Alonzo in Alonzo

Christopher Schankula & Dennis Yankovsky
(Group 1: "See: Group Name")

CAS-760 Fall 2022,
McMaster University

December 5th, 2022

Aloalonzonzo

Christopher Schankula & Dennis Yankovsky
(Group 1: "See: Group Name")

CAS-760 Fall 2022,
McMaster University

December 5th, 2022

Scope

- The scope for the project was to encode Alonzo types, expressions, free and bound variables, and substitutions in Alonzo
- Stretch goals include encoding theories and semantics
- For brevity, this presentation will focus only on types, pre-expressions and expressions

Conventions: The first major problem!

- As we started talking about the project, within 5 minutes we realized we were getting ourselves confused really easily!
- For example, does the word "type" refer to an actual Alonzo type, or a type in our new "Alonzo"?
- Thus, we settled on some conventions we will use throughout the presentation:
  - "Outer Alonzo" refers to the "real" Alonzo we all know and love
  - "Inner Alonzo" or "Alo" refers to our encoding of Alonzo inside Alonzo
  - "Alo" type constructors all have the suffix "AloTy" to differentiate them from outer Alonzo's type constructors
  - Similarly, "Alo" expression constructors all have the suffix "Expr"

"But wait! Alonzo inside Alonzo? Is that even possible?"

- Nope [1]. Not entirely, anyway.

"But wait! Alonzo inside Alonzo? Is that even possible?"

- Nope [1]. Not entirely, anyway.
- Tarski's theorem of the undefinability of truth; sufficiently strong systems cannot formalize their own semantics
- Gödel's second incompleteness theorem; proving consistency

Two different approaches

- In our early discussions with Bill, we identified two different approaches we could take:
  1. Alonzo expressions as strings of characters
  2. Alonzo expressions as an abstract syntax tree
- We decided to take approach #2 as it more closely matches Alonzo's definition, and it is easier to reason about

Strings

Theory Definition (Strings)

Name: STR
Base types: C, S
Constants:

$Stringify_{C \to S}, Append_{C \to S \to S}, A_C, B_C, C_C, ..., Z_C.$

New notational definition:
"$\mathbf{C}_C^0 \mathbf{C}_C^1 .... \mathbf{C}_C^n$" stands for
Append $\mathbf{C}_C^0$ (Append $\mathbf{C}_C^1$ ...(Stringify $\mathbf{C}_C^n$)...)

Strings

Theory Definition (Strings)

Axioms:

1. DISTINCT$(A, B, C, ..., Z)$
2. $\forall c_1, c_2 : C, s : S \; . \; \mathit{Stringify}\ c_1 \neq \mathit{Append}\ c_2\ s$
3. INJ$(\mathit{Stringify})$
4. TOTAL$(\mathit{Stringify})$
5. INJ2(Append)
6. TOTAL2(Append)
7. $\forall\, p : S \rightarrow o \; . \; (\forall\, c : C \; . \; p\,(\mathit{Stringify}\ c)) \wedge$
   $(\forall\, c : C,\ s : S \; . \; p\,s \Rightarrow p\,(\mathit{Append}\ c\ s)) \Rightarrow \forall\, s : S \; . \; p\,s$

1-6 ensure "no confusion": each member of S is denoted by exactly one constructor

7 is the induction principle for S and ensures "no junk"

Types

Theory Extension (Types)

Name: TY
Extends STR
New base types: $AloTy$
New constants:

$BoolAloTy_{AloTy}$
$BaseAloTy_{S \to AloTy}$
$FunAloTy_{AloTy \to AloTy \to AloTy}$
$ProdAloTy_{AloTy \to AloTy \to AloTy}$
$isBoolAloTy_{AloTy \to o}$

Type Axioms

Theory Extension (Types)

New Axioms:

1. $\forall a, b, c, d : AloTy, s : S \ . \ DISTINCT(BoolAloTy, BaseAloTy\ s, FunAloTy\ a\ b, ProdAloTy\ c\ d)$
2. $\mathrm{INJ}(BaseAloTy)$
3. $\mathrm{INJ2}(FunAloTy)$
4. $\mathrm{INJ2}(ProdAloTy)$
5. $\mathrm{TOTAL}(BaseAloTy)$
6. $\mathrm{TOTAL2}(FunAloTy)$
7. $\mathrm{TOTAL2}(ProdAloTy)$
8. $isBoolAloTy = (\lambda\, t : AloTy \ . \ t = BoolAloTy)$

1-7 ensure "no confusion": each member of *AloTy* is denoted by exactly one constructor

8 is the definition of *isBoolAloTy*

Type Axioms

Theory Extension (Types)

New Axioms:

9 $\forall\, p : AloTy \to o \,.\, (p\, BoolAloTy) \wedge (\forall\, s : S \,.\, p\,(BaseAloTy\; s)) \wedge$
$(\forall\, t : AloTy,\, u : AloTy \,.\, p\, t \wedge p\, u \Rightarrow p\,(FunAloTy\, t\, u)) \wedge$
$(\forall\, t : AloTy,\, u : AloTy \,.\, p\, t \wedge p\, u \Rightarrow p\,(ProdAloTy\, t\, u))$
$\Rightarrow \forall\, t : AloTy \,.\, p\, t$

9 is the induction principle for the *AloTy* type. It ensures there is "no junk"

Pre-Expressions

- Pre-Expressions are expressions which are (almost! we'll explain...) syntactically correct, but not necessarily correct in terms of their types (recall Assignment 2 question 2)
- Later we will describe how we "sanitize" them such that they are guaranteed to be well-formed expressions
- (Too bad we didn't have this when solving A2Q2!)

Pre-Expressions

Theory Extension (Pre-Expressions)

Name: PREXPR

Extends TY

New base types: *Expr*

New constants:

$VarExpr_{S \to AloTy \to Expr}$

$ConstExpr_{S \to AloTy \to Expr}$

$EqExpr_{Expr \to Expr \to Expr}$,

$FunAppExpr_{Expr \to Expr \to Expr}$

$FunAbsExpr_{Expr \to Expr \to Expr}$

$DefDesExpr_{Expr \to Expr \to Expr}$

$OrdPairExpr_{Expr \to Expr \to Expr}$

Pre-Expressions

Theory Extension (Pre-Expressions)

New Axioms:

1. $\forall e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10} : \mathit{Expr}, t_1, t_2 : \mathit{AloTy}, s_1, s_2 : S$. $\mathrm{DISTINCT}(\mathit{VarExpr}\, s_1\ t_1, \mathit{ConstExpr}\, s_2\ t_2, \mathit{EqExpr}\, e_1\ e_2, \mathit{FunAppExpr}\, e_3\ e_4, \mathit{FunAbsExpr}\, e_5\ e_6, \mathit{DefDesExpr}\, e_7\ e_8, \mathit{OrdPairExpr}\, e_9\ e_{10})$

2–8 *(all constructors are injective)*

9–15 *(all constructors are total)*

1–15 ensure "no confusion": each member of *Expr* is denoted by exactly one constructor

Pre-Expressions

Theory Extension (Pre-Expressions)

New Axioms:

16 $$\begin{aligned}&\forall\, p : Expr \to o \,.\, (\forall\, s : S,\, t : AloTy \,.\, p\,(VarExpr\, s\, t))\wedge\\ &(\forall\, s : S,\, t : AloTy \,.\, p\,(ConstExpr\, s\, t))\wedge\\ &(\forall\, e_1, e_2 : Expr \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(EqExpr\, e_1\, e_2))\wedge\\ &(\forall\, e_1, e_2 : Expr \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(FunAppExpr\, e_1\, e_2))\wedge\\ &(\forall\, e_1, e_2 : Expr \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(FunAbsExpr\, e_1\, e_2))\wedge\\ &(\forall\, e_1, e_2 : Expr \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(DefDesExpr\, e_1\, e_2))\wedge\\ &(\forall\, e_1, e_2 : Expr \,.\, p\, e_1 \wedge p\, e_2 \Rightarrow p\,(OrdPairExpr\, e_1\, e_2))\\ &\Rightarrow \forall\, e : Expr \,.\, p\, e\end{aligned}$$

