Aloalonzonzo

Christopher Schankula, Student ID: 400026650, MacID: schankuc
Dennis Yankovsky, Student ID: 400189961, MacID: yankovsd
Group 1 ("We are the Self-Referentialists")

December 2022

1 Introduction

The goal of this project is to formalize Alonzo types and expressions, the notion of free and bound variables, and the notion of substitution, in Alonzo.

2 New Notational Definitions

The following new notational definitions will be used throughout this document: $\mathsf{INJ2}_{(\alpha\to\beta\to\gamma)\to o}$ stands for $(\lambda\, f : \alpha \to \beta \to \gamma \;.\; \mathsf{INJ}(\mathrm{I}\, g : \alpha \times \beta \to \gamma \;.\; \forall\, a : \alpha, b : \beta \;.\; g\,(a, b) \simeq f\, a\, b))$

$-_{\{\alpha\}\to\{\alpha\}\to\{\alpha\}}$ stands for $\lambda\, A : \{\alpha\} \;.\; \lambda\, B : \{\alpha\} \;.\; A \cap \overline{B}$

3 Modules

Theory Definition 3.1 (Strings)

Name: STR.

Base types: S, C.

Constants: $\mathsf{Stringify}_{C \to S}$, $\mathsf{Append}_{C \to S \to S}$, $A_C, B_C, C_C, ..., Z_C$.

Axioms:

1. $\mathsf{DISTINCT}(A, B, C, ..., Z)$ (all characters are distinct)
2. $\forall c_1, c_2 : \mathsf{C}, s : S \;.\; \mathsf{Stringify} c_1 \neq \mathsf{Append} c_2 \; s$ (no confusion between Stringify and Append constructors)
3. $\mathsf{INJ}(\mathsf{Stringify})$ (no confusion)
4. $\mathsf{TOTAL}(\mathsf{Stringify})$ (no confusion)
5. $\mathsf{INJ2}(\mathsf{Append})$ (no confusion)
6. $\mathsf{TOTAL2}(\mathsf{Append})$ (no confusion)
7. $\forall\, p : S \to o \;.\; (\forall\, c : C \;.\; p\,(\mathsf{Stringify}\ c)) \wedge$
   $(\forall\, c : C,\ s : S \;.\; p\, s \Rightarrow p\,(\mathsf{Append}\, c\, s))$
   $\Rightarrow \forall\, s : S \;.\; p\, s$ (induction principle for strings)

Theory Extension 3.2 (Types)

Name: TY.

Extends STR.

New base types: $AloTy$.

New constants:
$\mathsf{BoolAloTy}_{AloTy}$,
$\mathsf{BaseAloTy}_{S \to AloTy}$,
$\mathsf{FunAloTy}_{AloTy \to AloTy \to AloTy}$,
$\mathsf{ProdAloTy}_{AloTy \to AloTy \to AloTy}$,
$\mathsf{isBoolAloTy}_{AloTy \to o}$,
$\mathsf{getBaseAloTy}_{AloTy \to S}$,
$\mathsf{getFunAloTy}_{AloTy \to AloTy \times AloTy}$,
$\mathsf{getProdAloTy}_{AloTy \to AloTy \times AloTy}$

New axioms:

1. $\forall a, b, c, d : AloTy, s : S \;.\; \mathsf{DISTINCT}(\mathsf{BoolAloTy}, \mathsf{BaseAloTy} s,$
   $\mathsf{FunAloTy} a b, \mathsf{ProdAloTy} c d)$
   (no confusion)
2. $\mathsf{INJ}(\mathsf{BaseAloTy})$ (no confusion)

3. $\mathsf{INJ2}(\mathsf{FunAloTy})$ (no confusion)

4. $\mathsf{INJ2}(\mathsf{ProdAloTy})$ (no confusion)

5. $\mathsf{TOTAL}(\mathsf{BaseAloTy})$ (all strings have a corresponding base type)

6. $\mathsf{TOTAL2}(\mathsf{FunAloTy})$ (functions accept all types)

7. $\mathsf{TOTAL2}(\mathsf{ProdAloTy})$ (products accept all types)

8. $\mathsf{isBoolAloTy} = (\lambda\, t : \mathit{AloTy} \,.\, t = \mathsf{BoolAloTy})$ (definition of isBoolAloTy)

