Recursive Operator Definitions

Georges Gonthier and Leslie Lamport

11 May 2020

Abstract

TLA\(^+\) originally allowed recursive function definitions, but not recursive operator definitions, because it was not clear how to define their semantics. They were added to the language in 2006 after we discovered how to define a satisfactory semantics for them. We describe that semantics here.

Contents

1 Introduction 2

2 The Problem 3

3 Simple Recursive Definitions 4
   3.1 The Semantics 4
   3.2 Inductive Definitions 6
   3.3 Correctness for Inductive Definitions 7

4 Multiple Arguments 14

5 Mutual Recursion 16

References 17

Clickable links are colored like this.
1 Introduction

Recursive function definitions were allowed in the first version of TLA+ [2]. The definition

(1) \[ f[n \in \text{Nat}] \triangleq \text{if } n = 0 \text{ then } 1 \text{ else } n \cdot f[n - 1] \]

is an abbreviation for:

(2) \[ f \triangleq \text{choose } g : g = [n \in \text{Nat} \mapsto \text{if } n = 0 \text{ then } 1 \text{ else } n \cdot g[n - 1]] \]

However, recursive operator definitions were not allowed because it wasn’t known how to assign a meaning to them.

Most of the time, recursive function definitions suffice—even to define a recursively defined operator. For example, consider the operator Cardinality recursively defined as follows so \( \text{Cardinality}(S) \) is the number of elements in a finite set \( S \):

\[
\text{Cardinality}(S) \triangleq \begin{cases} 
0 & \text{if } S = \{\} \\
1 + \text{Cardinality}(S \setminus \{x : x \in S\}) & \text{else}
\end{cases}
\]

It can be defined as follows using a recursively defined function:

\[
\text{Cardinality}(S) \triangleq \begin{cases} 
0 & \text{if } T = \{\} \\
1 + \text{Cardinality}(T \setminus \{x : x \in T\}) & \text{else}
\end{cases}
\]

While not mathematically necessary, recursive operator definitions may be necessary in practice. Defining a recursive function requires defining the function’s domain, but that definition may be extremely complicated and the TLC model checker may not be able to evaluate it. This was the case with a specification of the PlusCal to TLA+ translation—a specification that was tested by having the actual PlusCal translator call TLC to evaluate it to perform part of the translation. As of now, the TLAPS proof system handles only recursive function definitions, not recursive operator definitions.

Recursive operator definitions were added to TLA+ when, in 2005, we figured out how to give them a correct semantics. This note belatedly explains that semantics and what “correct” means. The discussion here is informal but, we believe, rigorous. Our results are not quite expressed in TLA+ because they require declarations of higher-level operator parameters, while TLA+ only allows the definition of such operators. However, the meaning of those declarations should be clear.
Many of our results were independently discovered by Charguéraud [1]. While he was concerned with recursive definitions of functions rather than operators, some of his definitions and results closely match ours.

2 The Problem

To appreciate the problem posed by recursive operator definitions, consider this example:

\[ Op(x) \triangleq \text{choose } y : y \neq Op(x) \]

It would appear to define \( Op \) so that \( Op(42) \neq Op(42) \), which is impossible since every value equals itself.

Most logic texts that discuss definitions consider them to be axioms, so the meaning of (3) would be:

\[ \text{Axiom } \forall x : Op(x) = \text{choose } y : y \neq Op(x) \]

Since this axiom implies the false formula \( Op(42) \neq Op(42) \), (3) could not be a legal definition. To be legal, a recursive definition would have to satisfy some rule, and showing that the rule is satisfied would essentially require proving a theorem.

In TLA\(^+\), a definition simply asserts that one expression is a syntactic abbreviation for another expression. An ordinary, non-recursive definition

\[ Op(x) \triangleq \ldots \]

asserts that, for any expression \( e \), the expression \( Op(e) \) is an abbreviation for the expression obtained by syntactically substituting the expression \( e \) for \( x \) in the expression to the right of the \( \triangleq \). There is no need to prove a theorem to define a syntactic abbreviation.

There is nothing magic in declaring definitions to be abbreviations. To use the definition (1), we will have to prove this:

\[ \text{Theorem } \forall n \in \text{Nat} : f[n] = \text{if } n = 0 \text{ then } 1 \text{ else } n \ast f[n - 1] \]

The proof of the theorem is, of course, the same proof needed to justify a recursive definition of factorial if the meaning of that definition is taken to be an axiom.

Mathematicians write proofs, so it makes little difference to them whether they have to write a proof to make a definition or to use it. However, most TLA\(^+\) users are engineers who don’t write proofs. If a proof were required
to write a recursive operator definition in TLA
, the recursive definitions one could write would have to be constrained so that the proof was obvious enough to be found by the parser. Such a constraint would have been unacceptably restrictive without drastically changing TLA
. In particular, it would have required complicating the language by adding some form of typing.

