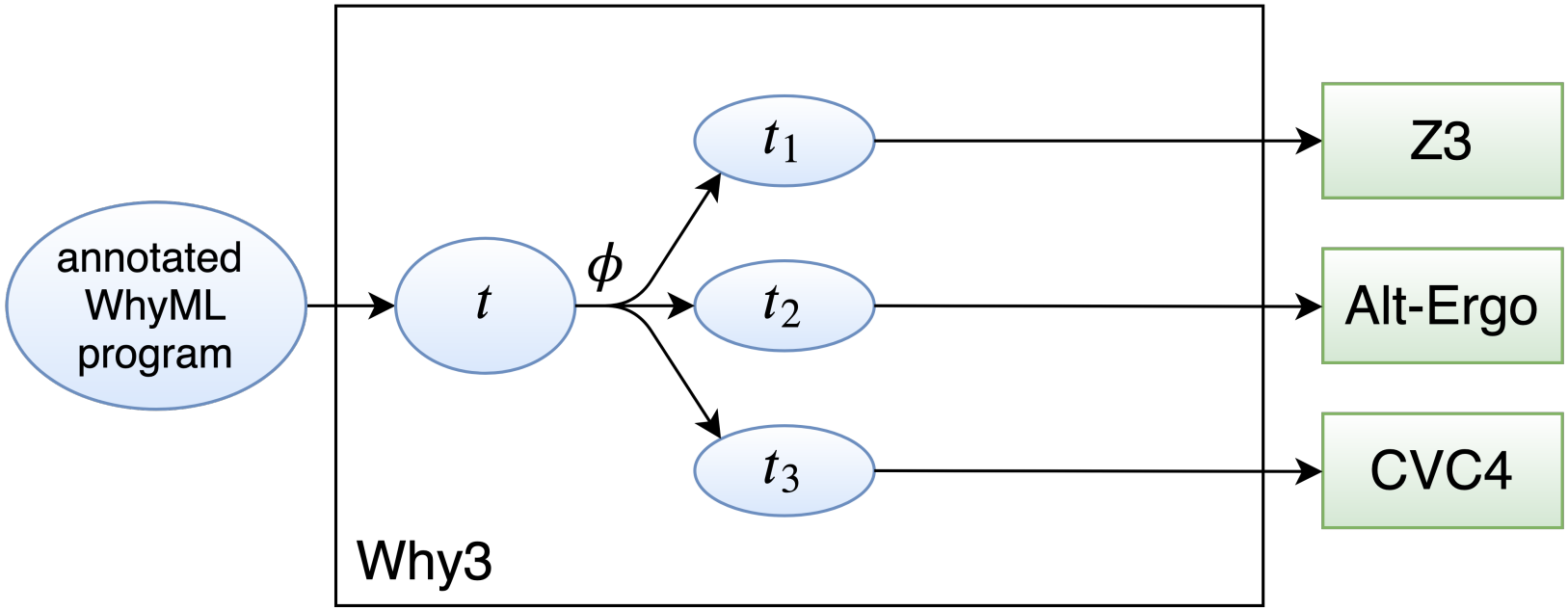
How to explain Why3

UPSCaLe WG

4th june 2019

Quentin Garchery
LRI, Univ. Paris-Sud

Overview of Why3



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?

