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

$$(1) \quad f[n \in Nat] \triangleq \text{IF } n = 0 \text{ THEN } 1 \text{ ELSE } n * f[n-1]$$

is an abbreviation for:

$$(2) \quad f \triangleq \text{CHOOSE } g : g = [n \in Nat \mapsto \text{IF } n = 0 \text{ THEN } 1 \text{ ELSE } n * g[n-1]]$$

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 $Cardinality(S)$ is the number of elements in a finite set S:

$$\begin{array}{l} Cardinality(S) \triangleq \\ \quad \text{IF } S = \{\} \text{ THEN } 0 \text{ ELSE } 1 + Cardinality(S \setminus \{\text{CHOOSE } x : x \in S\}) \end{array}$$

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

$$\begin{array}{l} Cardinality(S) \triangleq \\ \quad \text{LET } f[T \in \text{SUBSET } S] \triangleq \\ \qquad\quad \text{IF } T = \{\} \text{ THEN } 0 \text{ ELSE } 1 + f[T \setminus \{\text{CHOOSE } x : x \in T\}] \\ \quad \text{IN } \ f[S] \end{array}$$

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:

(3) $\quad Op(x) \;\triangleq\; \text{CHOOSE } y : y \neq Op(x)$

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:

$$\text{THEOREM } \forall n \in Nat : f[n] = \text{IF } n = 0 \text{ THEN } 1 \text{ ELSE } n * f[n-1]$$

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:[1]

$$(4) \quad F(x) \triangleq Def(x, F)$$

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

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

(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 $Def(y, g)$ for all y in its domain. This isn't quite right, since the second argument of *Def* must be an operator that takes an argument. To correct it, let's define $def(y, g)$ to be $Def(y, G)$ when G is the operator "obtained from" g:

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

[1] In TLA$^+$, 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:[2]

$$(5)\quad \text{LET } f \triangleq \text{CHOOSE } g : \begin{array}{l} \land\ x \in \text{DOMAIN } g \\ \land\ g = [y \in \text{DOMAIN } g \mapsto \mathit{def}(y, g)] \end{array} \\ \text{IN } f[x]$$

It's not hard to see that this definition isn't right. For example, suppose *Def* were defined by:

$$(6)\quad \mathit{Def}(x, F(\_)) \triangleq \text{IF } x = 0 \text{ THEN } 1 \text{ ELSE } x * F(x-1)$$

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, $g[3] = 3 * g[2]$, so the body of the CHOOSE statement in (5) equals TRUE for this function g and $x = 3$. Thus, it's possible that the 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 $0 \,..\, x$. 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

$$\mathit{fdef}(S, g) \triangleq [x \in S \mapsto \mathit{def}(x, g)]$$

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

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

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

$$Fr(x) \triangleq (\text{CHOOSE } g : (x \in \text{DOMAIN } g) \land \mathit{fix}(g))[x]$$

[2] A sophisticated TLA$^+$ user might think that the body of the 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

(7) $\quad Fr(x) = Def(x, Fr)$

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 (CHOOSE f : 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 $27 * F(26)$, applying it a second time to see that it equals $27 * 26 * F(25)$, 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 $i \in 0 \,..\, (n-1)$.

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 $WellFounded(S)$ as follows to mean that $\prec$ is a well-founded partial order on S:

$$\begin{array}{l} WellFounded(S) \;\triangleq \\ \quad \land\ \forall x, y, z \in S : (x \prec y) \land (y \prec z) \Rightarrow (x \prec z) \\ \quad \land\ \forall T \in (\text{SUBSET } S) \setminus \{\,\} : \exists x \in T : LT(x, S) = \{\,\} \end{array}$$

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

$$\begin{array}{lll} \text{THEOREM } GeneralInduction \;\triangleq & & \\ \quad \text{ASSUME} & \text{NEW } S,\ WellFounded(S),\ \text{NEW } P(\_) & \\ & \forall x \in S : (\forall y \in LT(x, S) : P(y)) \Rightarrow P(x) & \\ \quad \text{PROVE} & \forall x \in S : P(x) & \end{array}$$

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

(8) $\quad (\forall\, y \in LT(x, S) \,:\, G(y) = H(y)) \;\Rightarrow\; (Def(x, G) = Def(x, H))$

When 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:

(9) $$\begin{array}{ll} \textsc{assume} & \textsc{new}\ G(\_),\ \textsc{new}\ H(\_) \\ \textsc{prove} & \forall\, x \in S \,:\, (\forall\, y \in LT(x, S) \,:\, G(y) = H(y)) \\ & \qquad\qquad \Rightarrow\ (Def(x, G) = Def(x, H)) \end{array}$$

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

$$\begin{array}{l} Contractive(S) \;\triangleq \\ \quad \forall\, g, h \,:\, \forall\, x \in S \,: \\ \qquad (\forall\, y \in LT(x, S) \,:\, g[y] = h[y]) \;\Rightarrow\; (def(x, g) = def(x, h)) \end{array}$$

3.3 Correctness for Inductive Definitions

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

(10) $\quad \forall\, x \in S \,:\, Fr(x) = Def(x, Fr)$

We begin our proof with the following lemma.

Lemma 1 $$\begin{array}{ll} \forall\, S, f, g, x \,: & \land\ WellFounded(S) \\ & \land\ Contractive(S) \\ & \land\ fix(f) \land fix(g) \\ & \land\ x \in (\textsc{domain}\ f) \cap (\textsc{domain}\ g) \cap S \\ & \Rightarrow (f[x] = g[x]) \end{array}$$

$$\langle 1 \rangle\ \textsc{define}\ h +\!\!+ k \;\triangleq\; [y \in (\textsc{domain}\ h) \cup (\textsc{domain}\ k) \mapsto \textsc{if}\ \ y \in \textsc{domain}\ h\ \ \textsc{then}\ \ h[y]\ \ \textsc{else}\ \ k[y]\,]$$

$\langle 1 \rangle 1.$ ASSUME NEW h, NEW k, $\mathit{fix}(h)$
PROVE $h = \mathit{fdef}(\text{DOMAIN } h,\ h +\!\!+ k)$

PROOF: By the assumption $\mathit{fix}(h)$, since $h[z] = (h +\!\!+ k)[z]$ for all z in DOMAIN h.

$\langle 1 \rangle 2.$ SUFFICES ASSUME NEW S, NEW f, NEW g,
$\mathit{Contractive}(S)$, $\mathit{fix}(f)$, $\mathit{fix}(g)$
PROVE $\forall x \in (\text{DOMAIN } f) \cap (\text{DOMAIN } g) \cap S : f[x] = g[x]$

PROOF: By simple logic.

$\langle 1 \rangle$ DEFINE $\mathit{Sfg} \triangleq (\text{DOMAIN } f) \cap (\text{DOMAIN } g) \cap S$

$\langle 1 \rangle 3.$ SUFFICES ASSUME NEW $x \in \mathit{Sfg}$,
$\forall y \in \mathit{LT}(x, \mathit{Sfg}) : f[y] = g[y]$
PROVE $f[x] = g[x]$

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

$\langle 1 \rangle 4.$ $\forall y \in \mathit{LT}(x, S) : (f +\!\!+ g)[y] = (g +\!\!+ f)[y]$

PROOF: Since $\mathit{LT}(x, S) \subseteq S$, if y is in $(\text{DOMAIN } f) \cap (\text{DOMAIN } g)$ then it is also in Sfg, so $\langle 1 \rangle 3$ implies $(f +\!\!+ g)[y] = (g +\!\!+ f)[y]$. If y is not in $(\text{DOMAIN } f) \cap (\text{DOMAIN } g)$, then the definition of $+\!\!+$ implies $(f +\!\!+ g)[y] = (g +\!\!+ f)[y]$.

$\langle 1 \rangle 5.$ $\mathit{def}(x, f +\!\!+ g) = \mathit{def}(x, g +\!\!+ f)$

PROOF: By $\langle 1 \rangle 4$ and $\mathit{Contractive}(S)$ (from $\langle 1 \rangle 2$).

$\langle 1 \rangle 6.$ Q.E.D.

PROOF: $f[x] = \mathit{def}(x, f +\!\!+ g)$ [by $\langle 1 \rangle 1$, $\mathit{fix}(f)$ (from $\langle 1 \rangle 2$), and $x \in \text{DOMAIN } f$ (from $\langle 1 \rangle 3$)]
$= \mathit{def}(x, g +\!\!+ f)$ [by $\langle 1 \rangle 5$]
$= g[x]$ [by $\langle 1 \rangle 1$, $\mathit{fix}(g)$ (from $\langle 1 \rangle 2$), and $x \in \text{DOMAIN } g$ (from $\langle 1 \rangle 3$)]

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

$$\mathit{fr}(S) \triangleq [x \in S \mapsto \mathit{Fr}(x)]$$

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⟩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 $f = fr(S)$.