We therefore had to provide a semantics for recursive definitions in which any such definition is legal—including definition (3). A definition can then be incorrect only in the sense that it doesn’t mean what its writer thought it meant. That kind of error can usually be found when the defined operator is used in a specification that is checked by a tool such as the TLC model checker.

3 Simple Recursive Definitions

We begin by considering a recursive definition of a single operator with a single argument. Such a definition has this form:

\[ F(x) \triangleq \text{Def}(x, F) \]  

(We consider multiple-argument operators and mutually recursive definitions below.) We fix \( \text{Def} \) by writing all subsequent definitions in this section in a module that begins with this declaration:

\text{CONSTANT \( \text{Def}(\_ , \_ (\_)) \)}

(As mentioned in the introduction, TLA
 doesn’t allow such a higher-order operator declaration.)

3.1 The Semantics

In 2005, the second author had the idea of letting (4) assert that \( F(x) \) equals \( g[x] \) for some function \( g \) containing \( x \) in its domain such that \( g[y] \) equals \( \text{Def}(y, g) \) for all \( y \) in its domain. This isn’t quite right, since the second argument of \( \text{Def} \) must be an operator that takes an argument. To correct it, let’s define \( \text{def}(y, g) \) to be \( \text{Def}(y, G) \) when \( G \) is the operator “obtained from” \( g \):

\[ \text{def}(y, g) \triangleq \text{Def}(y, \lambda z : g[z]) \]

\[ ^{1}\text{In TLA}^{+}, \text{this recursive operator definition must be preceded by a recursive \( F(\_ ) \) declaration. We will not bother writing those declarations.} \]
The precise statement of the idea was to let \( F(x) \) equal:

\[
\text{LET } f \overset{\Delta}{=} \text{CHOOSE } g : \land x \in \text{DOMAIN } g \\
\land g = [y \in \text{DOMAIN } g \mapsto \text{def}(y, g)] \\
\text{IN } f[x]
\]

It’s not hard to see that this definition isn’t right. For example, suppose \( \text{Def} \) were defined by:

\[
\text{Def}(x, F(\_)) \overset{\Delta}{=} \text{IF } x = 0 \text{ THEN } 1 \text{ ELSE } x \ast F(x - 1)
\]

We would expect this to define \( F(n) \) to equal \( n! \) (\( n \) factorial) for every natural number \( n \). However, let \( g \) be the function with domain \( \{3\} \) such that \( g[3] = 0 \). The semantics of TLA\(^+\) doesn’t specify what the value of \( g[y] \) is if \( y \) is not in the domain of \( g \). So, it’s possible that \( g[2] = 0 \). In that case, \( g[3] = 3 \ast g[2] \), so the body of the \text{CHOOSE} statement in (5) equals \text{true} for this function \( g \) and \( x = 3 \). Thus, it’s possible that the \text{CHOOSE} chooses this function \( g \) to equal \( f \) when \( x \) equals 3, thereby defining \( F(3) \) to equal 0. This means it’s impossible to prove that \( F(3) \) does not equal 0; and therefore it’s impossible to prove that it does equal 3! as it should.

The first author came up with a way to fix this problem. For the factorial definition (6), we need to fix the choice of \( g \) in (5) so that, for a natural number \( x \), the domain of \( g \) must include \( 0 \ldots x \). In general, we want to require that, if the recursion uniquely determines the value of \( F(x) \), then the domain of \( g \) is large enough so it “fixes” (uniquely determines) the value of \( g \) on all its elements. We first define

\[
\text{fdef}(S, g) \overset{\Delta}{=} [x \in S \mapsto \text{def}(x, g)]
\]

and then define the fixing condition to be \( \text{fix}(g) \), where

\[
\text{fix}(g) \overset{\Delta}{=} \forall h : (\forall x \in \text{DOMAIN } g : h[x] = g[x]) \Rightarrow \\
(g = \text{fdef} (\text{DOMAIN } g, h))
\]

We give a semantics to the recursive definition (4) by letting it define \( F \) to be this operator \( Fr \):

\[
Fr(x) \overset{\Delta}{=} \text{CHOOSE } g : (x \in \text{DOMAIN } g) \land \text{fix}(g) \} [x]
\]

\( ^2 \)A sophisticated TLA\(^+\) user might think that the body of the \text{CHOOSE} should also assert that \( g \) is a function, but further thought shows that’s not necessary.
3.2 Inductive Definitions

Correctness of our semantics means that

\[ Fr(x) = Def(x, Fr) \] (7)

is true for those values of \( x \) for which we expect it to be true. For example, if \( Def \) is defined by (6), then we expect (7) to be true for all \( x \) in \( Nat \). We expect it not to be true for any \( x \) when \( Def \) is defined by (3). (In this case, \( Op(x) \) equals \( \text{CHOOS} \ f : \text{FALSE} \{ x \} \) for all \( x \), but who cares?)

