21 *Metatheory of Recursive Types*

In Chapter 20 we saw two alternative presentations of recursive types: equi-recursive types, which are definitionally equivalent to their unfoldings, and iso-recursive types, where this equivalence is explicitly witnessed by `fold` and `unfold` terms. In this chapter, we develop the theoretical foundations of typecheckers for equi-recursive types. (Implementing iso-recursive types is comparatively straightforward.) We will deal with a system including both recursive types and subtyping, since these are often combined in practice. A system with equi-recursive types but not subtyping would be only a little simpler, since we would still need to check equivalence of recursive types.

We saw in Chapter 20 that subtyping for equi-recursive types can be understood intuitively in terms of infinite subtyping derivations over infinite types. Our job here is to make these intuitions precise using the mathematical framework of *coinduction,* and to draw a precise connection between infinite trees and derivations and the finite representations manipulated by an actual subtyping algorithm.

We begin in §21.1 by reviewing the basic theory of inductive and coinductive definitions and their associated proof principles. §21.2 and §21.3 instantiate this general theory for the case of subtyping, defining both the familiar inductive subtype relation on finite types and its coinductive extension to infinite types. §21.4 makes a brief detour to consider some issues connected with the rule of transitivity (a notorious troublemaker in subtyping systems, as we have seen). §21.5 derives simple algorithms for checking membership in inductively and co-inductively defined sets; §21.6 considers more refined algorithms. These algorithms are applied to subtyping for the important special case of "regular" infinite types in §21.7. §21.8 introduces μ-types as a finite notation for representing infinite types and proves that the more complex (but finitely implementable) subtype relation on μ-types corresponds to

---

The system studied in this chapter is the simply typed calculus with subtyping (Figure 15-1), products (11-5), and equi-recursive types. The corresponding checker is `equirec`.

the ordinary coinductive definition of subtyping between infinite types. §21.9 proves termination of the subtyping algorithm for μ-types. §21.10 compares this algorithm with another algorithm due to Amadio and Cardelli. §21.11 briefly discusses iso-recursive types.

21.1 Induction and Coinduction

Assume we have fixed some *universal set* $\mathcal{U}$ as the domain of discourse for our inductive and coinductive definitions. $\mathcal{U}$ represents the set of "everything in the world," and the role of an inductive or coinductive definition will be to pick out some subset of $\mathcal{U}$. (Later on, we are going to choose $\mathcal{U}$ to be the set of all pairs of types, so that subsets of $\mathcal{U}$ are relations on types. For the present discussion, an arbitrary set $\mathcal{U}$ will do.)

21.1.1 Definition: A function $F \in \mathcal{P}(\mathcal{U}) \to \mathcal{P}(\mathcal{U})$ is *monotone* if $X \subseteq Y$ implies $F(X) \subseteq F(Y)$. (Recall that $\mathcal{P}(\mathcal{U})$ is the set of all subsets of $\mathcal{U}$.) □

In the following, we assume that F is some monotone function on $\mathcal{P}(\mathcal{U})$. We often refer to F as a *generating function*.

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

1. X is *F-closed* if $F(X) \subseteq X$.
2. X is *F-consistent* if $X \subseteq F(X)$.
3. X is a *fixed point* of F if $F(X) = X$. □

A useful intuition for these definitions is to think of the elements of $\mathcal{U}$ as some sort of statements or assertions, and of F as representing a "justification" relation that, given some set of statements (premises), tells us what new statements (conclusions) follow from them. An F-closed set, then, is one that cannot be made any bigger by adding elements justified by F—it already contains all the conclusions that are justified by its members. An F-consistent set, on the other hand, is one that is "self-justifying": every assertion in it is justified by other assertions that are also in it. A fixed point of F is a set that is both closed and consistent: it includes all the justifications required by its members, all the conclusions that follow from its members, and nothing else.

21.1.3 Example: Consider the following generating function on the three-element universe $\mathcal{U} = \{a, b, c\}$:

$$\begin{array}{lcl@{\qquad}lcl}
E_1(\emptyset) & = & \{c\} & E_1(\{a,b\}) & = & \{c\} \\
E_1(\{a\}) & = & \{c\} & E_1(\{a,c\}) & = & \{b,c\} \\
E_1(\{b\}) & = & \{c\} & E_1(\{b,c\}) & = & \{a,b,c\} \\
E_1(\{c\}) & = & \{b,c\} & E_1(\{a,b,c\}) & = & \{a,b,c\}
\end{array}$$

There is just one E_1-closed set—$\{a, b, c\}$—and four E_1-consistent sets—$\emptyset$, $\{c\}$, $\{b, c\}$, $\{a, b, c\}$.