⟨2⟩3. Q.E.D.

PROOF: By simple logic, ⟨2⟩1 and ⟨2⟩2 show that the ASSUME/PROVE of ⟨1⟩2 implies the theorem.

⟨1⟩3. SUFFICES ASSUME NEW $x \in S$,
$\forall\, y \in LT(x, S) : Fr(y) = def(y, fr(S))$
PROVE $Fr(x) = def(x, fr(S))$

⟨2⟩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 $fr(S) = fdef(S, fr(S))$

⟨3⟩1. $\forall\, x \in S : Fr(x) = def(x, fr(S))$

PROOF: By $WellFounded(S)$ (by ⟨1⟩2), the ⟨2⟩1 assumption and $GeneralInduction$.

⟨3⟩2. Q.E.D.

PROOF: For all $x \in S$:

$$\begin{aligned} fr(S) &= [x \in S \mapsto Fr(x)] && \text{[by definition of } fr(S)\text{]} \\ &= [x \in S \mapsto def(x, fr(S))] && \text{[by ⟨3⟩1]} \\ &= fdef(S, fr(S)) && \text{[by definition of } fdef\text{]} \end{aligned}$$

⟨2⟩2. Q.E.D.

PROOF: By ⟨2⟩1, since its ASSUME formula is equivalent to the ASSUME/PROVE of ⟨1⟩3, and its PROVE formula is the goal introduced by ⟨1⟩2.

⟨1⟩ DEFINE $LE(x, T) \triangleq \{y \in T : (y \prec x) \lor (y = x)\}$
$fx \triangleq fdef(LE(x, S), fr(S))$

⟨1⟩4. $fx = fdef(LE(x, S), fx)$

⟨2⟩1. SUFFICES ASSUME NEW $y \in LE(x, S)$
PROVE $def(y, fr(S)) = def(y, fx)$

PROOF: By the definitions of fx and $fdef$.

⟨2⟩2. SUFFICES ASSUME NEW $z \in LT(y, S)$
PROVE $fr(S)[z] = fx[z]$

PROOF: $Contractive(S)$ (assumed in ⟨1⟩2) implies that ⟨2⟩2 implies the current goal (introduced by ⟨2⟩1).

$\langle 2\rangle 3.\ z \in S \wedge z \in LT(x, S) \wedge z \in LE(x, S)$

PROOF: By $\langle 2\rangle 1$, $\langle 2\rangle 2$, the definitions of LE and LT, and the transitivity of $\prec$ (implied by $WellFounded(S)$).

$\langle 2\rangle 4.$ Q.E.D.

PROOF:
$$\begin{aligned} fr(S)[z] &= Fr(z) && \text{[definition of } fr \text{ and } \langle 2\rangle 3]\\ &= def(z, fr(S)) && [\langle 1\rangle 3 \text{ and } \langle 2\rangle 3]\\ &= fx[z] && \text{[definitions of } fx \text{ and } fdef\text{, and } \langle 2\rangle 3]\end{aligned}$$

$\langle 1\rangle 5.\ WellFounded(LE(x, S)) \wedge Contractive(LE(x, S))$

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

$\langle 1\rangle 6.\ fix(fx) \wedge (x \in \text{DOMAIN } fx)$

PROOF: By $\langle 1\rangle 5$, $\langle 1\rangle 1$ (with $LE(x, S)$ substituted for S), and $\langle 1\rangle 4$.

$\langle 1\rangle$ DEFINE $gx \triangleq \text{CHOOSE } g : (x \in \text{DOMAIN } g) \wedge fix(g)$

$\langle 1\rangle 7.\ (x \in \text{DOMAIN } gx) \wedge fix(gx)$

PROOF: By $\langle 1\rangle 6$ the defining property of gx is satisfied by fx.

$\langle 1\rangle 8.$ Q.E.D.