For \( Def \) defined by (6), we expect (7) to be true when \( x \in Nat \) because \( Def \) is inductive on \( Nat \). Intuitively, this means that it allows the value of \( F(n) \) for any \( n \in Nat \) to be computed by a finite number of applications of the definition (4). For example, we can compute \( F(27) \) by applying the definition once to see that it equals \( 27 \times F(26) \), applying it a second time to see that it equals \( 27 \times 26 \times F(25) \), and so on until we get to \( F(0) = 1 \). In general, \( Def \) is inductive on \( Nat \) iff for every \( n \in Nat \), the value of \( Def(n, F) \) depends only on the values of \( Def(i, F) \) for \( i \in 0 \ldots (n - 1) \).

We can generalize from the set \( Nat \) to any set with a well-founded order relation on the set. An (irreflexive) partial order \( \prec \) on a set \( S \) is well-founded iff there does not exist any infinite sequence \( \ldots \prec s_3 \prec s_2 \prec s_1 \) of elements of \( S \). We fix the relation \( \prec \) by making it a parameter of our module, declaring it with:

\[ \text{CONSTANT} \_ \prec \_ \]

It’s convenient to define \( LT \) so \( LT(x, S) \) is the set of elements of the set \( S \) that are \( \prec x \):

\[ LT(x, S) \triangleq \{ y \in S : y \prec x \} \]

We define \( \text{WellFounded}(S) \) as follows to mean that \( \prec \) is a well-founded partial order on \( S \):

\[ \text{WellFounded}(S) \triangleq \]

\[ \land \forall x, y, z \in S : (x \prec y) \land (y \prec z) \Rightarrow (x \prec z) \]

\[ \land \forall T \in (\text{SUBSET} S) \setminus \{ \} : \exists x \in T : LT(x, S) = \{ \} \]

Proof by mathematical induction on the set of natural numbers is generalized to the following proof rule:

\[ \text{THEOREM GeneralInduction} \triangleq \]

\[ \text{ASSUME} \quad \text{NEW} S, \ \text{WellFounded}(S), \ \text{NEW} P(\_), \]

\[ \quad \forall x \in S : (\forall y \in LT(x, S) : P(y)) \Rightarrow P(x) \]

\[ \text{PROVE} \quad \forall x \in S : P(x) \]
The natural definition of what it means for \( \text{Def} \) to be inductive on a set \( S \) with well-founded order \( \prec \) is that, for any operator \( G \), the value of \( \text{Def}(x, G) \) for any \( x \in S \) depends only on the values of \( G(y) \) for \( y \in LT(x, S) \). More precisely, define \( \text{Def} \) inductive on \( S \) to mean that the following condition holds for any operators \( G \) and \( H \):

\[
(\forall y \in LT(x, S) : G(y) = H(y)) \Rightarrow (\text{Def}(x, G) = \text{Def}(x, H))
\]

When \( \text{Def} \) is inductive on \( S \), we expect (7) to be true for all \( x \in S \).

We can’t write this definition of inductive in TLA\(^+\), since stating that (8) is true for all operators \( G \) and \( H \) mean quantifying over operators, which requires higher-order logic. However, we can write it as the following assume/prove, which can appear as the hypothesis of a theorem:

\[
\text{assume new } G, \text{ new } H \Rightarrow
\forall x \in S : (\forall y \in LT(x, S) : G(y) = H(y)) \Rightarrow (\text{Def}(x, G) = \text{Def}(x, H))
\]

We will prove that this hypothesis and \( \text{WellFounded}(S) \) imply that (7) holds for all \( x \in S \). To do that, we define \( \text{Def} \) to be contractive on \( S \) iff (8) holds when \( G \) and \( H \) are obtained from functions. The precise definition is:

\[
\text{Contractive}(S) \triangleq \forall g, h : \forall x \in S : (\forall y \in LT(x, S) : g[y] = h[y]) \Rightarrow (\text{def}(x, g) = \text{def}(x, h))
\]

3.3 Correctness for Inductive Definitions

We will show that (7) holds if \( \text{Def} \) is contractive—more precisely, that \( \text{Def} \) contractive on \( S \) implies:

\[
\forall x \in S : \text{Fr}(x) = \text{Def}(x, \text{Fr})
\]

We begin our proof with the following lemma.

