The Magic of Specifications and Type Systems

Amin Bandali
June 17, 2017

Software Engineering Lab, EECS
York University
1. Introduction

2. Significance & Contributions

3. Type Checking

4. Well-definedness Checking

5. Conclusion
Introduction
Architects draw detailed plans before a brick is laid or a nail is hammered. Programmers and software engineers don’t.

Can this be why houses seldom collapse and programs often crash?

To designers of complex systems, the need for formal specifications should be as obvious as the need for blueprints of a skyscraper.

But few software developers write specifications because they have little time to learn how on the job, and they are unlikely to have learned in school.

— Leslie Lamport, Turing Award Winner, 2013
Architects draw detailed plans before a brick is laid or a nail is hammered. Programmers and software engineers don’t.

*Can this be why houses seldom collapse and programs often crash?*

To designers of complex systems, the need for formal specifications should be as obvious as the need for blueprints of a skyscraper.

But few software developers write specifications because they have little time to learn how on the job, and they are unlikely to have learned in school.

— Leslie Lamport, Turing Award Winner, 2013
Architects draw detailed plans before a brick is laid or a nail is hammered. Programmers and software engineers don’t.

*Can this be why houses seldom collapse and programs often crash?*

*To designers of complex systems, the need for formal specifications should be as obvious as the need for blueprints of a skyscraper.*

But few software developers write specifications because they have little time to learn how on the job, and they are unlikely to have learned in school.

— Leslie Lamport, Turing Award Winner, 2013
Architects draw detailed plans before a brick is laid or a nail is hammerd. Programmers and software engineers don’t.

**Can this be why houses seldom collapse and programs often crash?**

*To designers of complex systems, the need for formal specifications should be as obvious as the need for blueprints of a skyscraper.*

*But few software developers write specifications because they have little time to learn how on the job, and they are unlikely to have learned in school.*

— Leslie Lamport, Turing Award Winner, 2013
Architects draw detailed plans before a brick is laid or a nail is hammered. Programmers and software engineers don’t.

Can this be why houses seldom collapse and programs often crash?

To designers of complex systems, the need for formal specifications should be as obvious as the need for blueprints of a skyscraper.

But few software developers write specifications because they have little time to learn how on the job, and they are unlikely to have learned in school.

— Leslie Lamport, Turing Award Winner, 2013
Gaining Traction

Formal methods used to be relegated to safety critical systems:

- nuclear plants
- avionics
- medical devices
Some formal methods are now practical and adopted by technology leaders:

- Amazon
- Microsoft
- Facebook
- Dropbox
Significance & Contributions
Unit-B [3] is a new framework for specifying and modelling systems that must satisfy both safety and liveness properties.
Unit-B Logic supports arithmetic, sets, functions, relations, and intervals theories.
Unit-B Logic & Related Work

Unit-B vs Event-B [1]

- record types
- complete well-definedness

Unit-B vs TLA$^+$ [4]

- type checking
- [static] well-definedness checking
- quantification over infinite sets$^1$

Unit-B vs Logitext

- support for higher-order logic in both predicate and sequent calculi
Unit-B Logic & Related Work

Unit-B vs Event-B [1]

- record types
- complete well-definedness

Unit-B vs TLA⁺ [4]

- type checking
- [static] well-definedness checking
- quantification over infinite sets

Unit-B vs Logitext

- support for higher-order logic in both predicate and sequent calculi

¹limitation of the TLC tooling
Unit-B Logic & Related Work

Unit-B vs Event-B [1]

- record types
- complete well-definedness

Unit-B vs TLA⁺ [4]

- type checking
- [static] well-definedness checking
- quantification over infinite sets\(^1\)

Unit-B vs Logitext

- support for higher-order logic in both predicate and sequent calculi

\(^1\)limitation of the TLC tooling
Unit-B Web makes the Literate Unit-B prover available on the web.

While Literate Unit-B supports both the Unit-B Logic and Unit-B’s computation models, Unit-B Web currently only supports Unit-B Logic.
Unit-B Web makes the Literate Unit-B prover available on the web.