16 is the induction principle for the *Expr* type. It ensures there is "no junk"

(Sanitized) Expressions

- As mentioned previously, simply using the constructors of the type `Expr` will give you (almost!) syntactically-correct expressions, but not necessarily ones which are "type"-correct according to the textbook section 4.4
- Recall from A2Q2 that we can easily "write down" any expression, but that doesn't mean it's well-formed in a type (or syntax) sense
- Our constructors provide an embedding that almost eliminates "inner Alonzo" syntax errors by construction (making them type errors in "outer Alonzo")
- But "not quite" because *FunAbsExpr*, *FunAppExpr* and *DefDesExpr* accept any expression as their first argument, whereas the textbook is a bit more restrictive. We'll fix this in a moment!

(Sanitized) Expressions

- We will create new "smart constructors" (called $*VExpr$) which are only defined when returning properly-typed *Expr*s
- We will introduce three new constants as well:
  - $\text{hasAloTy}_{Expr \rightarrow AloTy}$: assigns types to well-formed expressions
  - $\text{VE}_{\{Expr\}}$: represents a quasitype of expressions which are well-formed
  - $\text{sane}_{Expr \rightarrow Expr}$: "sanitizes" an expression, returning only if valid

Expressions

Theory Extension (Expressions)

Name: EXPR

Extends PREXPR

New base types: N/A

New constants:

$$VarVExpr_{S \to AloTy \to Expr},$$
$$ConstVExpr_{S \to AloTy \to Expr},$$
$$EqVExpr_{Expr \to Expr \to Expr},$$
$$FunAppVExpr_{Expr \to Expr \to Expr},$$
$$FunAbsVExpr_{Expr \to Expr \to Expr},$$
$$DefDesVExpr_{Expr \to Expr \to Expr},$$
$$OrdPairVExpr_{Expr \to Expr \to Expr},$$
$$hasAloTy_{Expr \to AloTy},$$
$$VE_{\{Expr\}},$$
$$sane_{Expr \to Expr}$$

Expressions

Theory Extension (Expressions)

New Axioms:

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

1. Defines the rules for valid types of an expression, according to the textbook.

- This function is undefined for ill-formed expressions

Expressions

Theory Extension (Expressions)

New Axioms:

2 $VE_{\{Expr\}} = \mathrm{dom}(\mathit{hasAloTy})$

2 : valid expressions are expressions for which the *hasAloTy* function is defined (i.e. its domain)

Expressions

Theory Extension (Expressions)

New Axioms:

3 $sane = \lambda\, e : VE_{\{Expr\}} \,.\, e$

3 : the *sane* function permits only valid expressions

Expressions

Theory Extension (Expressions)

New Axioms:

4. $VarVExpr = VarExpr$
5. $ConstVExpr = ConstExpr$

4 & 5 : variables and constants are always defined

Expressions

Theory Extension (Expressions)

New Axioms:

6 $EqVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}} \;.\; sane\,(EqExpr\, e_1\; e_2)$

7 $FunAppVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}} \;.\; sane\,(FunAppExpr\, e_1\; e_2)$

8 $FunAbsVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}} \;.\; sane\,(FunAbsExpr\, e_1\; e_2)$

9 $DefDesVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}} \;.\; sane\,(DefDesExpr\, e_1\; e_2)$

10 $OrdPairVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}} \;.\; sane\,(OrdPairExpr\, e_1\; e_2)$

6 - 10 : others are validated by running them through *sane*

Expressions

Theory Extension (Expressions)

New Axioms:

2 $VE_{\{Expr\}} = \mathrm{dom}(hasAloTy)$
3 $sane = \lambda\, e : VE_{\{Expr\}}\ .\ e$
4 $VarVExpr = VarExpr$
5 $ConstVExpr = ConstExpr$
6 $EqVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}}\ .\ sane\,(EqExpr\, e_1\; e_2)$
7 $FunAppVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}}\ .\ sane\,(FunAppExpr\, e_1\; e_2)$
8 $FunAbsVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}}\ .\ sane\,(FunAbsExpr\, e_1\; e_2)$
9 $DefDesVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}}\ .\ sane\,(DefDesExpr\, e_1\; e_2)$
10 $OrdPairVExpr = \lambda\, e_1, e_2 : VE_{\{Expr\}}\ .\ sane\,(OrdPairExpr\, e_1\; e_2)$