**Lemma 1** \( \forall S, f, g, x : \land \text{WellFounded}(S) \land \text{Contractive}(S) \land \text{fix}(f) \land \text{fix}(g) \land x \in (\text{domain } f) \cap (\text{domain } g) \cap S \Rightarrow (f[x] = g[x]) \)

\[
\text{define } h \leftrightarrow k \triangleq [y \in (\text{domain } h) \cup (\text{domain } k) \Rightarrow (\text{if } y \in \text{domain } h \text{ then } h[y] \text{ else } k[y])]
\]
1. Assume new \( h, \) new \( k, \) fix\((h)\)

Prove \( h = \text{def}(\text{domain} \ h, \ h ++ k) \)

Proof: By the assumption fix\((h)\), since \( h[z] = (h ++ k)[z] \) for all \( z \) in domain \( h \).

2. Sufficiently assume new \( S, \) new \( f, \) new \( g, \) Contractive\((S)\), fix\((f)\), fix\((g)\)

Prove \( \forall x \in (\text{domain} \ f) \cap (\text{domain} \ g) \cap S : f[x] = g[x] \)

Proof: By simple logic.

3. Define \( Sfg \triangleq (\text{domain} \ f) \cap (\text{domain} \ g) \cap S \)

4. Sufficiently assume new \( x \in Sfg, \)\n
\( \forall y \in LT(x, Sfg) : f[y] = g[y] \)

Proof: Sfg \( \subseteq S \) implies WellFounded\((Sfg)\), so to prove \( f[x] = g[x] \), by GeneralInduction, it suffices to prove it under the assumption that \( f[y] = g[y] \) for all \( y \in LT(x, Sfg) \).

5. \( \text{def}(x, f ++ g) = \text{def}(x, g ++ f) \)

Proof: By \( 1 \)\(4 \) and Contractive\((S)\) (from \( 1 \)\(2 \)).

6. Q.E.D.

Proof: \( f[x] = \text{def}(x, f ++ g) \) [by \( 1 \)\(1 \), fix\((f)\) (from \( 1 \)\(2 \))], and \( x \in \text{domain} \ f \) (from \( 1 \)\(3 \))

\( = \text{def}(x, g ++ f) \) [by \( 1 \)\(5 \)]

\( = g[x] \) [by \( 1 \)\(1 \), fix\((g)\) (from \( 1 \)\(2 \)), and \( x \in \text{domain} \ g \) (from \( 1 \)\(3 \))]

We now define \( fr(S) \) to be the function with domain \( S \) that agrees with \( Fr \) on that set:

\( fr(S) \triangleq [x \in S \mapsto Fr(x)] \)
The next theorem shows that if $Def$ is contractive on $S$, then $fr(S)$ equals $fdef(S, fr(S))$, and this equality uniquely determines the function $fr(S)$.

**Theorem 1** \[ \forall S : WellFounded(S) \land Contractive(S) \Rightarrow (\forall f : (f = fdef(S, f)) \equiv (f = fr(S))) \]

(1) Assume new $S$, $WellFounded(S)$, $Contractive(S)$,
new $f$, $f = fdef(S, f)$  
prove $fix(f) \land (\text{domain } f = S)$

(2) 1. Domain $f = S$
   Proof: By the (1) 1 assumption and the definition of $fdef$.

(2) 2. Suffices assume new $g$, $\forall x \in S : g[x] = f[x]$  
   prove $fdef(S, g) = f$
   Proof: By (2) 1 it suffices to prove $fix(f)$, which by (2) 1 and the definition of $fix$ means proving that the (2) 2 assume implies $f = fdef(S, g)$.

(2) 3. $\forall x \in S : \forall y \in LT(x, S) : g[x] = f[x]$
   Proof: By the (2) 2 assumption, since $LT(x, S) \subseteq S$ by definition of $LT$.

(2) 4. Q.E.D.
   Proof: (2) 3 and the assumption $Contractive(S)$ (from (1) 1) imply $\forall x \in S : def(x, g) = def(x, f)$, which by definition of $fdef$ implies $fdef(S, g) = fdef(S, f)$. By the (1) 1 assumption, this is equivalent to the current goal.

(1) 2. Suffices assume new $S$, $WellFounded(S)$, $Contractive(S)$  
prove $fr(S) = fdef(S, fr(S))$

(2) 1. Assume new $S$, $WellFounded(S)$, $Contractive(S)$,
$fr(S) = fdef(S, fr(S))$,  
new $f$, $f = fdef(S, f)$  
prove $f = fr(S)$
   Proof: (1) 1 and the assumptions imply $fix(f)$, $fix(fr(S))$, and both domain $f$ and domain $fr(S)$ equal $S$. By Lemma 1, this implies $f = fr(S)$.

(2) 2. Assume new $S$, $WellFounded(S)$, $Contractive(S)$,
$fr(S) = fdef(S, fr(S))$,  
new $f$, $f = fr(S)$  
prove $f = fdef(S, f)$
Proof: The conclusion follows immediately from the hypotheses

\[ fr(S) = fdef(S, fr(S)) \] and \( f = fr(S) \).

\( \langle 2 \rangle \). Q.E.D.

