Combining Agda with External Tools

Stephan Adelsberger[1] and Anton Setzer[2]

Agda Implementors meeting XXXII Online

1 June 2020

[1]WU Vienna, Austria, https://nm.wu.ac.at/nm/en:adelsberger
[2]Swansea University, UK, http://www.cs.swan.ac.uk/~csetzer/index.html

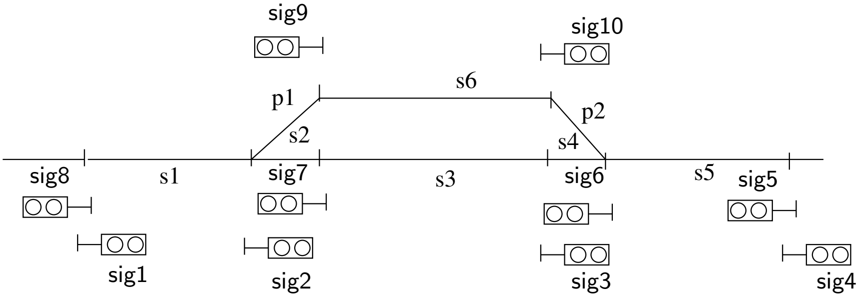
Integrating External Tools via Builtins

Integrating λ-Prolog into Agda

Connecting Agda with why3 and SPARK Ada

Karim Kanso (PhD thesis) Verification of Real World Railway Interlocking Systems using Agda

Example of Railway Interlocking System:



Approach

- We have a control program P which depending on commands and detected trains in segments sets the signals and sets of points.
- So we have vectors of Booleans expressing
  - the state of the system $\overrightarrow{\mathrm{State}}$,
  - and the inputs $\overrightarrow{\mathrm{Input}}$.
- P can be expressed as Boolean valued formulae

$$\varphi_P(\overrightarrow{\mathrm{State}_{\mathrm{in}}}, \overrightarrow{\mathrm{Input}}, \overrightarrow{\mathrm{State}_{\mathrm{out}}})$$

Proof of Safety in Agda

- We can write a **simulator in Agda** for this programs, which moves trains, around, provided they obey signals and executes P.
- A **state of the program is safe** if
  - there are never two trains in the same train segment,
  - more conditions esp. regarding sets of points.
- P **is safe** if from specific allowed initial states when running the program and moving trains one never reaches an unsafe state.
- Difficult to do directly in Agda because φ_P is very complex.
- Instead **separate** tasks between interactive theorem proving (**ITP**) and automated theorem proving (**ATP**).
  - By ATP we mean here SAT solvers and model checkers
  - Later we discuss as well other ATP tools.

Distribution of Tasks between interactive and automated theorem proving

- Introduce safety conditions $\varphi_{\text{safe}}(\overrightarrow{\text{State}})$ and invariants $\varphi_{\text{invariant}}(\overrightarrow{\text{State}})$
- Prove using **ATP** certain **signalling principles**

$$
\begin{array}{l}
(\varphi_{\text{safe}}(\overrightarrow{\text{State}_{\text{in}}}) \wedge \varphi_{\text{invariant}}(\overrightarrow{\text{State}_{\text{in}}}) \wedge \varphi_P(\overrightarrow{\text{State}_{\text{in}}}, \overrightarrow{\text{Input}}, \overrightarrow{\text{State}_{\text{out}}})) \\
\to \varphi_{\text{safe}}(\overrightarrow{\text{State}_{\text{out}}}) \wedge \varphi_{\text{invariant}}(\overrightarrow{\text{State}_{\text{out}}})
\end{array}
$$

- Prove using **ITP** that signalling principles imply that P is safe.
- In order to get a complete proof in Agda, we need
  - not only that ATP returns value true,
  - but as well that this implies that the checked formula is true.

Approach in Karim's Thesis [1, 2, 3, 4].

- Develop a naive SAT solver or model checker in Agda, and show it is sound:

$$\begin{array}{lll} \mathrm{check} & : & \mathrm{Formula} \to \mathrm{Bool} \\ \mathrm{sound} & : & (\varphi : \mathrm{Formula}) \to \mathrm{T}\ (\mathrm{check}\ \varphi) \to (\xi : \mathrm{Env}) \to [\![\, \varphi \,]\!]\xi \end{array}$$

- We override the check function by a **Builtin**, which calls an efficient SAT solver or model checker.
- Function sound links the result check from ATP to the validity of a formula which can be used in ITP.
- Now we get
  - Using ATP we check that signalling principles hold
  - Using the Builtin we translate the results into validity of the signalling principles in Agda.
  - Using ITP we prove that this implies that the system is safe.

Need for Flexible Builtins

- In order to get this machinery work we need two Builtins.
  - The function check.
  - The type of formulas $\mathrm{Formula}$.
- For more complex logics (e.g. for model checking) one needs a **cascade of Builtins**.
- Approach relies on trusting the ATP tool giving correct result.