While Literate Unit-B supports both the Unit-B Logic and Unit-B’s computation models, Unit-B Web currently only supports Unit-B Logic.
Unit-B Web

Teaching

• demonstrations
• online evaluations
• support for assignments

Online Proof Environment

• making specifications more accessible to casual users
• proof of concept for a web IDE for full modelling capabilities of Unit-B
Unit-B Web

Teaching

- demonstrations
- online evaluations
- support for assignments

Online Proof Environment

- making specifications more accessible to casual users
- proof of concept for a web IDE for full modelling capabilities of Unit-B
Technology Stack

Syntax

- \LaTeX\text{-based}

Web

- JavaScript
- JSON
- Yesod / Haskell

Prover

- Haskell
- Type checking
- Well-definedness
- Proof tactics
- Z3
- Predicate prover
Syntax

- \LaTeX\textsuperscript{-based}

Web

- JavaScript
- JSON
- Yesod / Haskell

Prover

Haskell

- Type checking
- Well-definedness
- Proof tactics

Z3

- Predicate prover
Technology Stack

Syntax

- \text{\LaTeX}-based

Web

- JavaScript
- JSON
- Yesod / Haskell

Prover

- Haskell
  - Type checking
  - Well-definedness
  - Proof tactics

Z3

- Predicate prover
Type Checking
Type Checking

- \( \{x\} + 3 \leq 7 \)
- not meaningful
- caught by Unit-B’s type checker
- TLA\(^+\) doesn’t recognize this as an error
Type Checking

• \{x\} + 3 \leq 7
• not meaningful
  • caught by Unit-B’s type checker
  • TLA+ doesn’t recognize this as an error
Type Checking

- \( \{ x \} + 3 \leq 7 \)
- not meaningful
- caught by Unit-B’s type checker
- TLA\(^+\) doesn’t recognize this as an error
**Figure 1:** A type error — $x$ is expected to be a set of numbers
Type Checking

- \{x\} + 3 \leq 7
- not meaningful
- caught by Unit-B’s type checker
  - TLA\(^+\) doesn’t recognize this as an error
Type Checking

• $\{x\} + 3 \leq 7$
• not meaningful
• caught by Unit-B’s type checker
• $\text{TLA}^+$ doesn’t recognize this as an error
• TLA\(^+\)'s untyped logic allows \(\{3, \{7\}\}\)

• Event-B’s simple type system forbids this

• ???

• subtyping to the rescue!

• type variables \(\rightarrow\) polymorphic definitions
• TLA$^+$’s untyped logic allows $\{3, \{7\}\}$
• Event-B’s simple type system forbids this
  • ???
  • subtyping to the rescue!
• type variables $\rightarrow$ polymorphic definitions
Challenges & Rewards

• TLA$^+$’s untyped logic allows $\{3, \{7\}\}$
• Event-B’s simple type system forbids this
• $\text{??}$
  • subtyping to the rescue!
  • type variables $\rightarrow$ polymorphic definitions
• TLA$^+$’s untyped logic allows $\{3, \{7\}\}$
• Event-B’s simple type system forbids this
• ???
• subtyping to the rescue!
• type variables $\rightarrow$ polymorphic definitions
• TLA$^+$’s untyped logic allows $\{3, \{7\}\}$
• Event-B’s simple type system forbids this
• ???
• subtyping to the rescue!
• type variables $\rightarrow$ polymorphic definitions
Well-definedness Checking
Well-definedness Checking

Catches meaningless formulas that type checker can’t catch:

- division by zero
- array index out of bounds
- more sophisticated errors
Well-definedness Checking

Catches meaningless formulas that type checker can’t catch:

- division by zero
- array index out of bounds
- more sophisticated errors
Well-definedness Checking

Catches meaningless formulas that type checker can’t catch:

- division by zero
- array index out of bounds
- more sophisticated errors
Well-definedness Checking

Catches meaningless formulas that type checker can’t catch:

- division by zero
- array index out of bounds
- more sophisticated errors
Figure 2: An ill-defined predicate — $x$ is not in the domain of $f$
Conclusion
Summary

- **Unit-B Web**, a web application for doing predicate calculus proofs, bringing the Literate Unit-B prover to the web.

- **Type Checking** helps identify a certain class of meaningless formulas (i.e. type-incorrect formulas) efficiently.

- **Well-definedness Checking** catches the rest of meaningless formulas that are not type errors.
• **Unit-B Web**, a web application for doing predicate calculus proofs, bringing the Literate Unit-B prover to the web.

• **Type Checking** helps identify a certain class of meaningless formulas (i.e. type-incorrect formulas) efficiently.

• **Well-definedness Checking** catches the rest of meaningless formulas that are not type errors.
• **Unit-B Web**, a web application for doing predicate calculus proofs, bringing the Literate Unit-B prover to the web.

• **Type Checking** helps identify a certain class of meaningless formulas (i.e. type-incorrect formulas) efficiently.

