How to explain Why3

*UPSCaLe WG*

4th June 2019

Quentin Garchery

*LRI, Univ. Paris-Sud*
Overview of Why3

- annotated WhyML program
- Why3
- $t$ → $t_1$, $t_2$, $t_3$
- $\phi$
- Z3, Alt-Ergo, CVC4
Motivation: improve trust

Trust Why3?

- Automatic provers
- Transformations
- Task generation
Motivation : improve trust

Trust Why3?

Automatic provers   SMTCoq

Transformations

Task generation
Motivation: improve trust

Trust Why3?

Automatic provers  SMTCoq

Transformations  subject of this talk

Task generation
Motivation: improve trust

Trust Why3?

- Automatic provers
- Transformations
- Task generation

SMTCoq

subject of this talk

future works
Drivers for automatic provers

annotated WhyML program → Why3

\[ t \phi \]

\[ t_1 \quad \text{Z3 driver} \quad t'_1 \quad \text{Z3} \]
\[ t_2 \quad \text{Alt-Ergo driver} \quad t'_2 \quad \text{Alt-Ergo} \]
\[ t_3 \quad \text{CVC4 driver} \quad t'_3 \quad \text{CVC4} \]
Skeptical and autarkic approaches

**Autarkic approach**
Transformations are correct w.r.t. every task
→ prove the code of the transformation

**Skeptical approach**
Transformations are correct on tasks they are applied to
→ check that the application is correct

With certificates, we get:
- modularity
- ease of certification

We do not lose efficiency, for example: $\Gamma, H : A, \Gamma' \vdash A$
Simplification of tasks

New representation $ctask$:
- no memoization
- allow multiple goals
- locally nameless representation of terms

Write a translation from a why3 task to a $ctask$

Use the notation $\Gamma \vdash \Delta$ for both
Skeptical approach with *ctask*
Skeptical approach with $ctask$
Skeptical approach with \textit{ctask}
Trust with checker

What is trusted still:

- a translated task is 'equivalent' to the original task
- our code checks that the diagram commutes
- \( \forall \text{certif } ctask. \bigwedge \text{ccheck certif } ctask \Rightarrow ctask \)

![Diagram showing the relationship between translated tasks and their checks.](image)
Originality of the method

Works when task is left open

Diagram commutation catches bugs:

\[ \phi (\not\vdash A_1 \land A_2) \text{ returns } certif \text{ and } \vdash A_1 \]

\[ \vdash A_1 \]

\[ ccheck \text{ certif } (\not\vdash A_1 \land A_2) \text{ does not return } \vdash A_1 \]

\[ \vdash A_1 \]
Certificates and checker

We define:

- certificates \( \text{certif} \)
  \( \rightarrow \) skeleton of a sequent calculus derivation
- a function \( \text{ccheck} : \text{certif} \rightarrow \text{ctask} \rightarrow \text{ctask list} \)
  \( \rightarrow \) \( \text{ctask} \) is the root
  \( \rightarrow \) recursively computes the nodes of \( \text{certif} \)

Example:

With \( t := \Gamma, H : A \vdash \Delta, G : A \)
and \( c := \text{Assumption} \ (H, G) \)
we compute \( \text{ccheck} \ c \ t = [] \)

We then write:

\[
\Gamma, H : A \vdash \Delta, G : A \quad \text{Assumption} \ (H, G)
\]
Chaining certificates

Compute:

\[
\text{ccheck} \left( \text{Split} \ (G, c_1, c_2) \right) \ (\vdash G : A_1 \land A_2)
\]
\[
\equiv \text{ccheck} \ c_1 \ (\vdash G : A_1) \odot \text{ccheck} \ c_2 \ (\vdash G : A_2)
\]
\[
\equiv \text{lt}_1 \odot \text{lt}_2
\]

With \( \text{lt}_1 := \text{ccheck} \ c_1 \ (\vdash G : A_1) \) and \( \text{lt}_2 := \text{ccheck} \ c_2 \ (\vdash G : A_2) \)