PROOF:
$$\begin{aligned} Fr(x) &= gx[x] && \text{[definitions of } Fr \text{ and } gx]\\ &= fx[x] && [\text{Lemma 1, } \langle 1\rangle 6, \langle 1\rangle 7, \text{ and } \langle 1\rangle 3]\\ &= def(x, fr(S)) && \text{[definition of } fx \text{ and } \langle 1\rangle 7]\end{aligned}$$

Theorem 1 has the following corollary:

Corollary 1 $\forall S : WellFounded(S) \wedge Contractive(S) \Rightarrow$
$$\forall x \in S : Fr(x) = def(x, fr(S))$$

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 $Contractive(S)$ is too weak a condition to imply (10). Define

$$fromFcn(G(\_)) \triangleq \exists g : \forall x : G(x) = g[x]$$

and suppose *Def* is defined by

$$Def(x, G) \triangleq \text{IF } fromFcn(G) \text{ THEN } 1 \text{ ELSE } 0$$

For any function f and any $x \in \text{DOMAIN } f$, the definition of *def* implies $def(x, f) = 1$ for this operator *Def*. This implies *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 $fix(f_x)$ is true, and that this implies $Fr(x) = 1$ 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, $fromFcn(Fr)$ equals FALSE, so $Def(x, Fr)$ equals 0 for all x, and therefore $Fr(x) \neq Def(x, Fr)$ for all x.

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

$$\begin{aligned} &Representable(G(\_), S) \triangleq \\ &\quad \forall x \in S : Def(x, G) = def(x, [y \in S \mapsto G(y)]) \end{aligned}$$

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

Theorem 2

$$\begin{aligned} &\text{ASSUME } \text{NEW } S,\ WellFounded(S),\ Contractive(S), \\ &\qquad\quad\ \text{NEW } G(\_),\ Representable(G, S) \\ &\text{PROVE } \ (\forall x \in S : G(x) = Def(x, G)) \ \equiv\ (\forall x \in S : G(x) = Fr(x)) \end{aligned}$$

PROOF: Let $f \triangleq [x \in S \mapsto G(x)]$. Then:

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

The following corollary to Theorem 2 asserts that *Def* contractive and representable on S implies (10).

Corollary 2

$$\begin{aligned} &\forall S : WellFounded(S) \wedge Contractive(S) \wedge Representable(Fr) \\ &\qquad\qquad \Rightarrow (\forall x \in S : Fr(x) = Def(x, Fr)) \end{aligned}$$

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 $G(\_)$,
ASSUME NEW $H(\_)$, NEW $J(\_)$
PROVE $\forall x \in S : (\forall y \in LT(x, S) : H(y) = J(y)) \Rightarrow (Def(x, H) = Def(x, J))$
PROVE $Contractive(S) \wedge Representable(G, S)$

⟨1⟩1. ASSUME NEW g, NEW h, NEW $x \in S$, $(\forall y \in LT(x, S) : g[y] = h[y])$
PROVE $def(x, g) = def(x, h)$

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

⟨1⟩2. $Representable(G, S)$

PROOF: Define
$$h \triangleq [y \in S \mapsto G(y)] \quad . \qquad H(x) \triangleq h[x]$$

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

$$\begin{aligned} Def(x, G) &= Def(x, H) && \text{[By the ASSUME/PROVE assumption]} \\ &= def(x, h) && \text{[By definition of } def\text{]} \end{aligned}$$

⟨1⟩3. 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, $WellFounded(S)$,
ASSUME NEW $G(\_)$, NEW $H(\_)$
PROVE $\forall x \in S : (\forall y \in LT(x, S) : G(y) = H(y)) \Rightarrow (Def(x, G) = Def(x, H))$
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 $Def(v, F)$ 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 $F(v)$ equal to $Fr(v)$. 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 $Representable(G, S)$ and Theorem 2 then implies $G(v) = Fr(v)$.

Our theorems and corollaries cannot be expressed in TLA$^+$ because Def needs to be declared as 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 ASSUME /PROOF statements with Def declared in a NEW clause, which TLA$^+$ also does not permit. Our results should be provable now with TLAPS by writing an arbitrary definition of 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 $n \in Nat$:

(11) $F(x_1, \ldots, x_n) \;\triangleq\; Def(x_1, \ldots, x_n, F)$

where F is declared by

$$\text{CONSTANT } F(\_, \ldots, \_, \_(\_, \ldots, \_))$$

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

$$F \;\triangleq\; \text{CHOOSE } G : G = Def(G)$$

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

$$F(\mathbf{x}) \;\triangleq\; Def(\mathbf{x}, F)$$

We will define the operator G such that $F(\mathbf{x})$ equals $G(\langle \mathbf{x} \rangle)$. For any expression z and any operator H of a single argument, we define

$$
\begin{aligned}
{}_{n}z &\triangleq z[1], \ldots z[n] \\
{}^{n}H(\mathbf{x}) &\triangleq H(\langle \mathbf{x} \rangle)
\end{aligned}
$$

