

Formalising Configuration Languages: Why is this Important in Practice?

Paul Anderson and Herry Herry, ***University of Edinburgh***

https://www.usenix.org/conference/lisa14/poster-session/poster/anderson

This paper is included in the Proceedings of the 28th Large Installation System Administration Conference (LISA14).

November 9–14, 2014 • Seattle, WA

ISBN 978-1-931971-17-1

Open access to the Proceedings of the 28th Large Installation System Administration Conference (LISA14) is sponsored by USENIX

Formalising Configuration Languages
Why is this important in practice?

Paul Anderson
University of Edinburgh
dcspaul@ed.ac.uk

Herry Herry
University of Edinburgh
h.herry@sms.ed.ac.uk

Formalising configuration languages sounds rather academic - why is this important in practice?

Almost all large installations - including critical systems such as air traffic control[4] - now rely very heavily on configuration tools, and their associated languages, to define and manage their infrastructure. This growth means that configuration errors are becoming more serious, and more prevalent: they were found to be the second major cause of service-level failures in one of Google's main services[3], and they have been responsible for serious outages at most of the main internet service providers.

Unlike modern programming languages, configuration languages are usually defined by a single implementation, with no formal specification of the language - ambiguities and implementation errors are much more likely, and it is hard to create alternative compilers or other tools.

How does formalisation help with these problems?

• A precise definition enables us to create multiple implementations which are truly compatible and "correct".
• It also allows other people to experiment with compatible language extensions.
• And to implement supporting tools such as IDEs, graphical tools, analysers ...
• We can formally prove properties of the configuration making it highly reliable.
• Crucially, the process of developing the semantics also gives us a deeper understanding of the language and highlights problems with the language design.

What did we do?

We used an approach known as *denotational semantics* to develop a formal specification for the core of the SmartFrog[1] configuration language. SmartFrog is a declarative configuration language which is comparatively well-defined and has a typical structure. The denotational approach provides a direct mapping from the statements of the language onto their "meaning" - i.e. the real, resulting configuration.

What did this achieve?

• We used the semantics to prove some critical properties of the language, such as the fact that the compiler always terminates.
• Two people independently created three different implementations of the compiler[2] using the semantics as a specification. These were automatically compared using auto-generated, and hand-crafted examples, and found to be highly compatible. Two of these were extended to be fully compatible with the production compiler.
• We identified real bugs in the production compiler which has been in use for many years.
• We identified reasons for some "awkward" features in the language and possible ways of avoiding these.

What did we learn from this?

• It is possible to develop a formal semantics for configuration languages, and this helps to alleviate many of the practical problems mentioned above, and makes it much easier to create clear and correct compilers.
• This process is much more natural for "declarative" languages and further strengthens the case for their use (this bears comparison with the growth in popularity of functional programming languages).
• A more careful approach to the design and evolution of production configuration languages is necessary to avoid deep problems being caused by apparently small language changes.

References

[1] SmartFrog
http://www.smartfrog.org

[2] Demonstration SmartFrog compilers
https://github.com/herry13/smartfrog-lang

[3] Luiz André Barroso and Urs Hölzle. 2009. The Datacenter as a Computer: An Introduction to the Design of Warehouse-scale Machines

[4] Vervloesem K., FOSDEM 2011
http://www.lwn.net/Articles/428207