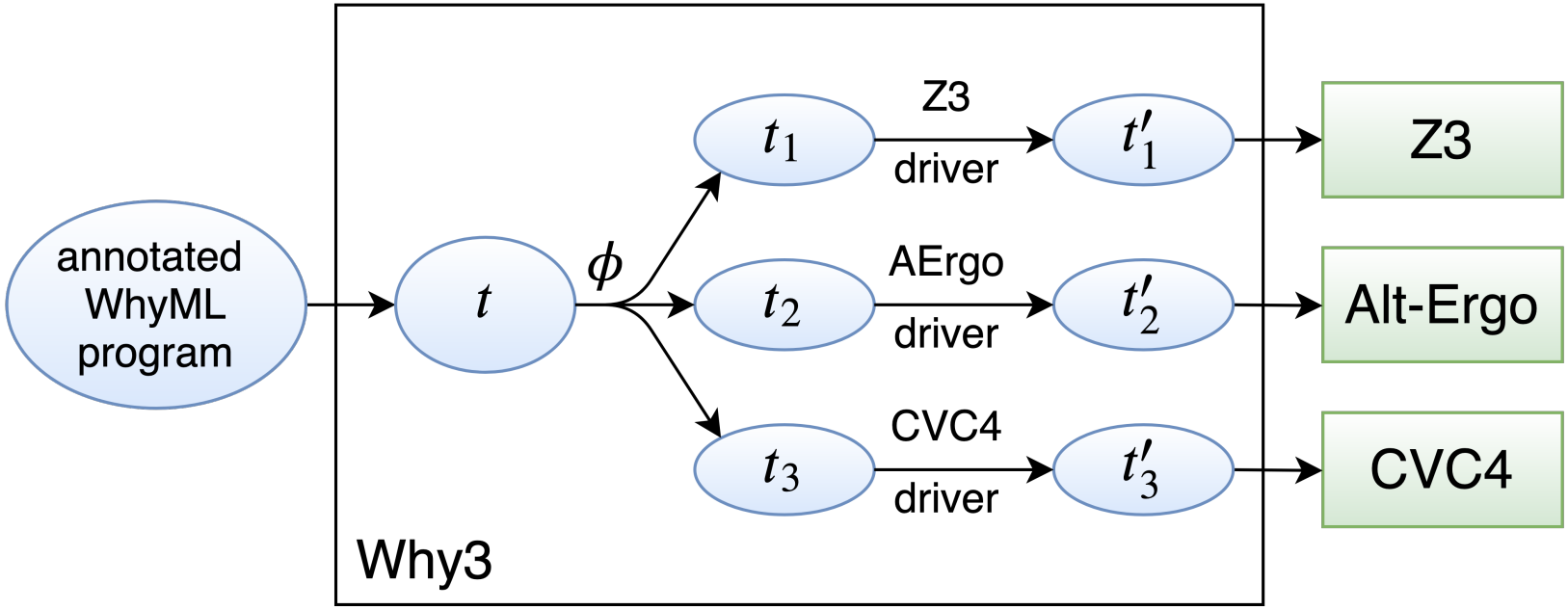
Automatic provers	SMTCoq
Transformations	subject of this talk
Task generation	

Motivation : improve trust

Trust Why3?

Automatic provers	SMTCoq
Transformations	subject of this talk
Task generation	future works

Drivers for automatic provers



Skeptical and autarkic approaches

Autarkic approach
Transformations are correct w.r.t. every task
$\rightarrow$ prove the code of the transformation

Skeptical approach
Transformations are correct on tasks they are applied to
$\rightarrow$ 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

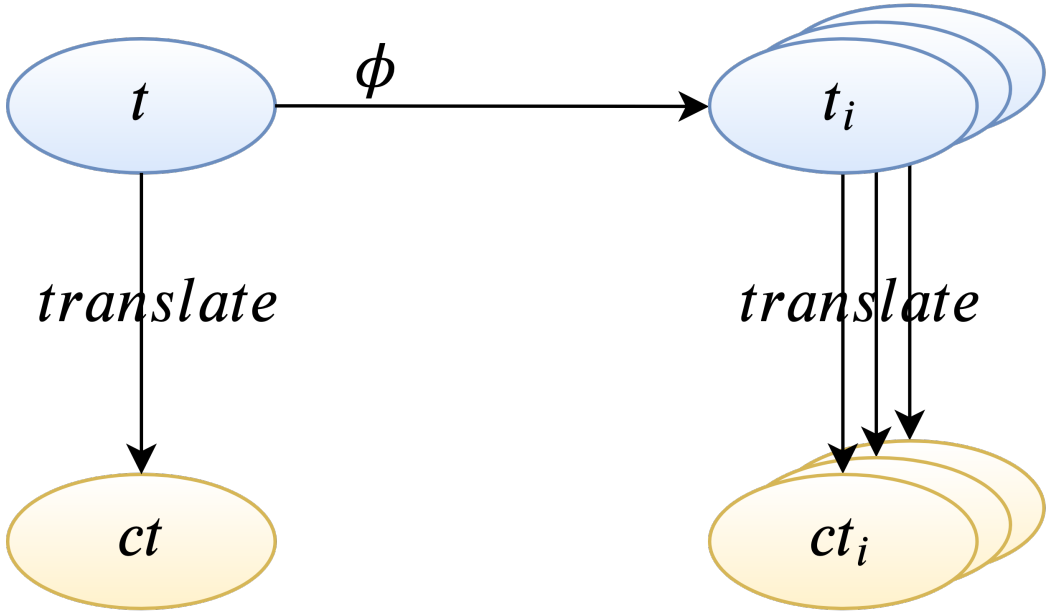
New representation *ctask* :

- no memoization
- allow multiple goals
- locally nameless representation of terms

