The Magic of Specifications and Type Systems

Amin Bandali

June 17, 2017

Software Engineering Lab, EECS
York University

Outline

1. Introduction
2. Significance & Contributions
3. Type Checking
4. Well-definedness Checking
5. Conclusion

Introduction

Specifications

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

Specifications

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

Specifications

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

Specifications

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

Specifications

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

Gaining Traction

Some formal methods are now practical and adopted by technology leaders:

- Amazon
- Microsoft
- Facebook
- Dropbox

Significance & Contributions

Unit-B

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 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[1]

Unit-B vs Logitext

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

[1] limitation of the TLC tooling

Unit-B Logic & Related Work

Unit-B vs Event-B [1]

- record types
- complete well-definedness

Unit-B vs Logitext

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

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

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

[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

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.

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

- $\LaTeX$-based

Web

- JavaScript
- JSON
- Yesod / Haskell

Prover

Haskell

- Type checking
- Well-definedness
- Proof tactics

Z3

- Predicate prover

Technology Stack

Syntax

- LaTeX-based

Web

- JavaScript
- JSON
- Yesod / Haskell

Prover

Haskell

- Type checking
- Well-definedness
- Proof tactics

Z3

- Predicate prover

Technology Stack

Syntax

- 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

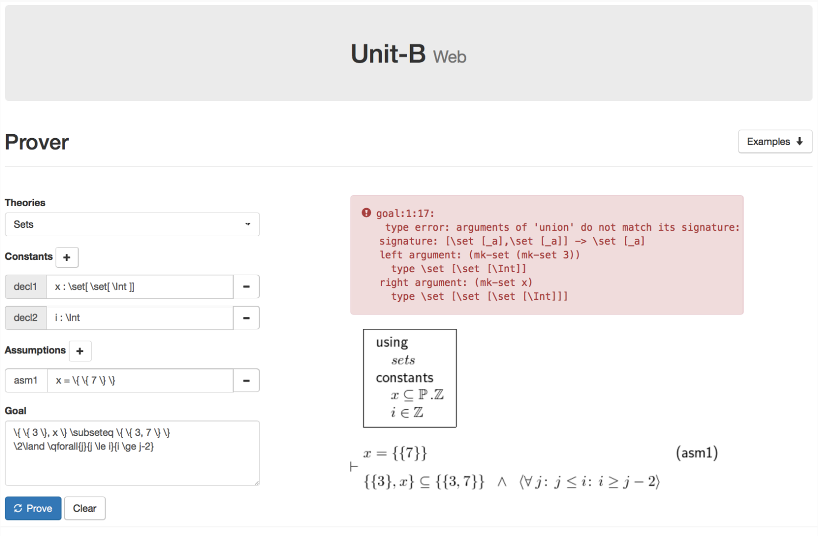

Figure 1: A type error — x is expected to be a set of numbers

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
- TLA$^+$ doesn't recognize this as an error

Challenges & Rewards

- 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
- ???
- 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
- ???
- 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
- ~~???~~
- 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
- ~~???~~
- 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

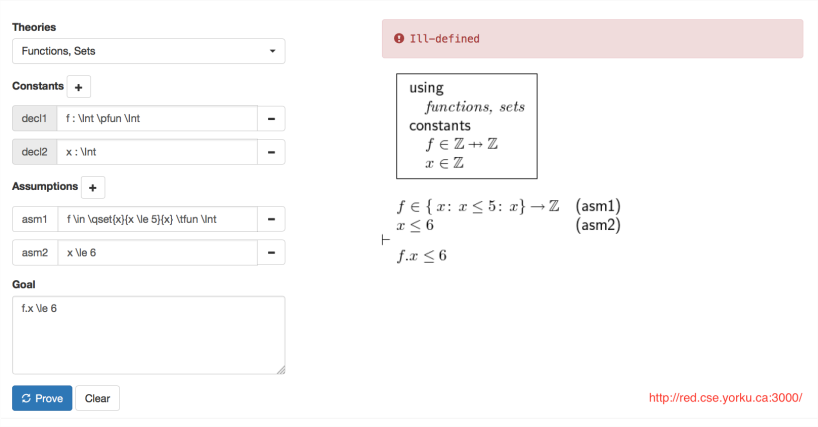

Figure 2: An ill-defined predicate — x is not in the domain of f

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.

