Behavioral Software Contracts

Robert Bruce Findler
Northwestern University & PLT
robby@eecs.northwestern.edu

Programmers embrace contracts. They can use the language they know and love to formulate logical assertions about the behavior of their programs. They can use the existing IDE infrastructure to log contracts, to test, to debug, and to profile their programs.

The keynote presents the challenges and rewards of supporting contracts in a modern, full-spectrum programming language. It covers technical challenges of contracts while demonstrating the non-technical motivation for contract system design choices and showing how contracts and contract research can serve practicing programmers.

The remainder of this article is a literature survey of contract research, with an emphasis on recent work about higher-order contracts and blame.

Early Contracts. Parnas (1972) suggested the use of logical assertions to describe software components. Meyer (1991; 1992) implemented the first full-fledged contract system and developed a matching software engineering philosophy, design by contract.

Findler and Felleisen (2002) introduced contracts to the functional programming world, generalizing them to higher-order languages, and introduced the ideas of blame and boundaries as independent concepts worthy of study.

Semantics. Findler and Felleisen used an operational model for contracts and did not define a notion of contract satisfaction. Blume and McAllester (2006) recognized this lack and responded with a quotient model, shedding light on the special status of the contract that does no checking. In parallel, Findler and Blume (2006) investigated contracts as Scott projections, thanks to a timely question from Bob Harper in 2002. Dimoulas and Felleisen (2011) countered from an observational equivalence perspective and pointed out that software engineering and formal methods naturally deal with different satisfaction relations.

Greenberg et al. (2010) studied dependent contracts, showing how there are natural variations hiding in Blume and McAllester's model. Dimoulas et al. (2011) designed a new combinator to monitor dependent contracts that assigns blame correctly when a dependent contract violates part of itself. Dimoulas et al. (2012) extended this model to introduce a notion of contract system completeness, i.e., a contract system that accounts for all possible violations.

Laziness. Contracts in lazy languages lead to complex and interesting semantic questions. As a hint at the complexity, consider a function that does not explore all of its argument, but where the unexplored part is rejected by the contract. Should this be a violation? Chitil et al. (2003) take the negative answer and show how to delay checks until the program observes the values that trigger the violation. Chitil and Huch (2006) later refine their technique to eliminate accidental sequentiality in the contract specification itself. Degen et al. (2012) tackle this question head on, showing that a contract system cannot report contract violations for all of the values that influence the program's final result without introducing unwanted strictness.

Features. Sophisticated language features demand sophisticated contract systems and more nuanced ways to assign blame.

Data structures require care to avoid excessive performance overhead (Findler et al. 2007).

Delimited and composable control operators provide new ways for values to flow and thus require special contract support (Takikawa et al. 2013).

Classes and object systems also lead to new concerns for contracts, from behavioral subtyping (Findler and Felleisen 2001; Findler et al. 2001) to support for first-class classes (Strickland and Felleisen 2010; Strickland et al. 2013).

ICFP '14, September 1–6, 2014, Gothenburg, Sweden.

ACM 978-1-4503-2873-9/14/09.
http://dx.doi.org/10.1145/2628136.2632855

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features; D.2.4 [*Software Engineering*]: Programming by contract

Keywords Contracts

Pragmatics. Strickland and Felleisen (2009) explore the crucial pragmatic question of how to draw boundaries between components.

A number of researchers have also explored parametric polymorphic contracts (Guha et al. 2007; Matthews and Ahmed 2008; Ahmed et al. 2011), using the idea that runtime sealing is the dynamic analog of polymorphic type checking.

Implementation. Herman et al. (2007) demonstrate how contract implementations break tail-recursion and design a virtual machine that recovers it.

Strickland et al. (2012) show how to add primitive interposition support to a runtime system that is strong enough to support contracts, but weak enough to avoid breaking guarantees of the underlying programming language.

Dimoulas et al. (2013) demonstrate how to give programmatic control for enabling and disabling contract checks to balance checking with performance.

Gradual Typing. The most active application of contracts is gradual typing (Flanagan 2006; Tobin-Hochstadt and Felleisen 2006; Siek and Taha 2006), which exploits dynamic contract checking so programmers can incrementally add types to untyped programs. Gronski and Flanagan (2007) clarified the relationship between gradual types and contracts.

Findler et al. (2004) gave an early instance of gradual typing, showing how structural and nominal OO type systems can coexist.

Gradual typing can be viewed as an interoperability problem, based on Matthews and Findler (2007)'s notion of a boundary. Tov and Pucella (2010) use this perspective to connect a language with an affine type systems to a simply-typed one using contracts that exploit mutable references to track how often resources are used.

Wadler and Findler (2009) and Dimoulas et al. (2012) refined the proof techniques for the Blame Theorem for gradually typed calculi, which ensures that either blame for a contract violation lies on the side with the weak type system or that the type is too strong.

References

A. Ahmed, R. B. Findler, J. Matthews, and P. Wadler. Blame for All. In *Proc. ACM Sym. Principles of Programming Languages*, 2011.

M. Blume and D. McAllester. Sound and Complete Models of Contracts. *J. Functional Programming* 16(4-5), 2006.

O. Chitil and F. Huch. A Pattern Logic for Prompt Lazy Assertions in Haskell. In *Proc. Implementation and Application of Functional Languages*, 2006.