9. $\forall\, p : \mathit{AloTy} \rightarrow o \,.\, (p\, \mathsf{BoolAloTy}) \wedge$
   $(\forall\, s : S \,.\, p\, (\mathsf{BaseAloTy}\ s)) \wedge$
   $(\forall\, t : \mathit{AloTy},\, u : \mathit{AloTy} \,.\, p\, t \wedge p\, u \Rightarrow p\, (\mathsf{FunAloTy}\, t\, u)) \wedge$
   $(\forall\, t : \mathit{AloTy},\, u : \mathit{AloTy} \,.\, p\, t \wedge p\, u \Rightarrow p\, (\mathsf{ProdAloTy}\, t\, u))$
   $\Rightarrow \forall\, t : \mathit{AloTy} \,.\, p\, t$
   (induction principle for AloTy)

10. $\mathsf{getFunAloTy} = (\lambda\, t : \mathit{AloTy} \,.\, \mathrm{I}\, s : \mathsf{S} \,.\, t = \mathsf{BaseAloTy}\ s)$
    (base type extractability law)

11. $\mathsf{getFunAloTy} = (\lambda\, t : \mathit{AloTy} \,.\, \mathrm{I}\, p : \mathit{AloTy} \times \mathit{AloTy} \,.\, t = \mathsf{FunAloTy}\ p)$
    (function type extractability law)

12. $\mathsf{getProdAloTy} = (\lambda\, t : \mathit{AloTy} \,.\, \mathrm{I}\, p : \mathit{AloTy} \times \mathit{AloTy} \,.\, t = \mathsf{ProdAloTy}\ p)$
    (product type extractability law)

Theory Extension 3.3 (Pre - Expressions)

Name: PREXPR.

Extends TY.

New base types: Expr.

New constants:
$\mathsf{VarExpr}_{S \rightarrow \mathit{AloTy} \rightarrow \mathit{Expr}}$,
$\mathsf{ConstExpr}_{S \rightarrow \mathit{AloTy} \rightarrow \mathit{Expr}}$,
$\mathsf{EqExpr}_{\mathit{Expr} \rightarrow \mathit{Expr} \rightarrow \mathit{Expr}}$,
$\mathsf{FunAppExpr}_{\mathit{Expr} \rightarrow \mathit{Expr} \rightarrow \mathit{Expr}}$,
$\mathsf{FunAbsExpr}_{\mathit{Expr} \rightarrow \mathit{Expr} \rightarrow \mathit{Expr}}$,
$\mathsf{DefDesExpr}_{\mathit{Expr} \rightarrow \mathit{Expr} \rightarrow \mathit{Expr}}$,
$\mathsf{OrdPairExpr}_{\mathit{Expr} \rightarrow \mathit{Expr} \rightarrow \mathit{Expr}}$

New axioms:

1. $\forall e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10} : \mathit{Expr}, t_1, t_2 : \mathit{AloTy}, s_1, s_2 : S$.
   $\mathsf{DISTINCT}(\mathsf{VarExpr}\ s_1\, t_1, \mathsf{ConstExpr}\ s_2\, t_2, \mathsf{EqExpr}\ e_1\, e_2,$
   $\mathsf{FunAppExpr}\ e_3\, e_4, \mathsf{FunAbsExpr}\ e_5\, e_6, \mathsf{DefDesExpr}\ e_7\, e_8,$
   $\mathsf{OrdPairExpr}\ e_9\, e_{10})$
   (no confusion)