We define G by the recursive definition

(12) $G(z) \triangleq Def_G(z, G)$

where Def_G is defined by

(13) $Def_G(z, H) \triangleq Def({}_{n}z, {}^{n}H)$

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:

$$
\begin{aligned}
& fix_G(g) \equiv \\
& \quad \forall h : (\forall x \in \text{DOMAIN } g : h[x] = g[x]) \Rightarrow \\
& \qquad\qquad (g = [x \in \text{DOMAIN } g \mapsto Def({}_{n}x, {}^{n}(\text{LAMBDA } y : g[y]))])
\end{aligned}
$$

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:

$$
\begin{aligned}
(14)\quad & \text{ASSUME} \quad \text{NEW } S, \; WellFounded(S), \\
& \qquad\qquad \text{ASSUME} \quad \text{NEW } G(\_, \ldots, \_), \; \text{NEW } H(\_, \ldots, \_) \\
& \qquad\qquad \text{PROVE} \quad \forall x \in S : (\forall y \in LT(x, S) : G(y) = H(y)) \\
& \qquad\qquad\qquad\qquad\qquad \Rightarrow (Def({}_{n}x, G) = Def({}_{n}x, H)) \\
& \text{PROVE} \quad \forall \langle \mathbf{x} \rangle \in S : F(\mathbf{x}) = Def(\mathbf{x}, F)
\end{aligned}
$$

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$^+$ because we cannot express "...", 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 $G(\langle x_1, \ldots, x_n \rangle)$. We can then deduce the desired property of F by applying Corollary 3 to G.

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:

$$\begin{aligned}(15)\quad F_1(x_1, \ldots, x_{n_1}) &\triangleq Def_1(x_1, \ldots, x_{n_1}, F_1, \ldots, F_k) \\ &\vdots \\ F_k(x_1, \ldots, x_{n_k}) &\triangleq Def_k(x_1, \ldots, x_{n_k}, F_1, \ldots, F_k)\end{aligned}$$

(Each 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:

$$(16)\quad F_i(x_1, \ldots, x_{n_i}) \triangleq G(\langle i, \langle x_1, \ldots, x_{n_i} \rangle \rangle)$$

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

$$(17)\quad G(z) \triangleq Def_G(z, G)$$

with Def_G defined by

$$\begin{aligned}(18)\quad & Def_G(z, H) \triangleq \\ & \quad \text{CASE } z[1] = 1 \rightarrow Def_1(\langle z[2][1], \ldots, z[2][n_1], \widehat{H}_1, \ldots, \widehat{H}_k) \\ & \qquad \vdots \\ & \quad \Box\ z[1] = k \rightarrow Def_k(z[2][1], \ldots, z[2][n_k], \widehat{H}_1, \ldots, \widehat{H}_k) \\ & \text{where } \widehat{H}_i(x_1, \ldots, x_{n_i}) \triangleq H(i, \langle x_1, \ldots, x_{n_i} \rangle)\end{aligned}$$

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

$$\begin{aligned}(19)\quad & \text{ASSUME} \quad \text{NEW } S,\ WellFounded(S), \\ & \qquad \text{ASSUME} \quad \text{NEW } H(\_),\ \text{NEW } J(\_) \\ & \qquad \text{PROVE} \quad \forall x \in S : (\forall y \in LT(x, S) : H(y) = J(y)) \\ & \qquad\qquad\qquad \Rightarrow (Def_G(x, H) = Def_G(x, J)) \\ & \text{PROVE} \quad \forall \langle i, \langle x_1, \ldots, x_{n_i} \rangle \rangle \in S : \\ & \qquad (i \in 1 \,..\, k) \Rightarrow \\ & \qquad\quad F_i(x_1, \ldots, x_{n_i}) = Def_i(x_1, \ldots, x_{n_i}, F_1, \ldots, F_k)\end{aligned}$$

where (18) defines 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, ..., 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

[1] Arthur Charguéraud. The optimal fixed point combinator. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6172 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2010.

[2] Leslie Lamport. *Specifying Systems*. Addison-Wesley, Boston, 2003. Also available on the Web via a link at `http://lamport.org`.