Metatheory of Recursive Types

Types and Programming Languages - B. Pierce

07/02/2003

Élodie-Jane Sims.
Subject

- Checking membership in greatest fixed points
- Application to subtyping
Induction & Coinduction

\( \mathcal{U} \): universal set
\( F \in \mathcal{P}(\mathcal{U}) \rightarrow \mathcal{P}(\mathcal{U}) \): generating function (monotone)

- \( X \) is \( F \)-closed: \( F(X) \subseteq X \)
- \( X \) is \( F \)-consistent: \( X \subseteq F(X) \)
- \( X \) is a fixed point of \( F \): \( X = F(X) \)
- Th Knaster-Tarski
  \[ \diamond \quad \mu F = \bigcap \{ X \mid F(X) \subseteq X \} \] (least fixed point)
  \[ \diamond \quad \nu F = \bigcup \{ X \mid X \subseteq F(X) \} \] (greatest fixed point)

**Principle of induction** \( F(X) \subseteq X \Rightarrow \mu F \subseteq X \)
“To show that \( \mu F \) has a property we just show that this property is preserved by \( F \)”

**Principle of coinduction** \( X \subseteq F(X) \Rightarrow X \subseteq \nu F \)
“To check whether \( x \in \nu F \) we just need an \( F \)-consistent set \( X \) containing \( x \)”
A Disgression on Transitivity

2 standard formulations:

– declarative presentation with explicit transitivity rule

– algorithmic presentation without explicit transitivity rule

The declarative one work for induction definitions (finite case) but not for coinduction (infinite case).

*The problem is to apply the rule* \( \frac{S < U \quad U <: T}{S <: T} \) *how to find* \( U \) ?
Membership Checking

$x \in \mathcal{U}$, $F$ generating function: $x \in \nu F$?

**Idea of algo**: start from $\nu F$ and go backward with $F^{-1}$ until reaching $x$ or not.
Problems: there can be several way to go backward, danger of combinatorial explosion.

- **Generating set** $G_x = \{X \subseteq \mathcal{U} | x \in F(X)\}$
- **$F$ invertible**: $\forall x \in \mathcal{U}$.
  $G_x$ has at most one smallest element
- **Support set** ($F^{-1}$) if $F$ is invertible:
  
  $$\text{support}_F(x) = \begin{cases} X & \text{if } X \in G_x \text{ and } \forall X' \in G_x.X \subseteq X' \\ \uparrow & \text{if } G_x = \emptyset \end{cases}$$

- $x$ is $F$-supported if $\text{support}_F(x) \downarrow$
- $x$ is $F$-unsupported if $\text{support}_F(x) \uparrow$
- **Support graph**:
  - nodes = $F$-supported and $F$-unsupported elements of $\mathcal{U}$
  - edge $(x, y)$ if $y \in \text{support}_F(x)$

\[
\{F\text{-unsupported in the support graph}\} \cap \text{reachable}_F(x) = \emptyset
\]

$F$ **finite state** if $\forall x.\text{reachable}_F(x)$ is finite

$\Rightarrow$ algo terminate if $F$ is finite state
Example

Rules

F:

\[ F(\emptyset) = \{g\} \]

\[ F(\{g\}) = \{f\} \]

\[ \ldots \]
The algorithm and the principle of coinduction

I write $F^{-1}$ without having it defined, it’s just a hand show of how the coinduction principle is used.

We take $\{x\}$, compute $F^{-1}(\{x\})$, $F^{-1}(F^{-1}(\{x\}) \cup \{x\})$, $F^{-1}(F^{-1}(X_n) \cup X_n)$

Until $F^{-1}(F^{-1}(X_n) \cup X_n) \subseteq F^{-1}(X_n) \cup X_n$

Then this last set is $F$-consistent since:

$F^{-1}(X_n) = F(F^{-1}(F^{-1}(X_n) \cup X_n))$

and we also have $x$ in it so by the principle of coinduction $x \in \nu F$
Tree-TYPES: univers \( \mathcal{U} \)

\(\rightarrow, \times, \text{Top} : \) types constructors
\(\bullet : \) empty sequence
\(\pi, \sigma : \) concatenation of sequences

**Tree-type** partial function
\(T \in \{1, 2\}^* \rightarrow \{\rightarrow, \times, \text{Top}\} \) satisfying
- \(T(\bullet)\) defined
- \(T(\pi, \sigma)\) defined \(\Rightarrow T(\pi)\) defined
- \(T(\pi) = \rightarrow \) or \(\times \Rightarrow T(\pi, 1), T(\pi, 2)\) defined
- \(T(\pi) = \text{Top} \Rightarrow T(\pi, 1), T(\pi, 2)\) undefined

**finite tree-type**: \(\text{dom}(T)\) finite
\(\mathcal{T} : \) set of all tree-types
\(\mathcal{T}_f : \) set of all finite tree-types

\[
\begin{align*}
T & : = \ \text{Top} \\
& | \ T \times T \\
& | \ T \rightarrow T
\end{align*}
\]

\(\mathcal{U} = \) all finite and \(\infty\) trees labelled with \(\text{Top}, \rightarrow, \times\)
\(\mathcal{T} = \text{gfp} \) of the generating function described by the grammar above
\(\mathcal{T}_f = \text{lfp} \ldots\)
Subtyping: generating functions