2. $\mathsf{INJ2}(\mathsf{VarExpr})$ (no confusion)
3. $\mathsf{INJ2}(\mathsf{ConstExpr})$ (no confusion)
4. $\mathsf{INJ2}(\mathsf{EqExpr})$ (no confusion)
5. $\mathsf{INJ2}(\mathsf{FunAppExpr})$ (no confusion)
6. $\mathsf{INJ2}(\mathsf{FunAbsExpr})$ (no confusion)
7. $\mathsf{INJ2}(\mathsf{DefDesExpr})$ (no confusion)
8. $\mathsf{INJ2}(\mathsf{OrdPairExpr})$ (no confusion)
9. $\mathsf{TOTAL2}(\mathsf{VarExpr})$ (no confusion)
10. $\mathsf{TOTAL2}(\mathsf{ConstExpr})$ (no confusion)
11. $\mathsf{TOTAL2}(\mathsf{EqExpr})$ (no confusion)
12. $\mathsf{TOTAL2}(\mathsf{FunAppExpr})$ (no confusion)
13. $\mathsf{TOTAL2}(\mathsf{FunAbsExpr})$ (no confusion)
14. $\mathsf{TOTAL2}(\mathsf{DefDesExpr})$ (no confusion)
15. $\mathsf{TOTAL2}(\mathsf{OrdPairExpr})$ (no confusion)
16. $\forall\, p : \mathit{Expr} \to o \,.\, (\forall\, s : S,\, t : \mathit{AloTy} \,.\, p\,(\mathsf{VarExpr}\, s\, t)) \wedge$
    $(\forall\, s : S,\, t : \mathit{AloTy} \,.\, p\,(\mathsf{ConstExpr}\, s\, t)) \wedge$
    $(\forall\, e_1, e_2 : \mathit{Expr} \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(\mathsf{EqExpr}\, e_1 e_2)) \wedge$
    $(\forall\, e_1, e_2 : \mathit{Expr} \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(\mathsf{FunAppExpr}\, e_1 e_2)) \wedge$
    $(\forall\, e_1, e_2 : \mathit{Expr} \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(\mathsf{FunAbsExpr}\, e_1 e_2)) \wedge$
    $(\forall\, e_1, e_2 : \mathit{Expr} \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(\mathsf{DefDesExpr}\, e_1 e_2)) \wedge$
    $(\forall\, e_1, e_2 : \mathit{Expr} \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(\mathsf{OrdPairExpr}\, e_1 e_2))$
    $\Rightarrow \forall\, e : \mathit{Expr} \,.\, p\, e$
    (induction principle for Expr)

Theory Extension 3.4 (Expressions)

Name: *Expr*.

Extends PREXPR.

New base types: *Expr*.

New constants:
$\mathsf{hasAloTy}_{\mathit{Expr} \to \mathit{AloTy}}$,
$\mathsf{VE}_{\{\mathit{Expr}\}}$,
$\mathsf{sane}_{(\mathit{Expr} \to \mathit{Expr})}$,
$\mathsf{VarVExpr}_{S \to \mathit{AloTy} \to \mathit{Expr}}$,
$\mathsf{ConstVExpr}_{S \to \mathit{AloTy} \to \mathit{Expr}}$,

$\mathsf{EqVExpr}_{Expr \to Expr \to Expr}$,
$\mathsf{FunAppVExpr}_{Expr \to Expr \to Expr}$,
$\mathsf{FunAbsVExpr}_{Expr \to Expr \to Expr}$,
$\mathsf{DefDesVExpr}_{Expr \to Expr \to Expr}$,
$\mathsf{OrdPairVExpr}_{Expr \to Expr \to Expr}$

New axioms:

1. $\mathsf{hasAloTy} = \lambda\, e : Expr\ .\ \mathrm{I}\, t : AloTy\ .$
$(\exists!\, s : S\ .\ e = (\mathsf{VarExpr}\, s\, t))$
$\vee\, (\exists!\, s : S\ .\ e = (\mathsf{ConstExpr}\, s\, t))$
$\vee\, (\exists!\, e_1, e_2 : Expr\ .\ e = (\mathsf{EqExpr}\, e_1\, e_2) \wedge \mathsf{hasAloTy}\, e_1 = \mathsf{hasAloTy}\, e_2 \wedge \mathsf{isBoolAloTy}\, t)$
$\vee\, (\exists!\, e_1, e_2 : Expr,\ t' : AloTy\ .\ e = (\mathsf{FunAppExpr}\, e_1\, e_2) \wedge \mathsf{hasAloTy}\, e_1 = (\mathsf{FunAloTy}\, t'\, t) \wedge \mathsf{hasAloTy}\, e_2 = t')$
$\vee\, (\exists!\, e' : Expr,\ t' : AloTy,\ s : S\ .\ e = (\mathsf{FunAbsExpr}\, (\mathsf{VarExpr}\, s\, t')\, e') \wedge t = (\mathsf{FunAloTy}\, t'\, (\mathsf{hasAloTy}\, e')))$
$\vee\, (\exists!\, e' : Expr,\ s : S\ .\ e = (\mathsf{DefDesExpr}\, (\mathsf{VarExpr}\, s\, t)\, e') \wedge \neg\mathsf{isBoolAloTy}(t) \wedge \mathsf{isBoolAloTy}\, (\mathsf{hasAloTy}\, e'))$
$\vee\, (\exists!\, e_1, e_2 : Expr\ .\ (e = \mathsf{OrdPairExpr}\, e_1\, e_2) \wedge t = \mathsf{ProdAloTy}\, (\mathsf{hasAloTy}\, e_1)\, (\mathsf{hasAloTy}\, e_2))$
(definition of hasAloTy)

