times T)$$

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:

$$\mathsf{unfold}_{\mu X.\,T} : \mu X.\,T \to [X \to \mu X.\,T]^* T$$

$$\frac{U = \mu X.\,T_1 \qquad \Gamma \vdash t_1 : U}{\Gamma \vdash \mathsf{unfold}_U(t_1) : [X \to U]^* T_1} \text{ TUnfold}$$

Fold:

$$\mathsf{fold}_{\mu X.\,T} : [X \to \mu X.\,T]^* T \to \mu X.\,T$$

$$\frac{U = \mu X.\,T_1 \qquad \Gamma \vdash t_1 : [X \to U]^* T_1}{\Gamma \vdash \mathsf{fold}_U(t_1) : U} \text{ TFold}$$

Piggybacking

$$\mathsf{unfold}_{\mu X.\,T} : \mu X.\,T \to [X \to \mu X.\,T]^* T$$
$$\mathsf{fold}_{\mu X.\,T} : [X \to \mu X.\,T]^* T \to \mu X.\,T$$

Lists of natural numbers:

$$\mathsf{unfold}_{\mathsf{NatList}} : \mathsf{NatList} \to \mathsf{NatListBody}$$
$$\mathsf{fold}_{\mathsf{NatList}} : \mathsf{NatListBody} \to \mathsf{NatList}$$

$$\textbf{nil} = \mathsf{fold}_{\mathsf{NatList}}(\langle 1, \mathsf{Unit}\rangle)$$
$$\textbf{cons} = \lambda\, n : \mathsf{Nat}.\ \lambda\, l : \mathsf{NatList}.\ \mathsf{fold}_{\mathsf{NatList}}(\langle 2, (n, l)\rangle)$$

Overview

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

Infinite trees

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

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

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

Terminology:

- **nodes:** $\mathrm{dom}(\mathsf{T})$
- **root node:** $\varepsilon \in \mathrm{dom}(\mathsf{T})$
- **labels:** $\mathrm{codom}(\mathsf{T})$
- **daughter relation:** $\sigma \in \mathrm{daughters}_{\mathsf{T}}(\pi) \iff \mathsf{T}(\pi\sigma)\downarrow$

Regular trees and μ-types

μ-types are compact representations of regular trees:

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

Set of μ-types:

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

Contractive μ-types:

- $\mu X.\, X$ cannot reasonably be interpreted as a tree.
- allow only **contractive** μ-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 $F : \mathfrak{P}(U) \to \mathfrak{P}(U)$
- consider $\mu X.\, F(X)$

Example:

$$
\begin{aligned}
N_0 &= \varnothing \\
N_{k+1} &= \{0\} \cup \{\, \mathsf{succ}(n) \mid n \in N_k \,\} \\
N &= \bigcup_{k=0}^{\infty} N_k = \mu k.\, N_k
\end{aligned}
$$

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 $F : \mathfrak{P}(U) \rightarrow \mathfrak{P}(U)$
- 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 $F \in \mathfrak{P}(U) \rightarrow \mathfrak{P}(U)$ 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

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

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 $\{\, \mathsf{succ}(p) \mid p \in P \,\} \subseteq P$.

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 $\rightsquigarrow$ be the reduction relation on functional programs, and let the set of diverging programs be defined by the generating function

$$F(\uparrow) = \{\, a \mid \exists b\colon\ (a \rightsquigarrow b \wedge b \in \uparrow) \,\}.$$

Consider an expression Ω that reduces to itself ($\Omega \rightsquigarrow \Omega$), 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 νF (μF) 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 $\forall x \in \mathcal{U}\colon 0 \leq |G_x| \leq 1$.

Support graph

Support set: Let F be invertible.

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

Support graph:

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

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:

$$\mathsf{gfp}_F(X) = \mathsf{support}_F(X)\!\downarrow \wedge \big(\mathsf{support}_F(X) \subseteq X \vee \mathsf{gfp}_F(\mathsf{support}_F(X) \cup X)\big)$$

Reduction to a reachability problem in graphs

Partial correctness (1)

Let F be invertible.

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

Proof: Show that $x \in F(Y)$ if and only if support$_F(x)\downarrow$ and 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, support$_F(x)$, the smallest generating set, exists, and support$_F(x) \subseteq Y$.
- If support$_F(x) \subseteq Y$, then $F($support$_F(x)) \subseteq F(Y)$ due to the monotonicity of F. By the definition of support, $x \in F($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 support$_F(X)\downarrow$ and support$_F(X) \subseteq P$.