Using Builtins for Proof Search

- Karim linked as well tools for **proof search** to Agda using Builtins.
  - Karim used a SAT solver so the tool was total.
  - Here we show how to extend this to semi decision procedures.
- Assume you have an ATP tool which searches for proofs for certain formulas.
  - We have

$$\begin{array}{lll} \mathrm{Formula} & : & \mathrm{Set} \\ \mathrm{Proof} & : & \mathrm{Formula} \to \mathrm{Set} \end{array}$$

  - The ATP tool gives a function

$$\mathrm{poofsearch} : (\varphi : \mathrm{Formula}) \to \mathrm{Maybe}\ (\mathrm{Proof}\ \varphi)$$

- In Agda we can postulate such a function

$$\mathrm{postulate\ poofsearch} : (\varphi : \mathrm{Formula}) \to \mathrm{Maybe}\ (\mathrm{Proof}\ \varphi)$$

  and override it using a builtin by the ATP tool.

Using Builtins for Proof Search

- In Agda we prove soundness

$$\mathrm{sound} : (\varphi : \mathrm{Formula}) \to \mathrm{Proof}\ \varphi \to (\xi : \mathrm{Env}) \to [\![\varphi]\!]\xi$$

- We define

$$\mathrm{extract} : \{X : \mathrm{Set}\} \to (p : \mathrm{Maybe}\ X) \to \mathrm{IsJust}\ p \to X$$

- Therefore we get a proof

$$\mathrm{sound}\ \varphi\ (\mathrm{extract}\ (\mathrm{poofsearch}\ \varphi)\ \mathrm{isJust}) : (\xi : \mathrm{Env}) \to [\![\varphi]\!]\xi$$

provided $\mathrm{poofsearch}\ \varphi$ returns a just value
(type checking will run the external tool when checking
$\mathrm{isJust} : \mathrm{IsJust}\ (\mathrm{poofsearch}\ \varphi)$).

Advantages/Disadvantages of Approach using Profs

- Advantages
  - No reliance on the soundness of the ATP tool.
  - No need to write a naive implementation of the tool.
  - Allows as well ATP tools for semi decidable logics or which for other reasons don't always give an answer.
- Disadvantages
  - Slower to use since ATP tool needs to create a proof.
  - Restricts ATP tools available.
    - Especially model checkers usually don't provide proofs.
  - Tedious to translate ATP proofs into Agda
    - lack of documentation,
    - scripts not intended to be converted into Agda proofs.

Flexible Builtin Mechanism

- Builtins can be used for other purposes as well
  - cryptographic functions,
  - any computational complex functions.
- Karim added a flexible mechanism for adding builtins to Agda.

Caveats

- Allowing to add new builtins in Agda code causes a **security problem**, because it allows to execute arbitrary programs during type checking.
  - Solution: require that adding new builtin mechanism requires recompilation of Agda.
- Builtins are only **consistent** if the output of the builtin tool coincides with the the output of Agda.
  - Requires **checks** in Agda.
  - In case of **overridden postulates** requires that the original function was indeed a postulate.
- Karim's approach is reasonably flexible but still requires some programming.
  - A too generic approach will probably become inefficient.
  - Karim wrote a **domain specific language** for this to make it easy to add Builtins.

Code Sprint

- Karim created a branch [3] of Agda with his implementation of Builtins.
- Documented esp. in Appendix D and Sect. 5 of his PhD Thesis [1].
- Agda code and other material available from [2]
  (linked as well from the AIM XXXII webpage,
  see Code Sprint on Builtins)
- Goal of code sprint is to update it and integrate it into main Agda.

Integrating External Tools via Builtins

Integrating λ-Prolog into Agda

Connecting Agda with why3 and SPARK Ada

Presented by Stephan Adelsberger

Integrating External Tools via Builtins

Integrating λ-Prolog into Agda

Connecting Agda with why3 and SPARK Ada

SPARK Ada

- SPARK Ada is a tool set used in industry for developing safety critical systems.
- It extends Ada programs by adding data/information flow analysis and Hoare logic.
- Hoare logic allows to add pre-, post conditions to a program plus intermediate conditions, especially loop invariants.

Example

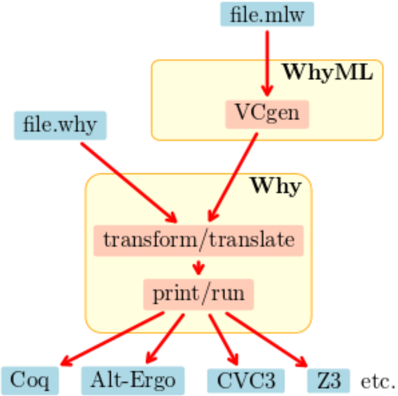
```
procedure Correct_Increment(X : in out Integer)
   with Depends   =>   (X => X),
        Pre       =>   X >= 0,
        Post      =>   X = X'Old + 1 and X >= 1;
procedure body Correct_Increment(X : in out Integer) is
   begin
      X := X + 1;
   end Correct_Increment;
```

Why3 Platform

- SPARK Ada uses the Why3 system from **INRIA**.
- Why3 is a tool which converts imperative code from the intermediate languages mlw and code from the language why3 into generated verification conditions which are then fed into various[3]
  - automated theorem provers
    Alt-ergo, Beagle, CVC3, CVC4, E prover, Gappa, Metis, Metitatrski, Princess, Psyche, Simplify, SPASS, Vampire, veriT, Yices, Ze.
  - interactive theorem provers
    Coq, PVC and Isabelle/HOL.
- SPARK Ada uses the why3 system to generate from a program and pre-/post-conditions and intermediate conditions verification conditions and feed them into the automated theorem prover alt-ergo.