O. Chitil, D. McNeill, and C. Runciman. Lazy Assertions. In *Proc. Implementation and Application of Functional Languages*, 2003.

M. Degen, P. Thiemann, and S. Wehr. The Interaction of Contracts and Laziness. In *Proc. Partial Evaluation and Program Manipulation*, 2012.

C. Dimoulas and M. Felleisen. On Contract Satisfaction in a Higher-Order World. *Trans. Programming Languages and Systems* 33(5), 2011.

C. Dimoulas, R. B. Findler, and M. Felleisen. Option Contracts. In *Proc. ACM Conf. Object-Oriented Programming, Systems, Languages and Applications*, 2013.

C. Dimoulas, R. B. Findler, C. Flanagan, and M. Felleisen. Correct Blame for Contracts: No More Scapegoating. In *Proc. ACM Sym. Principles of Programming Languages*, 2011.

C. Dimoulas, S. Tobin-Hochstadt, and M. Felleisen. Complete Monitors for Behavioral Contracts. In *Proc. Europ. Sym. on Programming*, 2012.

R. B. Findler and M. Blume. Contracts as Pairs of Projections. In *Proc. Sym. Functional and Logic Programming*, 2006.

R. B. Findler and M. Felleisen. Contract Soundness for Object-Oriented Languages. In *Proc. ACM Conf. Object-Oriented Programming, Systems, Languages and Applications*, 2001.

R. B. Findler and M. Felleisen. Contracts for Higher-order Functions. In *Proc. ACM Intl. Conf. Functional Programming*, 2002.

R. B. Findler, M. Flatt, and M. Felleisen. Semantic Casts: Contracts and Structural Subtyping in a Nominal World. In *Proc. Europ. Conf. Object-Oriented Programming*, 2004.

R. B. Findler, S. Guo, and A. Rogers. Lazy Contract Checking for Immutable Data Structures. In *Proc. Implementation and Application of Functional Languages*, 2007.

R. B. Findler, M. Latendresse, and M. Felleisen. Behavioral Contracts and Behavioral Subtyping. In *Proc. ACM Conf. Object-Oriented Programming, Systems, Languages and Applications*, 2001.

C. Flanagan. Hybrid Type Checking. In *Proc. ACM Sym. Principles of Programming Languages*, 2006.

M. Greenberg, B. C. Pierce, and S. Weirich. Contracts Made Manifest. In *Proc. ACM Sym. Principles of Programming Languages*, 2010.

J. Gronski and C. Flanagan. Unifying Hybrid Types and Contracts. In *Proc. Sym. Trends in Functional Programming*, 2007.

A. Guha, J. Matthews, R. B. Findler, and S. Krishnamurthi. Relationally-Parametric Polymorphic Contracts. In *Proc. Dynamic Languages Symposium*, 2007.

D. Herman, A. Tomb, and C. Flanagan. Space-Efficient Gradual Typing. In *Proc. Sym. Trends in Functional Programming*, 2007.

J. Matthews and A. Ahmed. Parametric Polymorphism Through Run-Time Sealing or, Theorems for Low, Low Prices! In *Proc. Europ. Sym. on Programming*, 2008.

J. Matthews and R. B. Findler. Operational Semantics for Multi-Language Programs. In *Proc. ACM Sym. Principles of Programming Languages*, 2007.

B. Meyer. *Eiffel: The Language*. Prentice Hall, 1991.

B. Meyer. Applying "Design by Contract". *IEEE Computer* 25(10), 1992.

D. L. Parnas. A Technique for Software Module Specification with Examples. *Communications of the ACM* 15(5), 1972.

J. G. Siek and W. Taha. Gradual Typing for Functional Languages. In *Proc. Scheme and Functional Programming*, 2006.

T. S. Strickland, C. Dimoulas, A. Takikawa, and M. Felleisen. Contracts for First-Class Classes. *Trans. Programming Languages and Systems* 35(3), 2013.

T. S. Strickland and M. Felleisen. Nested and Dynamic Contract Boundaries. In *Proc. Implementation and Application of Functional Languages*, 2009.

T. S. Strickland and M. Felleisen. Contracts for First-Class Classes. In *Proc. Dynamic Languages Symposium*, 2010.

T. S. Strickland, S. Tobin-Hochstadt, R. B. Findler, and M. Flatt. Chaperones and Impersonators: Run-Time Support for Reasonable Interposition. In *Proc. ACM Conf. Object-Oriented Programming, Systems, Languages and Applications*, 2012.

A. Takikawa, T. S. Strickland, and S. Tobin-Hochstadt. Constraining Delimited Control with Contracts. In *Proc. Europ. Sym. on Programming*, 2013.

S. Tobin-Hochstadt and M. Felleisen. Interlanguage Migration: from Scripts to Programs. In *Proc. Dynamic Languages Symposium*, 2006.

J. A. Tov and R. Pucella. Stateful Contracts for Affine Types. In *Proc. Europ. Sym. on Programming*, 2010.

P. Wadler and R. B. Findler. Well-typed Programs Can't Be Blamed. In *Proc. Europ. Sym. on Programming*, 2009.