Proof: By simple logic, \( \langle 2 \rangle 1 \) and \( \langle 2 \rangle 2 \) show that the assume/prove of \( \langle 1 \rangle 2 \) implies the theorem.

\( \langle 1 \rangle 3. \) Suffices Assume

\[ \text{new } x \in S, \quad \forall y \in LT(x, S) : Fr(y) = def(y, fr(S)) \]

Prove \( Fr(x) = def(x, fr(S)) \)

\( \langle 2 \rangle 1. \) Assume \( \forall x \in S : \)

\[ (\forall y \in LT(x, S) : Fr(y) = def(y, fr(S))) \]

\[ \Rightarrow (Fr(x) = def(x, fr(S))) \]

Prove \( fr(S) = fdef(S, fr(S)) \)

\( \langle 3 \rangle 1. \) \( \forall x \in S : Fr(x) = def(x, fr(S)) \)

Proof: By \( \text{WellFounded}(S) \) (by \( \langle 1 \rangle 2 \)), the \( \langle 2 \rangle 1 \) assumption and \emph{GeneralInduction}.

\( \langle 3 \rangle 2. \) Q.E.D.

Proof: For all \( x \in S : \)

\[ fr(S) = [x \in S \mapsto Fr(x)] \quad \quad \text{[by definition of } fr(S) \text{]} \]

\[ = [x \in S \mapsto def(x, fr(S))] \quad \quad \text{[by \( \langle 3 \rangle 1 \)]} \]

\[ = fdef(S, fr(S)) \quad \quad \text{[by definition of } fdef \text{]} \]

\( \langle 2 \rangle 2. \) Q.E.D.

Proof: By \( \langle 2 \rangle 1 \), since its assume formula is equivalent to the assume/prove of \( \langle 1 \rangle 3 \), and its prove formula is the goal introduced by \( \langle 1 \rangle 2 \).

\( \langle 1 \rangle \) Define \( LE(x, T) \triangleq \{ y \in T : (y < x) \lor (y = x) \} \)

\( \quad \text{fx } \triangleq \text{fdef}(LE(x, S), fr(S)) \)

\( \langle 1 \rangle 4. \) \( f = fdef(LE(x, S), fx) \)

\( \langle 2 \rangle 1. \) Suffices Assume \text{new } y \in LE(x, S)

Prove \( \text{def}(y, fr(S)) = def(y, fx) \)

Proof: By the definitions of \( fx \) and \( fdef \).

\( \langle 2 \rangle 2. \) Suffices Assume \text{new } z \in LT(y, S)

Prove \( fr(S)[z] = fr[z] \)

Proof: \emph{Contractive}(S) (assumed in \( \langle 1 \rangle 2 \)) implies that \( \langle 2 \rangle 2 \) implies the current goal (introduced by \( \langle 2 \rangle 1 \)).
3. \( z \in S \land z \in LT(x, S) \land z \in LE(x, S) \)

**Proof:** By (2)1, (2)2, the definitions of \( LE \) and \( LT \), and the transitivity of \( \prec \) (implied by \( \text{WellFounded}(S) \)).

4. Q.E.D.

\[
\text{Proof: }12\text{fr}(S)[z] = Fr(z) \quad \text{[definition of fr and (2)3]}
\]
\[
= \text{def}(z, fr(S)) \quad \text{[(1)3 and (2)3]}
\]
\[
= fx[z] \quad \text{[definitions of fx and fdef, and (2)3]}
\]

5. \( \text{WellFounded}(LE(x, S)) \land \text{Contractive}(LE(x, S)) \)

**Proof:** (1)2 implies \( \text{WellFounded}(S) \) and \( \text{Contractive}(S) \). The definition of \( LE \) implies \( LE(x, S) \subseteq S \). That and \( \text{WellFounded}(S) \) imply \( \text{WellFounded}(LE(x, S)) \). From \( x \in S \) and the transitivity of \( \prec \) on \( S \) (by \( \text{WellFounded}(S) \)), we have \( LT(y, LE(x, S)) = LT(y, S) \) for all \( y \in LE(x, S) \). This and \( \text{Contractive}(S) \) imply \( \text{Contractive}(LE(x, S)) \).