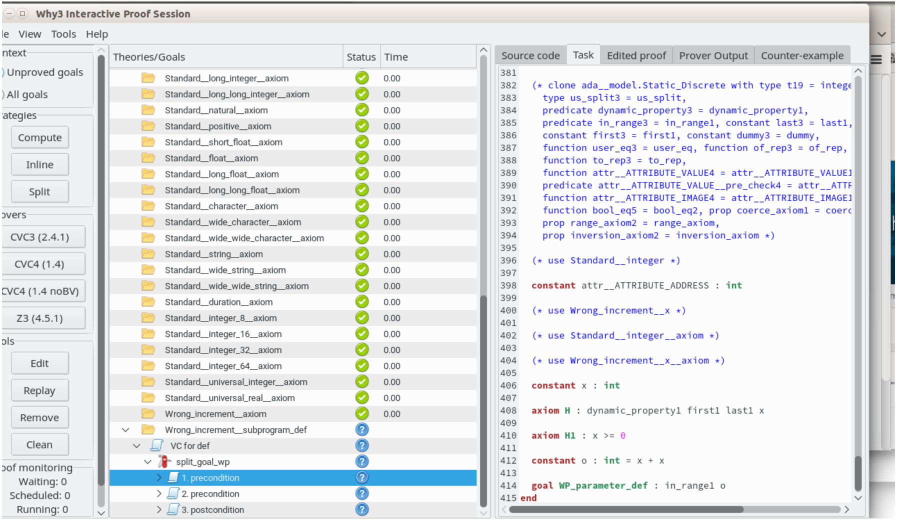
[3] http://why3.lri.fr/

Architecture of Why3 Platform



(Source: http://why3.lri.fr/queens/queens.pdf)

Result of Applying Why3 to .mlw Files



Need for Interactive Theorem Provers

- SPARK Ada works well when having verification conditions in propositional logic.
- As soon as one introduces quantifiers, one quickly reaches the limit of automated theorem provers.
- Workaround is to write verification conditions in propositional logic.
  - Instead of writing

$$\forall \mathrm{signal}_1, \mathrm{signal}_2 : \mathrm{Signal.oppose}(\mathrm{signal}_1, \mathrm{signal}_2) \wedge \mathrm{IsGreen}(\mathrm{signal}_1) \\ \to \mathrm{IsRed}(\mathrm{signal}_2)$$

  - one writes instead for each concrete signals signal_1, signal_2 opposing each others

$$\mathrm{IsGreen}(\mathrm{signal}_1) \to \mathrm{IsRed}(\mathrm{signal}_2)$$

  - Specification becomes very long (lots and lots of conditions) and it is likely to overlook a condition.
  - Instead of a program errors one is facing specification errors.

Incorporating Hoare Logic into Agda

- Therefore a good idea to link ITP tools such a Agda to why3.
- Linking Agda to why3 would provide an easy way of getting Hoare logic into Agda.
- It would allow to verify “real” programs in Agda.
- Will certainly depend on integration of ATP tools in Agda.

Bibliography I

K. Kanso.
Agda as a Platform for the Development of Verified Railway Interlocking Systems.
PhD thesis, Dept. of Computer Science, Swansea University, Swansea SA2 8PP, UK, August 2012.
Available from http://www.swan.ac.uk/~csetzer/articlesFromOthers/index.html and http://cs.swan.ac.uk/~cskarim/files/.

K. Kanso.
Code of phd thesis, February 2013.
http://www.cs.swan.ac.uk/~csetzer/articlesFromOthers/index.html.
Main code

Bibliography II

http://www.cs.swan.ac.uk/~csetzer/articlesFromOthers/kanso/codeKansoPhDThesis.zip;
Agda fork
https://github.com/kazkansouh/agda;
material regarding the interlocking of the historic railway Gwili
http://www.cs.swan.ac.uk/~csetzer/articlesFromOthers/kanso/karimKansoPhDThesisAgdaAsAPlatformForVerifiedRailwaysGwili.tar.bz2.

K. Kanso.
Agda, 3 September 2017.
Github repository, fork of Agda installation, containing code from PhD thesis Karim Kanso.

Bibliography III

K. Kanso and A. Setzer.
A light-weight integration of automated and interactive theorem proving.
Mathematical Structures in Computer Science, FirstView:1–25, 12 November 2014.