2. $\mathsf{VE}_{\{Expr\}} = \mathsf{dom}(\mathsf{hasAloTy})$ (definition of valid expressions)

3. $\mathsf{sane} = \lambda\, e : \mathsf{VE}_{\{Expr\}}\ .\ e$ (definition of Expr sanitization function)

4. $\mathsf{VarVExpr} = \mathsf{VarExpr}$ (smart constructor for variables)

5. $\mathsf{ConstVExpr} = \mathsf{ConstExpr}$ (smart constructor for constants)

6. $\mathsf{EqVExpr} = \lambda\, e_1, e_2 : \mathsf{VE}_{\{Expr\}}\ .\ \mathsf{sane}\, (\mathsf{EqExpr}\, e_1\, e_2)$
(smart constructor for equality)

7. $\mathsf{FunAppVExpr} = \lambda\, e_1, e_2 : \mathsf{VE}_{\{Expr\}}\ .\ \mathsf{sane}\, (\mathsf{FunAppExpr}\, e_1\, e_2)$
(smart constructor for function application)

8. $\mathsf{FunAbsVExpr} = \lambda\, e_1, e_2 : \mathsf{VE}_{\{Expr\}}\ .\ \mathsf{sane}\, (\mathsf{FunAbsExpr}\, e_1\, e_2)$
(smart constructor for function abstraction)

9. $\mathsf{DefDesVExpr} = \lambda\, e_1, e_2 : \mathsf{VE}_{\{Expr\}}\ .\ \mathsf{sane}\, (\mathsf{DefDesExpr}\, e_1\, e_2)$
(smart constructor for definite description)

10. $\mathsf{OrdPairVExpr} = \lambda\, e_1, e_2 : \mathsf{VE}_{\{Expr\}}\ .\ \mathsf{sane}\, (\mathsf{OrdPairExpr}\, e_1\, e_2)$
(smart constructor for ordered pair)

Theory Extension 3.5 (Free and Bound Variables)

Name: FBVAR.

Extends Expr.

New base types: N/A

New constants:

$\mathsf{varIsFree}_{Expr \to Expr \to o}$,
$\mathsf{freeVars}_{Expr \to \{Expr\}}$,
$\mathsf{varsInExpr}_{Expr \to \{Expr\}}$,
$\mathsf{varIsBound}_{Expr \to Expr \to o}$,
$\mathsf{boundVars}_{Expr \to \{Expr\}}$

New axioms:

1. $\mathsf{freeVars} = \lambda\, e : Expr\, .\, \mathrm{I}\, x : \{\mathsf{Expr}\}\, .$
   $(\exists!\, s : S,\, t : AloTy\, .\, (e = \mathsf{VarVExpr}\, s\, t) \wedge x = \{e\})$
   $\vee\, (\exists!\, s : S,\, t : AloTy\, .\, (e = \mathsf{ConstVExpr}\, s\, t) \wedge x = \emptyset_{\{Expr\}})$
   $\vee\, (\exists!\, e_1, e_2 : Expr\, .\, e = (\mathsf{EqVExpr}\, e_1\, e_2) \wedge x = (\mathsf{freeVars}\, e_1) \cup (\mathsf{freeVars}\, e_2))$
   $\vee\, (\exists!\, e_1, e_2 : Expr,\, t' : AloTy\, .\, e = (\mathsf{FunAppVExpr}\, e_1\, e_2) \wedge$
   $x = (\mathsf{freeVars}\, e_1) \cup (\mathsf{freeVars}\, e_2))$
   $\vee\, (\exists!\, v, e_1 : Expr\, .\, e = (\mathsf{FunAbsVExpr}\, v\, e_1) \wedge x = \mathsf{freeVars}(e_1) - \{v\})$
   $\vee\, (\exists!\, v, e_1 : Expr\, .\, e = (\mathsf{DefDesVExpr}\, v\, e_1) \wedge x = \mathsf{freeVars}(e_1) - \{v\})$
   $\vee (\exists!\, e_1, e_2 : Expr\, .\, (e = \mathsf{OrdPairVExpr}\, e_1\, e_2) \wedge x = \mathsf{freeVars}(e_1) \cup \mathsf{freeVars}(e_2))$
   (definition of free)