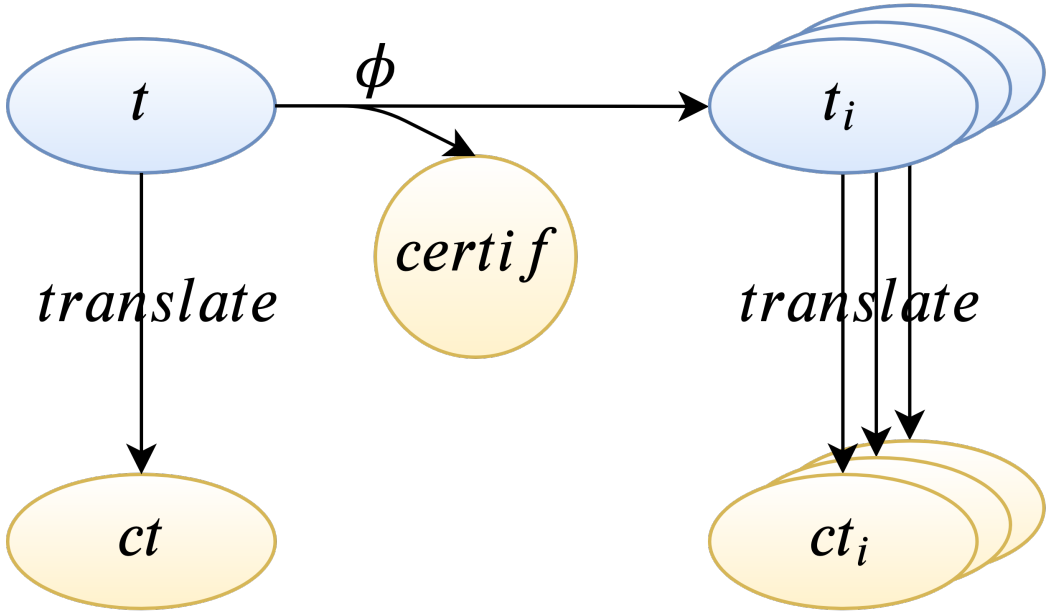
Write a translation from a why3 *task* to a *ctask*

Use the notation $\Gamma \vdash \Delta$ for both

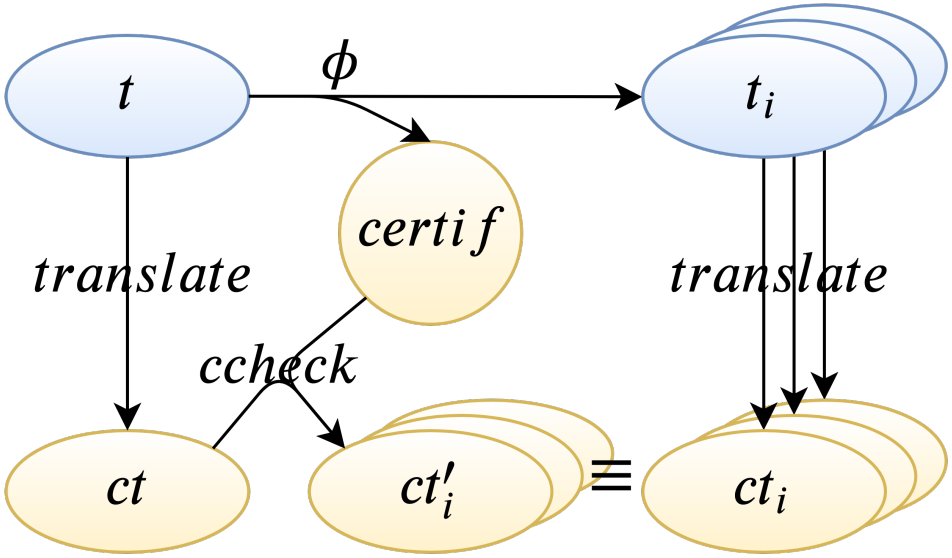
Skeptical approach with *ctask*



Skeptical approach with *ctask*



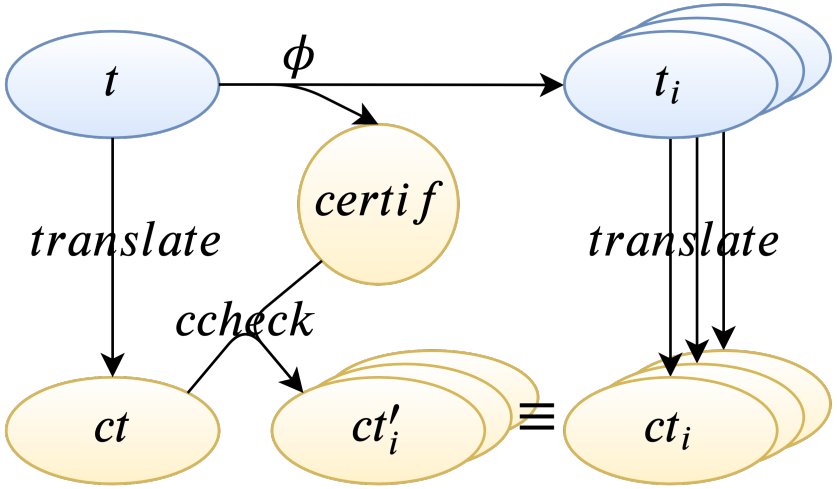
Skeptical approach with *ctask*



Trust with checker

What is trusted still :