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.

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.

Try Unit-B Web

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.

Presentation

The source code of this presentation is available at

`github.com/aminb/cucsc-2017`

Polymorphic Definitions

SameFields

$$
\begin{aligned}
&\mathit{SameFields}(\mathit{fs}, r0, r1) \triangleq \\
&\qquad (\forall x : x \in \mathit{fs} : (x \in \mathsf{dom}.r0 \land x \in \mathsf{dom}.r1 \land r0.x = r1.x) \\
&\qquad\qquad \lor (\neg x \in \mathsf{dom}.r0 \land \neg x \in \mathsf{dom}.r1))
\end{aligned}
$$

- 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

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

- 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.

Polymorphic Definitions

SameFields

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

- 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.

Completeness

Unit-B's WD-calculus [2] is complete; while Event-B's isn't.

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

$$A \Rightarrow WD(B)$$
$$B \Rightarrow WD(C)$$
$$B \Rightarrow WD(D)$$

Completeness

Unit-B's WD-calculus [2] is complete; while Event-B's isn't.

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

$$A \Rightarrow WD(B)$$
$$B \Rightarrow WD(C)$$
$$B \Rightarrow WD(D)$$

Completeness

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

$$A \Rightarrow WD(B)$$

$$B \Rightarrow WD(C)$$

$$B \Rightarrow WD(D)$$

$$
\begin{aligned}
& A \land B \land (C \lor D) \\
= & \{\textit{commutativity}\} \\
& A \land (C \lor D) \land B \\
= & \{\textit{distributivity}\} \\
& ((A \land C) \lor (A \land D)) \land B
\end{aligned}
$$

where

$$
\begin{aligned}
A: &\quad x \in dom.f \\
B: &\quad f.x \in dom.g \\
C: &\quad g.(f.x) \leq 3 \\
D: &\quad 7 \leq g.(f.x)
\end{aligned}
$$

Completeness

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

$$A \Rightarrow WD(B)$$
$$B \Rightarrow WD(C)$$
$$B \Rightarrow WD(D)$$

$$
\begin{aligned}
& A \land B \land (C \lor D) \\
= \; & \{\textit{commutativity}\} \\
& A \land (C \lor D) \land B \\
= \; & \{\textit{distributivity}\} \\
& ((A \land C) \lor (A \land D)) \land B
\end{aligned}
$$

where

$$
\begin{aligned}
A : \quad & x \in dom.f \\
B : \quad & f.x \in dom.g \\
C : \quad & g.(f.x) \leq 3 \\
D : \quad & 7 \leq g.(f.x)
\end{aligned}
$$

Completeness

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

$$A \Rightarrow WD(B)$$
$$B \Rightarrow WD(C)$$
$$B \Rightarrow WD(D)$$

$$\begin{aligned}
& A \wedge B \wedge (C \vee D) \\
= & \{\textit{commutativity}\} \\
& A \wedge (C \vee D) \wedge B \\
= & \{\textit{distributivity}\} \\
& ((A \wedge C) \vee (A \wedge D)) \wedge B
\end{aligned}$$

where

$$\begin{aligned}
A: &\quad x \in dom.f \\
B: &\quad f.x \in dom.g \\
C: &\quad g.(f.x) \le 3 \\
D: &\quad 7 \le g.(f.x)
\end{aligned}$$

References i

- Jean-Raymond Abrial.
  ***Modeling in Event-B - System and Software Engineering*.**
  Cambridge University Press, 2010.
- Ádám Darvas, Farhad Mehta, and Arsenii Rudich.
  Efficient well-definedness checking.
  In *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, pages 100–115, 2008.
- Simon Hudon, Thai Son Hoang, and Jonathan S. Ostroff.
  The Unit-B method: refinement guided by progress concerns.
  Software & Systems Modeling, pages 1–26, 2015.

Leslie Lamport.
***Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*.**
Addison-Wesley, 2002.