2. $\mathsf{varsInExpr} = \lambda\, e : Expr\, .\, \mathrm{I}\, x : \{\mathsf{Expr}\}\, .$
   $(\exists!\, s : S,\, t : AloTy\, .\, (e = \mathsf{VarVExpr}\, s\, t) \wedge x = \{e\})$
   $\vee\, (\exists!\, s : S,\, t : AloTy\, .\, (e = \mathsf{ConstVExpr}\, s\, t) \wedge x = \emptyset_{\{Expr\}})$
   $\vee (\exists!\, e_1, e_2 : Expr\, .\, e = (\mathsf{EqVExpr}\, e_1\, e_2) \wedge x = (\mathsf{varsInExpr}\, e_1) \cup (\mathsf{varsInExpr}\, e_2))$
   $\vee\, (\exists!\, e_1, e_2 : Expr,\, t' : AloTy\, .\, e = (\mathsf{FunAppVExpr}\, e_1\, e_2) \wedge$
   $x = (\mathsf{varsInExpr}\, e_1) \cup (\mathsf{varsInExpr}\, e_2))$
   $\vee\, (\exists!\, v, e_1 : Expr\, .\, e = (\mathsf{FunAbsVExpr}\, v\, e_1) \wedge x = \mathsf{varsInExpr}(e_1))$
   $\vee\, (\exists!\, v, e_1 : Expr\, .\, e = (\mathsf{DefDesVExpr}\, v\, e_1) \wedge x = \mathsf{varsInExpr}(e_1))$
   $\vee\, (\exists!\, e_1, e_2 : Expr\, .\, (e = \mathsf{OrdPairVExpr}\, e_1\, e_2) \wedge x = \mathsf{varsInExpr}(e_1) \cup$
   $\mathsf{varsInExpr}(e_2))$
   (definition of varsInExpr)

3. $\mathsf{boundVars} = (\lambda\, e : Expr\, .\, (\mathsf{varsInExpr}\, e) - (\mathsf{freeVars}\, e))$ (set of free)

4. $\mathsf{varIsBound} = (\lambda\, v : Expr\, .\, (\lambda\, e : Expr\, .\, (\exists!\, s : S, t : AloTy\, .\, (v = \mathsf{VarVExpr}\, s\, t) \mapsto (v \in (\mathsf{boundVars}\, e)) \mid \bot)))$ (bound variable predicate)

5. $\mathsf{varIsFree} = (\lambda\, v : Expr\, .\, (\lambda\, e : Expr\, .\, (\exists!\, s : S, t : AloTy\, .\, (v = \mathsf{VarVExpr}\, s\, t) \mapsto (v \in (\mathsf{freeVars}\, e)) \mid \bot)))$ (free variable predicate)

Theory Extension 3.6 (Substitution)

Name: SUBS.

Extends FBVAR.

New base types: N/A

New constants:

$\mathsf{sub}_{Expr \to Expr \to Expr \to Expr}$

New axioms:

1. $\mathsf{sub} = \lambda\, A : Expr\ .\ \lambda\, x : Expr\ .\ \lambda\, B : Expr\ .\ \mathrm{I}\, C : Expr\ .$
$(\mathsf{hasAloTy}\, A = \mathsf{hasAloTy}\, x) \wedge (\exists\, s : S,\, t : AloTy\ .\ x = \mathsf{VarVExpr}\, s\, t)$
$\wedge\ ((\exists !\, s : S,\, t : AloTy\ .\ B = (\mathsf{VarVExpr}\, s\, t) \wedge B = x \wedge C = A)$
$\vee\ (\exists !\, s : S,\, t : AloTy\ .\ B = (\mathsf{VarVExpr}\, s\, t) \wedge B \neq x \wedge C = B)$
$\vee\ (\exists !\, s : S,\, t : AloTy\ .\ B = (\mathsf{ConstVExpr}\, s\, t) \wedge C = B)$
$\vee\ (\exists !\, e_1, e_2 : Expr\ .\ B = (\mathsf{EqVExpr}\, e_1\, e_2) \wedge$
$\quad C = \mathsf{EqVExpr}\, (\mathsf{sub}\, A\, x\, e_1)\, (\mathsf{sub}\, A\, x\, e_2))$
$\vee\ (\exists !\, e_1, e_2 : Expr\ .\ B = (\mathsf{FunAppVExpr}\, e_1\, e_2) \wedge$
$\quad C = \mathsf{FunAppVExpr}\, (\mathsf{sub}\, A\, x\, e_1)\, (\mathsf{sub}\, A\, x\, e_2))$
$\vee\ (\exists !\, e_1, e_2 : Expr\ .\ B = (\mathsf{FunAbsVExpr}\, e_1\, e_2) \wedge e_1 = x \wedge C = B)$
$\vee\ (\exists !\, e_1, e_2 : Expr\ .\ B = (\mathsf{FunAbsVExpr}\, e_1\, e_2) \wedge e_1 \neq x \wedge$
$\quad C = \mathsf{FunAbsVExpr}\, e_1\, (\mathsf{sub}\, A\, x\, e_2))$
$\vee\ (\exists !\, e_1, e_2 : Expr\ .\ B = (\mathsf{DefDesVExpr}\, e_1\, e_2) \wedge e_1 = x \wedge C = B)$
$\vee\ (\exists !\, e_1, e_2 : Expr\ .\ B = (\mathsf{DefDesVExpr}\, e_1\, e_2) \wedge e_1 \neq x \wedge$
$\quad C = \mathsf{DefDesVExpr}\, e_1\, (\mathsf{sub}\, A\, x\, e_2))$
$\vee\ (\exists !\, e_1, e_2 : Expr\ .\ B = (\mathsf{OrdPairVExpr}\, e_1\, e_2) \wedge$
$\quad C = \mathsf{OrdPairVExpr}\, (\mathsf{sub}\, A\, x\, e_1)\, (\mathsf{sub}\, A\, x\, e_2)))$
(definition of the substitution function)

4 Conclusions

Implementing Alonzo's syntax in Alonzo was a wonderful exercise giving the team a much greater understanding of not only Alonzo but also how to use it to define inductively-defined sets as types and their properties.

4.1 Importance of Establishing Conventions

Initially, it was found that one of the most difficult parts of starting the project was being able to converse about the Alonzo system itself, as the team members easily got confused about whether one was discussing properties (e.g. types, expressions) of Alonzo itself or the embedding of Alonzo in Alonzo. Thus, it was decided that we would refer to Alonzo itself as "outer Alonzo" and our embedding of Alonzo inside Alonzo as "inner Alonzo" or "Alo" for short.

4.2 Use of Quasitypes

The use of quasitypes to restrict to expressions to well-defined ones was a very interesting use of this concept in Alonzo. It allowed us to reuse the logic from the hasAloTy function for defining the quasitype very easily, which ultimately made the smart constructors very easy to implement, if repetitive. The use of sorts in AlonzoS should be explored to make this much more type-safe.

4.3 Recommendations for Alonzo

Given the number of inductively-designed sets we needed for the project, there were a lot of extra axioms just to ensure "no confusion" and "no junk". While this was very instructive as to how inductively-designed sets like Haskell's algebaric data types work, it was also very repetitive. Thus, the first recommendation would be to create an extension of Alonzo allowing one to create these algebraic data types automatically, providing the axioms for distinctness, totality, injectivity and the induction principle.

Secondly, the use of pattern-matching in the textbook makes writing inductive definitions easier, but this is not natively supported in Alonzo. Exploring the addition of pattern-matching on inductively-defined sets is another recommendation of the team. Currently, it can be "faked" by using a definite description with a disjunction for all the cases. For base cases, this is usually a simple conjunction both matching the input and then assigning the output. For recursive cases, we use (optionally, unique) existential quantifiers for induction cases to "extract" the right variables, then set the variable bound by the definition description to the correct value for that case, or simply include the output variable as part of the matched input expression.

Thirdly, the team had to create a few new notational definitions, such as INJ2 and $-$ for set subtraction. It's recommended that Alonzo introduces these notational definitions natively, and perhaps others for injectivity for functions of more than 2 inputs.

The team also created some helper functions for Alonzo *LaTeX* notation, such as `\axNote` and `\axNoteNL` to name axioms:

```
\newcommand{\axNote}[1]{\hfill (\text{#1})}
\newcommand{\axNoteNL}[1]{\\ \text{} \axNote{#1}}
```

These could be incorporated into the official *LaTeX* for Alonzo documentation.