The authors postulate a language with both untyped and statically-typed syntax; their main result is that in such a language, a program with exactly one typed module will never raise a runtime type error due to type-checked code [2]. A secondary result is that the typed module will accurately report all runtime type errors caused by interaction with an untyped module.

The paper describes interoperability and runtime type enforcement using higher-order contracts to specify the behavior of functions. Inferred contracts list the typed module's expectations for untyped identifiers, and compiled contracts protect type invariants in untyped code.

Strengths

- The migration story is plausible. (But wouldn't we be better off fully-typed from the beginning?)
- The emphasis on preserving typed invariants & the compiler from contracts to types are novel.

Weaknesses

- The formalism in Section 2 should match the examples in Section 2.
- A "philosophical assumption" is missing: that the dynamically-typed language supports run-time, higher-order contracts.
- The contract language is limited to match the type language. One cannot write a faithful specification of functions like `square-root` [1].
- Poor title. Though, it surprisingly has more syllables than the authors' full names combined.

References

[1] Robert Bruce Findler and Matthias Felleisen. Contracts for higher-order functions. In *ICFP*, 2002.

[2] Sam Tobin-Hochstadt and Matthias Felleisen. Interlanguage migration: From scripts to programs. In *DLS*, 2006.