$F$'s

**Finite subtyping**: $S$ is a subtype of $T$ iff $(S, T') \in \mu S_f$

$$S_f : \mathcal{P}(T_f \times T_f) \to \mathcal{P}(T_f \times T_f)$$

$$S_f(R) = \{(T, \text{Top}) \mid T \in T_f\} \cup \{(S_1 \times S_2, T_1 \times T_2) \mid (S_1, T_1), (S_2, T_2) \in R\} \cup \{(S_1 \to S_2, T_1 \to T_2) \mid (T_1, S_1), (S_2, T_2) \in R\}$$

**Infinite subtyping**: $S$ is a subtype of $T$ iff $(S, T) \in \nu S$

$$S : \mathcal{P}(T \times T) \to \mathcal{P}(T \times T)$$

$$S(R) = \{(T, \text{Top}) \mid T \in T\} \cup \{(S_1 \times S_2, T_1 \times T_2) \mid (S_1, T_1), (S_2, T_2) \in R\} \cup \{(S_1 \to S_2, T_1 \to T_2) \mid (T_1, S_1), (S_2, T_2) \in R\}$$
Regular Trees

**Motivations**: if types are “regular” then reachable sets remain finites and subtype checking algo always terminate.

\[ S \text{ sub-tree } T : \exists \pi. S = \lambda \sigma . T(\pi, \sigma) \]

**Regular tree**: the set of subtrees is finite

\[ T_r = \text{all regular tree-types} \]

\[ S_r = \text{restriction of } S \text{ to } T_r \]

\[ S_r \text{ is finite state} \]

- we can have a decision procedure which terminate but we need a finite representation for regular trees
\(\mu\)-Types

\(\{X_i\}\) fixed countable set of type variables

\(\mathcal{T}_m^{\text{raw}}\): set of raw \(\mu\)-types, set of expressions:

\[
T : = X \\
\text{Top} \\
T \times T \\
T \rightarrow T \\
\mu X. T
\]

raw \(\mu\)-type contractive: for all subexpressions of the form \(\mu X. \mu X_1 \ldots \mu X_n. S\) we have \(S \neq X\).

(because \(\mu X. X\) can’t be interpreted as a type)

\(\mu - \text{type}\): contractive raw \(\mu\)-type

\(\mathcal{T}_m\): set of \(\mu\)-types

\(\mu\)-types subtyping: \(S\) is a subtype of \(T\) iff \((S, T) \in \nu S_m\)

\[
S_m : \quad \mathcal{P}(\mathcal{T}_m \times \mathcal{T}_m) \rightarrow \mathcal{P}(\mathcal{T}_m \times \mathcal{T}_m)
\]

\[
S_m(R) = \{(T, \text{Top}) \mid T \in \mathcal{T}_m\} \\
\cup \quad \{(S_1 \times S_2, T_1 \times T_2) \mid (S_1, T_1), (S_2, T_2) \in R\} \\
\cup \quad \{(S_1 \rightarrow S_2, T_1 \rightarrow T_2) \mid (T_1, S_1), (S_2, T_2) \in R\} \\
\cup \quad \{(S, \mu X. T) \mid (S, [X \mapsto \mu X. T] T) \in R\} \\
\cup \quad \{([X \mapsto \mu X. S] S, T) \in R \\
\quad \quad \quad \quad \quad T \neq \text{Top} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad T \neq \mu Y. T_1\}
\]

the last adding conditions are for reversibility
Link from $\mu$-types to tree-types

\texttt{treeof} : closed $\mu$-types $\rightarrow$ tree-types

\begin{align*}
\text{treeof}(\text{Top})(\bullet) &= \text{Top} \\
\text{treeof}(T_1 \rightarrow T_2)(\bullet) &= \rightarrow \\
\text{treeof}(T_1 \times T_2)(\bullet) &= \times \\
\text{treeof}(T_1 \rightarrow T_2)(i, \pi) &= \text{treeof}(T_i)(\pi) \\
\text{treeof}(T_1 \times T_2)(i, \pi) &= \text{treeof}(T_i)(\pi) \\
\text{treeof}(\mu X.T)(\pi) &= \text{treeof}([X \mapsto \mu X.T]T)(\pi)
\end{align*}

\begin{align*}
(S, T) \in \nu S_m \text{ iff } \text{treeof}(S, T) \in \nu S
\end{align*}
Instanciation of the general algorithm for $\mu$-types

\[
\text{support}_{S_m}(S, T) = \begin{cases} \\
\emptyset & \text{if } T = \text{Top} \\
\{(S_1, T_1), (S_2, T_2)\} & \text{if } S = S_1 \times S_2 \text{ and } T = T_1 \times T_2 \\
\{(S_1, T_1), (S_2, T_2)\} & \text{if } S = S_1 \rightarrow S_2 \text{ and } T = T_1 \rightarrow T_2 \\
\{(S, [x \mapsto \mu x.T_1] T_1)\} & \text{if } T = \mu x.T_1 \\
\{([x \mapsto \mu x.S_1]S_1, T_1)\} & \text{if } S = \mu x.S_1 \text{ and } T = \mu x.T_1, T \neq \text{Top} \\
\uparrow & \text{otherwise} \\
\end{cases}
\]

$S_m$ is invertible because $\text{support}_{S_m}$ is well-defined.

For any $\mu$-types $S, T$, $\text{reachables}_{S_m}(S, T)$ is finite.