6. \( \text{fix}(fx) \land (x \in \text{DOMAIN } fx) \)

**Proof:** By (1)5, (1)1 (with \( LE(x, S) \) substituted for \( S \)), and (1)4.

\[
\text{Define } gx \triangleq \text{choose } g : (x \in \text{DOMAIN } g) \land \text{fix}(g)
\]

7. \( (x \in \text{DOMAIN } gx) \land \text{fix}(gx) \)

**Proof:** By (1)6 the defining property of \( gx \) is satisfied by \( fx \).

8. Q.E.D.

\[
\text{Proof: } Fr(x) = gx[x] \quad \text{[definitions of Fr and gx]}
\]
\[
= fx[x] \quad \text{[Lemma 1, (1)6, (1)7, and (1)3]}
\]
\[
= \text{def}(x, fr(S)) \quad \text{[definition of fx and (1)7]}
\]

Theorem 1 has the following corollary:

**Corollary 1**  \( \forall S : \text{WellFounded}(S) \land \text{Contractive}(S) \Rightarrow \forall x \in S : Fr(x) = \text{def}(x, fr(S)) \)

**Proof:** Theorem 1 implies \( fr(S) = fdef(S, fr(S)) \), and the result then follows from the definitions of \( fr \) and \( fdef \).

The conclusion of Corollary 1 is very close to our goal, which is proving (10); but it doesn’t imply that goal. In fact, the following example shows that \( \text{Contractive}(S) \) is too weak a condition to imply (10). Define

\[
\text{fromFcn}(G(\_)) \triangleq \exists g : \forall x : G(x) = g[x]
\]
and suppose \( \text{Def} \) is defined by

\[
\text{Def}(x, G) \overset{\Delta}{=} \text{if fromFcn}(G) \text{ then 1 else 0}
\]

For any function \( f \) and any \( x \in \text{domain} f \), the definition of \( \text{def} \) implies \( \text{def}(x, f) = 1 \) for this operator \( \text{Def} \). This implies \( \text{Def} \) is contractive on any partially ordered set \( S \). For any \( x \), let \( f_x \) be the function \( [y \in \{x\} \mapsto 1] \). It’s easy to see that \( \text{fix}(f_x) \) is true, and that this implies \( \text{Fr}(x) = 1 \) for all \( x \). However, for a function \( f \), the semantics of TLA+ says nothing about the value of \( f[y] \) for \( y \notin \text{domain} f \). Therefore, there may be no function \( g \) such that \( \forall x : g[x] = 1 \) is true. In that case, \( \text{fromFcn}(\text{Fr}) \) equals \( \text{false} \), so \( \text{Def}(x, \text{Fr}) \) equals 0 for all \( x \), and therefore \( \text{Fr}(x) \neq \text{Def}(x, \text{Fr}) \) for all \( x \).

The reason we can’t derive (10) from the hypothesis that \( \text{Def} \) is contractive on \( S \) is that this hypothesis still allows the value of \( \text{Def}(x, \text{Fr}) \) to depend on values of \( \text{Fr}(y) \) for \( y \notin S \) even though \( x \in S \). This possibility is effectively ruled out by the condition that \( \text{Fr} \) is \( \text{representable} \) on \( S \), where representable is defined by:

\[
\text{Representable}(G(\_), S) \overset{\Delta}{=} \forall x \in S : \text{Def}(x, G) = \text{def}(x, [y \in S \mapsto G(y)])
\]

The following theorem shows that \( \text{Def} \) contractive on \( S \) and \( \text{Fr} \) representable on \( S \) implies (10), and that any operator satisfying the recurrence condition equals \( \text{Fr} \) on \( S \).

**Theorem 2**

\[
\text{assume new } S, \text{ WellFounded}(S), \text{ Contractive}(S), \text{ new } G(\_), \text{ Representable}(G, S) \text{ prove } (\forall x \in S : G(x) = \text{Def}(x, G)) \equiv (\forall x \in S : G(x) = \text{Fr}(x))
\]

**Proof:** Let \( f \overset{\Delta}{=} [x \in S \mapsto G(x)] \). Then:

\[
(\forall x \in S : G(x) = \text{Def}(x, G)) \equiv (f = \text{fdef}(S, f)) \quad \text{[definitions of } f \text{ and } \text{fdef}, \text{ and the } \text{Representable}(G, S) \text{ assumption]}
\equiv (f = \text{fr}(S)) \quad \text{[Theorem 1]}
\equiv (\forall x \in S : G(x) = \text{Fr}(x)) \quad \text{[definitions of } f \text{ and } \text{fr}(S)]
\]

The following corollary to Theorem 2 asserts that \( \text{Def} \) contractive and representable on \( S \) implies (10).

**Corollary 2**

\[
\forall S : \text{WellFounded}(S) \land \text{Contractive}(S) \land \text{Representable}(\text{Fr}) \Rightarrow (\forall x \in S : \text{Fr}(x) = \text{Def}(x, \text{Fr}))
\]
**Proof:** By Theorem 2 applied to $Fr$.

Our goal is to prove that $WellFounded(S)$ and $Def$ inductive on $S$ imply (10). We have proved that $WellFounded(S)$ and $Def$ implies (10). To complete the proof of our goal, we have to show that $WellFounded(S)$ and $Def$ inductive on $S$ imply $Def$ is contractive and representable on $S$. Since $Def$ inductive on $S$ is expressed by (9), this is done by the following theorem.