• **Well-definedness Checking** catches the rest of meaningless formulas that are not type errors.
Unit-B Web is available under the MIT open source license. You can get the source code from GitHub:

`github.com/unitb/unitb-web`
Acknowledgements

Simon Hudon (PhD Candidate)
Professor Jonathan Ostroff
Thanks!
Summary

- **Unit-B Web**, a web application for doing predicate calculus proofs, bringing the Literate Unit-B prover to the web.
- **Type Checking** helps identify a certain class of meaningless formulas (i.e. type-incorrect formulas) efficiently.
- **Well-definedness Checking** catches the rest of meaningless formulas that are not type errors.
The source code of this presentation is available at

github.com/aminb/cucsc-2017

licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
**Polymorphic Definitions**

**SameFields**

\[ \text{SameFields}(fs, r0, r1) \triangleq \]
\[ \forall x : x \in fs : (x \in \text{dom}.r0 \land x \in \text{dom}.r1 \land r0.x = r1.x) \lor (\neg x \in \text{dom}.r0 \land \neg x \in \text{dom}.r1) \]

- Given a set of strings \((fs)\) and two records \((r0, r1)\), checks that all the specified fields have same value in both records.
- Works on any pair of records represented as partial functions.
Given a set of strings ($fs$) and two records ($r0, r1$), checks that all the specified fields have same value in both records.

- Works on any pair of records represented as partial functions.
Polymorphic Definitions

**SameFields**

\[
\text{SameFields}(fs, r0, r1) \triangleq \\
(\forall x : x \in fs : (x \in \text{dom}.r0 \land x \in \text{dom}.r1 \land r0.x = r1.x) \\
\lor (\neg x \in \text{dom}.r0 \land \neg x \in \text{dom}.r1))
\]

- Given a set of strings \((fs)\) and two records \((r0, r1)\), checks that all the specified fields have same value in both records.
- Works on any pair of records represented as partial functions.
Unit-B’s WD-calculus [2] is complete; while Event-B’s isn’t.

Consider four propositions $A$, $B$, $C$, and $D$, where

$$A \Rightarrow WD(B)$$
$$B \Rightarrow WD(C)$$
$$B \Rightarrow WD(D)$$
Unit-B’s WD-calculus [2] is complete; while Event-B’s isn’t.

Consider four propositions $A$, $B$, $C$, and $D$, where

\[
A \Rightarrow WD(B) \\
B \Rightarrow WD(C) \\
B \Rightarrow WD(D)
\]
The following calculation is not well-defined in Event-B, but it is perfectly so in Unit-B:

\[ A \land B \land (C \lor D) \]
\[ = \{ \text{commutativity} \} \]
\[ A \land (C \lor D) \land B \]
\[ = \{ \text{distributivity} \} \]
\[ ((A \land C) \lor (A \land D)) \land B \]

where

\[ A : x \in \text{dom} \cdot f \]
\[ B : f \cdot x \in \text{dom} \cdot g \]
\[ C : g \cdot (f \cdot x) \leq 3 \]
\[ D : 7 \leq g \cdot (f \cdot x) \]
Completeness

The following calculation is not well-defined in Event-B, but it is perfectly so in Unit-B:

\[ A \land B \land (C \lor D) \]
\[ = \{ \text{commutativity} \} \]
\[ A \land (C \lor D) \land B \]
\[ = \{ \text{distributivity} \} \]
\[ ((A \land C) \lor (A \land D)) \land B \]

\[ A \Rightarrow WD(B) \]
\[ B \Rightarrow WD(C) \]
\[ B \Rightarrow WD(D) \]

where

\[ A : \ x \in \text{dom.f} \]
\[ B : \ f.x \in \text{dom.g} \]
\[ C : \ g.(f.x) \leq 3 \]
\[ D : \ 7 \leq g.(f.x) \]
Completeness

The following calculation is not well-defined in Event-B, but it is perfectly so in Unit-B:

\[ A \land B \land (C \lor D) = \{ \text{commutativity} \} (A \land (C \lor D)) \land B = \{ \text{distributivity} \} ((A \land C) \lor (A \land D)) \land B \]

\[ A \Rightarrow WD(B) \]
\[ B \Rightarrow WD(C) \]
\[ B \Rightarrow WD(D) \]

where

\[ A : \ x \in \text{dom}.f \]
\[ B : \ f.x \in \text{dom}.g \]
\[ C : \ g.(f.x) \leq 3 \]
\[ D : \ 7 \leq g.(f.x) \]
Jean-Raymond Abrial.  
*Modeling in Event-B - System and Software Engineering.*  

Ádám Darvas, Farhad Mehta, and Arsenii Rudich.  
**Efficient well-definedness checking.**  

Simon Hudon, Thai Son Hoang, and Jonathan S. Ostroff.  
**The Unit-B method: refinement guided by progress concerns.**  