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

Partial correctness (2)

$\mathsf{gfp}_F(X) = \mathsf{support}_F(X){\downarrow} \wedge \big(\mathsf{support}_F(X) \subseteq X \vee \mathsf{gfp}_F(\mathsf{support}_F(X) \cup X)\big)$

Theorem:

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

Proof: Induction on the recursive structure of gfp_F.

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

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

2. ...

Partial correctness (3)

$\mathsf{gfp}_F(X) = \mathsf{support}_F(X){\downarrow} \wedge \left(\mathsf{support}_F(X) \subseteq X \vee \mathsf{gfp}_F(\mathsf{support}_F(X) \cup X)\right)$

Theorem:

1. If $\mathsf{gfp}_F(X) = \mathbf{true}$, then $X \subseteq \nu F$.
2. If $\mathsf{gfp}_F(X) = \mathbf{false}$, then $X \nsubseteq \nu F$.

Proof: Induction on the recursive structure of gfp_F.

1. ...
2. Assume $\mathsf{support}_F(X){\uparrow}$. Then, by a previous lemma, $X \nsubseteq \nu F$.

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

Reachable elements

Problem: 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

$\mathsf{gfp}_F(X) = \mathsf{support}_F(X){\downarrow} \wedge \big(\mathsf{support}_F(X) \subseteq X \vee \mathsf{gfp}_F(\mathsf{support}_F(X) \cup X)\big)$

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

Proof: Define $\mathsf{next}_F(X) := \mathsf{support}_F(X) \cup X$.

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

$$m(\mathsf{gfp}_F(X)) := |\mathsf{reachable}_F(X)| - |\mathsf{next}_F(X)|$$

Variants of the algorithms

Adding an accumulator:

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

Threading the assumptions:

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

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:

$$
\begin{aligned}
S(R) = {} & \{ (S, \mathsf{Top}) \mid S \in \mathcal{T}_\mu \} \\
& \cup \{ (S_1 \times S_2, T_1 \times T_2) \mid (S_1, T_1) \in R, (S_2, T_2) \in R \} \\
& \cup \{ (S_1 \to S_2, T_1 \to T_2) \mid (T_1, S_1) \in R, (S_2, T_2) \in R \} \\
& \cup \{ (S, \mu X.\, T) \mid (S, [X \to \mu X.\, T]^* T) \in R \} \\
& \cup \{ (\mu X.\, S, T) \mid ([X \to \mu X.\, S]^* S, T) \in R \}
\end{aligned}
$$

Properties:

- monotone
- not invertible (but can be made so)

Proving termination (1)

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

Bottom-up subexpressions of μ-types:

$$
\begin{aligned}
\mathsf{Sub}_B(R) = {} & \{\, (T, T) \mid T \in \mathcal{T}_\mu \,\} \\
& \cup \{\, (S, T_1 \times T_2) \mid (S, T_1) \in R \,\} \\
& \cup \{\, (S, T_1 \times T_2) \mid (S, T_2) \in R \,\} \\
& \cup \{\, (S, T_1 \to T_2) \mid (S, T_1) \in R \,\} \\
& \cup \{\, (S, T_1 \to T_2) \mid (S, T_2) \in R \,\} \\
& \cup \{\, ([X \to \mu X.\, T]^* S, \mu X.\, T) \mid (S, T) \in R \,\}
\end{aligned}
$$

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

Proof: Structural induction on T.

Proving termination (2)

Top-down subexpressions of μ-types:

$$\begin{aligned}
\mathsf{Sub}_T(R) = & \; \{\, (T, T) \mid T \in \mathcal{T}_\mu \,\} \\
& \cup \{\, (S, T_1 \times T_2) \mid (S, T_1) \in R \,\} \\
& \cup \{\, (S, T_1 \times T_2) \mid (S, T_2) \in R \,\} \\
& \cup \{\, (S, T_1 \to T_2) \mid (S, T_1) \in R \,\} \\
& \cup \{\, (S, T_1 \to T_2) \mid (S, T_2) \in R \,\} \\
& \cup \{\, (S, \mu X.\, T) \mid (S, [X \mapsto \mu X.\, T]^* T) \in R \,\}
\end{aligned}$$

Lemma: $\mu\, \mathsf{Sub}_T \subseteq \mu\, \mathsf{Sub}_B$

Proof: requires some work

Conclusions

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