**Theorem 3**

**Assume** new $S$, $WellFounded(S)$, new $G(\_)$, new $H(\_)$, new $J(\_)$

**Prove** $\forall x \in S : (\forall y \in LT(x, S) : H(y) = J(y)) \implies (Def(x, H) = Def(x, J))$

**Proof:** Apply the theorem’s assumption with $H(y) \triangleq g[y]$ and $J(y) \triangleq h[y]$.

**1. Representable($G, S$)**

**Proof:** Define $h \triangleq [y \in S \mapsto G(y)]$ .

$H(x) \triangleq h[x]

By definition of Representable, it suffices to assume $x \in S$ and prove $Def(x, G) = def(x, h)$, which is done as follows:

$Def(x, G) = Def(x, H)$ [By the assume/prove assumption]

$= def(x, h)$ [By definition of $def$]

**Q.E.D.**

By (1)1, which is the definition of Contractive($S$), and (1)2.

Our goal is now a simple corollary of Corollary 2 and Theorem 3.

**Corollary 3**

**Assume** new $S$, $WellFounded(S)$, new $G(\_)$, new $H(\_)$

**Prove** $\forall x \in S : (\forall y \in LT(x, S) : G(y) = H(y)) \implies (Def(x, G) = Def(x, H))$

**Prove** $\forall x \in S : Fr(x) = Def(x, Fr)$

**Proof:** By Theorem 3, substituting $Fr$ for $G$, and Corollary 2.
In practice, for any \( v \), the value of \( \text{Def}(v, F) \) is defined in terms of \( v \) and \( F(v_1), \ldots, F(v_k) \) for some finite set \( \{v_1, \ldots, v_k\} \). There is then an obvious recursive algorithm for computing \( F(v) \). Our results imply that if this algorithm terminates, then it computes \( F(v) = \text{Fr}(v) \). To prove this, let \( S \) equal the set of all values \( x \) for which the algorithm computes \( F(x) \), and define the relation \( \prec \) on \( S \) so \( x \prec y \) is true iff the algorithm computes \( F(x) \) when computing \( F(y) \). Termination of the algorithm implies that \( \prec \) is an irreflexive partial order on \( S \) and that \( S \) is finite, so \( \prec \) is well-founded on \( S \).

Let \( G(x) \) be the value computed by the algorithm for all \( x \in S \). Theorem 3 implies \( \text{Representable}(G, S) \) and Theorem 2 then implies \( G(v) = \text{Fr}(v) \).

Our theorems and corollaries cannot be expressed in TLA\(^+\) because \( \text{Def} \) needs to be declared as \text{CONSTANT}, and TLA\(^+\) does not support declarations of operators with an operator argument. To state these results in a more easy to use way, we would write them as \text{ASSUME} / \text{PROOF} statements with \( \text{Def} \) declared in a \text{NEW} clause, which TLA\(^+\) also does not permit. Our results should be provable now with TLAPS by writing an arbitrary definition of \( \text{Def} \) and proving them without using that definition.

4 Multiple Arguments

TLA\(^+\) permits recursive definitions of operators that take multiple arguments. We must therefore assign a meaning to this definition, for all \( n \in \text{Nat} \):

\[
F(x_1, \ldots, x_n) \triangleq \text{Def}(x_1, \ldots, x_n, F) \tag{11}
\]

where \( F \) is declared by

\[
\text{CONSTANT} \ F(-, \ldots, -, -(-, \ldots, -))
\]

For \( n = 0 \), in which (11) is \( F \triangleq \text{Def}(F) \), its obvious meaning is:

\[
F \triangleq \text{CHOOSE } G : G = \text{Def}(G)
\]

We’ve already defined (11) for \( n = 1 \). For \( n > 1 \), we define \( F(x_1, \ldots, x_n) \) to equal \( G((x_1, \ldots, x_n)) \) for an operator \( G \) that has a single argument. To simplify things, we introduce some notation. Let \( x \) stand for \( x_1, \ldots, x_n \), so we can therefore write (11) as