2 : valid expressions are expressions for which the *hasAloTy* function is defined (i.e. its domain)

3 : the *sane* function permits only valid expressions

4 & 5 : variables and constants are always defined

6 - 10 : others are validated by running them through *sane*

Analysis: Tree Approach

- Recall the two possible approaches we discussed earlier:
  1. Alonzo expressions as strings of characters
  2. Alonzo expressions as an abstract syntax tree
- We decided to take approach #2 as it more closely matches Alonzo's definition
- This approach worked very well, as it allowed us to reason at a higher level than with the string approach
- Building #1 on top of number #2 is easier than the other way around
  - This would be like a "parsing" step, turning the strings into the AST
- Would recommend building the "string" approach atop this work

Analysis: Tree Approach

- Recall the two possible approaches we discussed earlier:
  1. Alonzo expressions as strings of characters
  2. Alonzo expressions as an abstract syntax tree
- One drawback: repetitive and tedious axioms for "no confusion" and "no junk"
- Would recommend adding algebraic data types to Alonzo

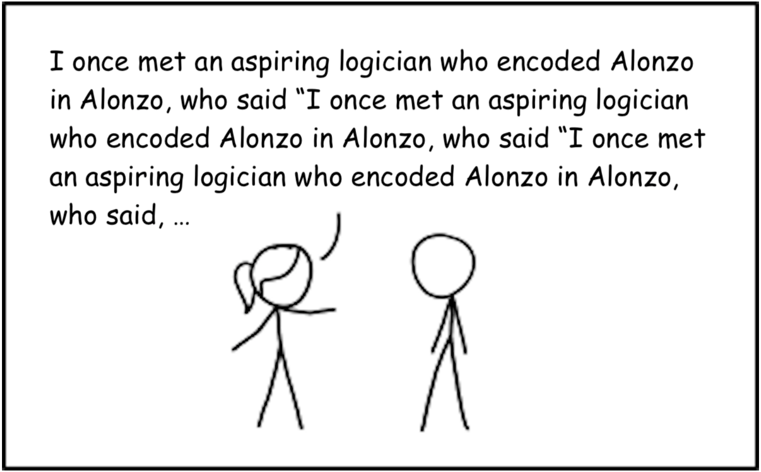
Analysis: Use of Quasitypes

- Pros:
  - Easy to reason about
  - Use the domain of our *hasAloTy* to define valid expressions: very clean and little repetition
- Cons:
  - No type-level separation between well-formed and ill-formed expressions or between our constructors
  - Need to define pretty repetitive smart constructors to use the quasitype
- Use of sorts should be explored to alleviate these issues

Conclusions & Future Work

- Talking about encoding a system in the same system is difficult!
- This was a very interesting project that deepened our understanding of Alonzo.
- Work to be submitted with report:
  - Theory of free & bound variables
  - Theory of substitutions
- Future work after this project:
  - Theory of thoeries
  - Theory of Alonzo semantics
  - Theories for additional notation (quantifiers, etc)
  - Theories for human-readable notation
- Long-term future work should explore using this as a basis for an Alonzo programming language / IDE
- Another future goal: implement Alo in Alo (and so on...) ☺

Thank you! Questions?



References

[1] Raymond M Smullyan. *Gödel's incompleteness theorems*. Oxford University Press on Demand, 1992.

Appendix

INJ2 stands for $(\lambda\, f : \alpha \rightarrow \beta \rightarrow \gamma \,.\, \mathsf{INJ}(\mathrm{I}\, g : \alpha \times \beta \rightarrow \gamma \,.\, \forall\, a : \alpha,\ b : \beta \,.\, (g\,(a, b) \simeq f\; a\; b)))$