Represent this computation as a derivation:

\[
\frac{\text{lt}_1 \ c_1 \quad \text{lt}_2 \ c_2}{\vdash G : A_1 \land A_2} \quad \text{Split} \ (G, c_1, c_2)
\]
Chaining certificates: drunker’s theorem

Suppose we have \( a \in A \) and pose:

\[
dr := \exists x. (P x \Rightarrow \forall y. P y)
\]

\[
\frac{H_1 : P a, H_2 : P y}{H_1 : P a \vdash G_1 : dr, G_2 : P y, G_3 : \forall y. P y}
\]

Assumption\((H_2, G_2)\)

\[
\frac{H_1 : P a \vdash G_1 : dr, G_2 : P y, G_3 : P y \Rightarrow \forall y. P y}{\text{Intro}(G_3, H_2, -)}
\]

\[
\frac{H_1 : P a \vdash G_1 : dr, G_2 : P y}{\text{InstQuant}(G_1, y, G_3, -)}
\]

\[
\frac{H_1 : P a \vdash G_1 : dr, G_2 : \forall y. P y}{\text{IntroQuant}(G_2, -)}
\]

\[
\frac{\vdash G_1 : dr, G_2 : P a \Rightarrow \forall y. P y}{\text{Intro}(G_2, H_1, -)}
\]

\[
\frac{\vdash G_1 : dr, G_2 : P a \Rightarrow \forall y. P y}{\text{InstQuant}(G_1, a, G_2, -)}
\]
Composing certifying transformations

Certificate *Skip* when task is unchanged

Consider \( \phi \) that completely splits a \( n \)-ary conjunction:
\( \phi \) applied to \( \vdash G : (A_1 \land A_2) \land A_3 \) gives the tasks \( \vdash G : A_i \)

Certified with \( \text{certif} := \text{Split}(G, \text{Split}(G, \text{Skip}, \text{Skip}), \text{Skip}) \)

\[
\begin{align*}
\vdash G : A_1 & \quad \vdash G : A_2 & \quad \vdash G : A_3 \\
\vdash G : \text{Skip} & \quad \vdash G : \text{Skip} & \quad \vdash G : \text{Skip} \\
\vdash G : A_1 \land A_2 & \quad \vdash G : \text{Split}(G, \text{Skip}, \text{Skip}) & \quad \vdash G : \text{Certif} \\
\vdash G : (A_1 \land A_2) \land A_3 &
\end{align*}
\]
Composing certifying transformations

Write $\phi$ as a composition of transformations?

$$\phi' (\vdash G : (A_1 \land A_2) \land A_3) = \left[ \vdash G : A_1 \land A_2 ; \vdash A_3 \right],$$
$$\text{Split}(G, \text{Skip}, \text{Skip})$$

$$\phi' (\vdash A_1 \land A_2) = \left[ \vdash G : A_1 ; \vdash G : A_2 \right],$$
$$\text{Split}(G, \text{Skip}, \text{Skip})$$

$$\phi' (\vdash A_3) = \left[ \vdash A_3 \right],$$
$$\text{Skip}$$

$$\phi' \circ \phi' (\vdash G : (A_1 \land A_2) \land A_3) = \left[ \vdash G : A_1 ; \vdash G : A_2 ; \vdash G : A_3 \right],$$
$$\text{Split}(G, \text{Split}(G, \text{Skip}, \text{Skip}), \text{Skip})$$
Contributions

We implemented:
- FOL certifying transformations
- OCaml checker
- chaining of certificate
- full composition of certifying transformations

Our method provides:
- abstraction
- efficiency
- certifiable checker
- modularity
Future works

Improve integration into Why3:
- instrument existing Why3 transformations
- correction of ccheck
- other ways to compose transformations

Goal: improve trust in Why3, make certification of Why3 transformations easier and make transformations usable in other contexts

→ gitlab.inria.fr/why3/why3/tree/certif