\[
F(x) \triangleq \text{Def}(x, F)
\]
We will define the operator $G$ such that $F(x)$ equals $G((x))$. For any expression $z$ and any operator $H$ of a single argument, we define
\[ n^z \triangleq z[1], \ldots, z[n] \]
\[ n^H(x) \triangleq H((x)) \]
We define $G$ by the recursive definition
\[ (12) \quad G(z) \triangleq Def_G(z, G) \]
where $Def_G$ is defined by
\[ (13) \quad Def_G(z, H) \triangleq Def(n^z, n^H) \]
Let $fix_G$ and $Fr_G$ be the operators $fix$ and $Fr$ defined in Section 3, when $Def_G$ is substituted for $Def$. In that section we defined $G$ to be this operator:
\[ Fr_G(x) \triangleq (\text{choose } g : (x \in \text{domain } g) \land fix_G(g)) [x] \]
Expanding definitions, we obtain:
\[ fix_G(g) \equiv \forall h : (\forall x \in \text{domain } g : h[x] = g[x]) \Rightarrow (g = [x \in \text{domain } g \mapsto Def(n^x, n^\lambda y : g[y])]) \]
We then define $F$ to equal $n^{Fr_G}$. Applying Corollary 3 to $Def_G$ and $Fr_G$ and expanding definitions, we get this result:
\[ (14) \quad \text{assume new } S, \text{ WellFounded}(S), \text{ new } G(-, \ldots, -), \text{ new } H(-, \ldots, -) \]
\[ \text{prove } \forall x \in S : (\forall y \in LT(x, S) : G(y) = H(y)) \Rightarrow (Def(n^x, G) = Def(n^x, H)) \]
\[ \text{prove } \forall \langle x \rangle \in S : F(x) = Def(x, F) \]
Like Corollary 3 of Section 3, the theorem (14) is the correctness condition for the meaning we assign to the definition (11).
We cannot write (11) in TLA\textsuperscript{+} because we cannot express “\ldots”, as in $G(-, \ldots, -)$. (Neither can we write the definition (14).) We can view (14) as a collection of theorems, one for each number $n$. A TLAPS library file could contain perhaps the first dozen of those theorems. Alternatively, instead of defining $F$ by (11), we can first define $G$ by (12) and (13) and then define $F(x_1, \ldots, x_n)$ to equal $G((x_1, \ldots, x_n))$. We can then deduce the desired property of $F$ by applying Corollary 3 to $G$.  

15
5 Mutual Recursion

In TLA\(^+\), an operator not declared in a recursive statement cannot be used before (or in) its definition. In defining the meaning of a module, all occurrences of such an operator can be eliminated by expanding the operator’s definition. What remains is a sequence of sets of definitions of recursive operator definitions, each set having this form for some \( k \):

\[
F_1(x_1, \ldots, x_{n_1}) \triangleq \text{Def}_1(x_1, \ldots, x_{n_1}, F_1, \ldots, F_k) \\
\vdots \\
F_k(x_1, \ldots, x_{n_k}) \triangleq \text{Def}_k(x_1, \ldots, x_{n_k}, F_1, \ldots, F_k)
\]

(Each \( \text{Def}_i \) need not actually depend on all the \( F_i \).) For \( k > 1 \), this is called a set of mutually recursive definitions. We define the meaning of (15) in terms of a recursive definition of a single operator \( G \) taking a single argument by:

\[
F_i(x_1, \ldots, x_{n_i}) \triangleq G(\langle i, \langle x_1, \ldots, x_{n_i} \rangle \rangle)
\]

We define \( G \) by this recursive definition of the form (4):

\[
G(z) \triangleq \text{Def}_G(z, G)
\]

with \( \text{Def}_G \) defined by

\[
\text{case } z[1] = 1 \rightarrow \text{Def}_1(\langle z[2][1], \ldots, z[2][n_1], \hat{H}_1, \ldots, \hat{H}_k \rangle) \\
\vdots \\
\square z[1] = k \rightarrow \text{Def}_k(\langle z[2][1], \ldots, z[2][n_k], \hat{H}_1, \ldots, \hat{H}_k \rangle)
\]

where \( \hat{H}_i(x_1, \ldots, x_{n_i}) \triangleq H(i, \langle x_1, \ldots, x_{n_i} \rangle) \)

Just as we obtained (14) for the case \( k = 1 \), we can apply Corollary 3 to \( G \) and expand definitions to get this result:

\[
\text{assume } \text{new } S, \text{ WellFounded}(S), \text{ new } H(\_), \text{ new } J(\_)
\]

\[
\text{prove } \forall x \in S : (\forall y \in LT(x, S) : H(y) = J(y)) \\
\Rightarrow (\text{Def}_G(x, H) = \text{Def}_G(x, J))
\]

\[
\text{prove } \forall \langle i, \langle x_1, \ldots, x_{n_i} \rangle \rangle \in S : \\
(i \in 1 \ldots k) \Rightarrow F_i(x_1, \ldots, x_{n_i}) = \text{Def}_i(x_1, \ldots, x_{n_i}, F_1, \ldots, F_k)
\]
where (18) defines $\text{Def}_G$ in terms of the $F_i$.

As with (14), formula (19) is not expressible in TLA+. It is a collection of formulas, one for each choice of the numbers $k, n_1, \ldots, n_k$. Unlike the $k = 1$ case, writing these as separate theorems in a TLAPS library file does not seem feasible. We can use the alternative approach of not writing (15), but instead first defining $G$ by (17) and (18), and then defining the $F_i$ by (16).

References