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



Originality of the method

Works when task is left open

Diagram commutation catches bugs :

ϕ ($\vdash A_1 \wedge A_2$) returns *certif* and $\begin{array}{l} \vdash A_1 \\ \vdash A_1 \end{array}$

ccheck certif ($\vdash A_1 \wedge A_2$) does not return $\begin{array}{l} \vdash A_1 \\ \vdash A_1 \end{array}$

Certificates and checker

We define :

- certificates *certif*
  $\rightarrow$ skeleton of a sequent calculus derivation
- a function `ccheck : certif -> ctask -> ctask list`
  $\rightarrow$ *ctask* is the root
  $\rightarrow$ recursively computes the nodes of *certif*

Example :

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

We then write :

$$\frac{}{\Gamma, H : A \vdash \Delta, G : A}\ \text{Assumption (H, G)}$$

Chaining certificates

Compute :

$$\begin{aligned} &\mathit{ccheck}\ (\mathit{Split}\ (G, c_1, c_2))\ (\vdash G : A_1 \wedge A_2) \\ &\qquad \equiv \mathit{ccheck}\ c_1\ (\vdash G : A_1)\ @\ \mathit{ccheck}\ c_2\ (\vdash G : A_2) \\ &\qquad \equiv \mathit{lt}_1\ @\ \mathit{lt}_2 \end{aligned}$$

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

Represent this computation as a derivation :

$$\dfrac{\dfrac{\mathit{lt}_1}{\vdash G : A_1}\ c_1 \qquad \dfrac{\mathit{lt}_2}{\vdash G : A_2}\ c_2}{\vdash G : A_1 \wedge A_2}\ \mathit{Split}\ (G, c_1, c_2)$$

Chaining certificates : drinker's theorem

Suppose we have $a \in A$ and pose :

$$dr := \exists x.(P\,x \Rightarrow \forall y.P\,y)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{H_1 : P\,a, H_2 : P\,y \vdash G_1 : dr, G_2 : P\,y, G_3 : \forall y.P\,y}\,\mathit{Assumption}(H_2, G_2)}{H_1 : P\,a \vdash G_1 : dr, G_2 : P\,y, G_3 : P\,y \Rightarrow \forall y.P\,y}\,\mathit{Intro}(G_3, H_2, \_)}{H_1 : P\,a \vdash G_1 : dr, G_2 : P\,y}\,\mathit{InstQuant}(G_1, y, G_3, \_)}{H_1 : P\,a \vdash G_1 : dr, G_2 : \forall y.P\,y}\,\mathit{IntroQuant}(G_2, \_)}{\vdash G_1 : dr, G_2 : P\,a \Rightarrow \forall y.P\,y}\,\mathit{Intro}(G_2, H_1, \_)}{\vdash G_1 : dr}\,\mathit{InstQuant}(G_1, a, G_2, \_)$$

Composing certifying transformations

Certificate *Skip* when task is unchanged

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

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

$$\dfrac{\dfrac{\dfrac{\vdash G : A_1}{\vdash G : A_1}\,\text{Skip} \qquad \dfrac{\vdash G : A_2}{\vdash G : A_2}\,\text{Skip}}{\vdash G : A_1 \wedge A_2}\,\mathit{Split}(G, \mathit{Skip}, \mathit{Skip}) \qquad \dfrac{\vdash G : A_3}{\vdash G : A_3}\,\text{Skip}}{\vdash G : (A_1 \wedge A_2) \wedge A_3}\,\mathit{certif}$$

Composing certifying transformations

Write ϕ as a composition of transformations ?

$$\phi'(\vdash G : (A_1 \wedge A_2) \wedge A_3) = [\vdash G : A_1 \wedge A_2; \vdash A_3], \mathit{Split}(G, \mathit{Skip}, \mathit{Skip})$$

$$\phi'(\vdash A_1 \wedge A_2) = [\vdash G : A_1; \vdash G : A_2], \mathit{Split}(G, \mathit{Skip}, \mathit{Skip})$$

$$\phi'(\vdash A_3) = [\vdash A_3], \mathit{Skip}$$

$$\phi' \circ \phi'(\vdash G : (A_1 \wedge A_2) \wedge A_3) = [\vdash G : A_1; \vdash G : A_2; \vdash G : A_3], \mathit{Split}(G, \mathit{Split}(G, \mathit{Skip}, \mathit{Skip}), \mathit{Skip})$$

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`