E_1 can be represented compactly by a collection of *inference rules:*

$$\frac{}{c} \qquad \frac{c}{b} \qquad \frac{b \quad c}{a}$$

Each rule states that if all of the elements above the bar are in the input set, then the element below is in the output set. □

21.1.4 Theorem [Knaster-Tarski (Tarski, 1955)]:

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. □

Proof: We consider only part (2); the proof of part (1) is symmetric. Let $C = \{X \mid X \subseteq F(X)\}$ be the collection of all F-consistent sets, and let P be the union of all these sets. Taking into account the fact that F is monotone and that, for any $X \in C$, we know both that X is F-consistent and that $X \subseteq P$, we obtain $X \subseteq F(X) \subseteq F(P)$. Consequently, $P = \bigcup_{X \in C} X \subseteq F(P)$, i.e. P is F-consistent. Moreover, by its definition, P is the largest F-consistent set. Using the monotonicity of F again, we obtain $F(P) \subseteq F(F(P))$. This means, by the definition of C, that $F(P) \in C$. Hence, as for any member of C, we have $F(P) \subseteq P$, i.e. P is F-closed. Now we have established both that P is the largest F-consistent set and that P is a fixed point of F, so P is the largest fixed point. □

21.1.5 Definition: The least fixed point of F is written μF. The greatest fixed point of F is written νF. □

21.1.6 Example: For the sample generating function E_1 shown above, we have $\mu E_1 = \nu E_1 = \{a, b, c\}$. □

21.1.7 Exercise [★]: Suppose a generating function E_2 on the universe $\{a, b, c\}$ is defined by the following inference rules:

$$\frac{}{a} \qquad \frac{c}{b} \qquad \frac{a \quad b}{c}$$

Write out the set of pairs in the relation E_2 explicitly, as we did for E_1 above. List all the E_2-closed and E_2-consistent sets. What are μE_2 and νE_2? □

Note that μF itself is F-closed (hence, it is the smallest F-closed set) and that νF is F-consistent (hence, it is the largest F-consistent set). This observation gives us a pair of fundamental reasoning tools:

21.1.8 Corollary [of 21.1.4]:

1. *Principle of induction:* If X is F-closed, then $\mu F \subseteq X$.
2. *Principle of coinduction:* If X is F-consistent, then $X \subseteq \nu F$. □

The intuition behind these principles comes from thinking of the set X as a predicate, represented as its characteristic set—the subset of $\mathcal{U}$ for which the predicate is true; showing that property X holds of an element x is the same as showing that x is in the set X. Now, the induction principle says that any property whose characteristic set is closed under F (i.e., the property is preserved by F) is true of all the elements of the inductively defined set μF.

The coinduction principle, on the other hand, gives us a method for establishing that an element x is *in* the coinductively defined set νF. To show $x \in \nu F$, it suffices to find a set X such that $x \in X$ and X is F-consistent. Although it is a little less familiar than induction, the principle of coinduction is central to many areas of computer science; for example, it is the main proof technique in theories of concurrency based on *bisimulation*, and it lies at the heart of many *model checking* algorithms.

The principles of induction and coinduction are used heavily throughout the chapter. We do not write out every inductive argument in terms of generating functions and predicates; instead, in the interest of brevity, we often rely on familiar abbreviations such as structural induction. Coinductive arguments are presented more explicitly.

21.1.9 Exercise [Recommended, ★★★]: Show that the principles of ordinary induction on natural numbers (2.4.1) and lexicographic induction on pairs of numbers (2.4.4) follow from the principle of induction in 21.1.8. □

21.2 Finite and Infinite Types

We are going to instantiate the general definitions of greatest fixed points and the coinductive proof method with the specifics of subtyping. Before we can do this, though, we need to show precisely how to view types as (finite or infinite) trees.

For brevity, we deal in this chapter with just three type constructors: $\rightarrow$, $\times$, and Top. We represent types as (possibly infinite) trees with nodes labeled by