Des transformations logiques passent leur certificat

Journées Francophones des Langages Applicatifs

Vendredi 31 janvier 2020

Quentin Garchery Chantal Keller

Claude Marché Andrei Paskevich

Vue d'ensemble

Contexte : vérification de programmes **sûre**

↳ noyau réduit

↳ certification a posteriori

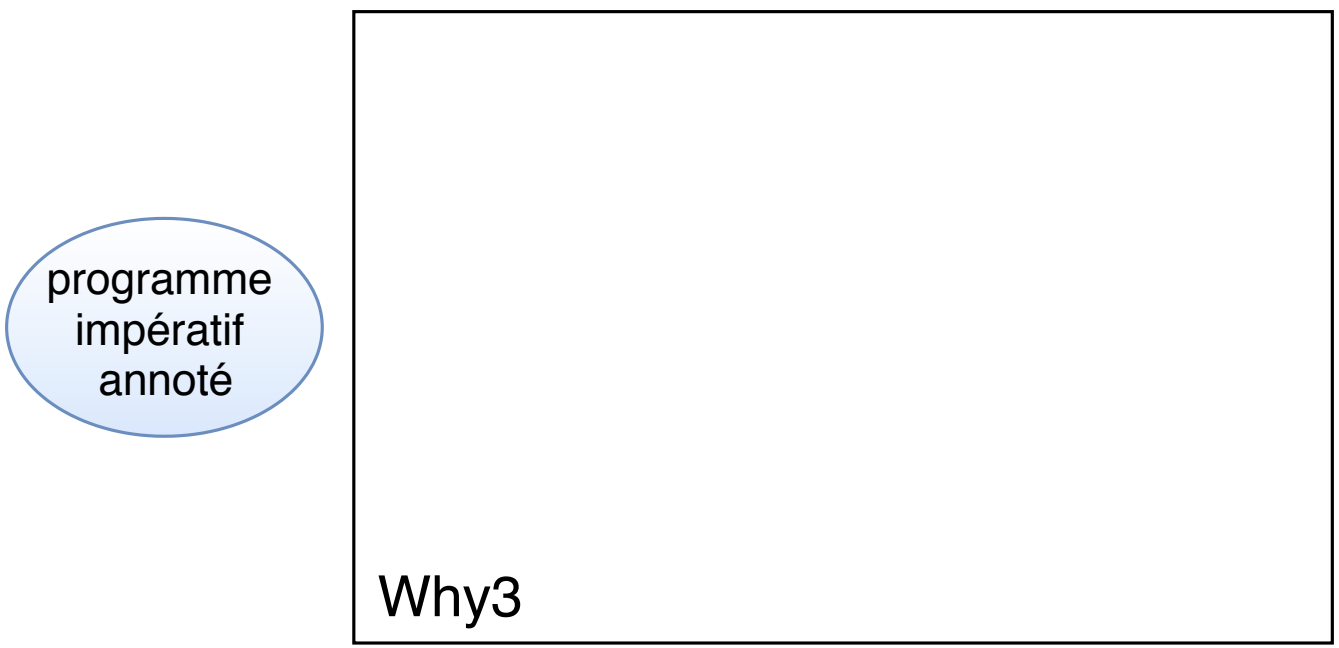
Comment vérifier les transformations de Why3 

$\Longrightarrow$ production et vérification de certificats

Why3 et motivations

Fonctionnement de Why3 :

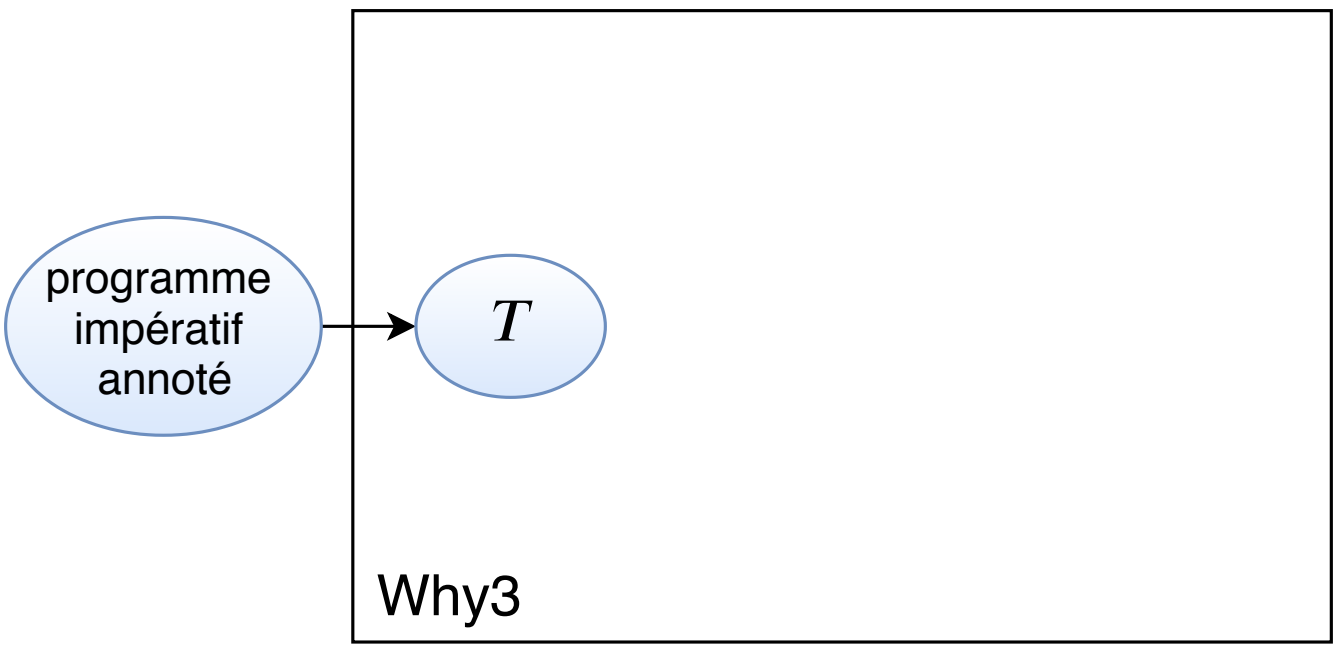
- création de la tâche initiale
- transformations logiques
- appel aux prouveurs



Why3 et motivations

Fonctionnement de Why3 :

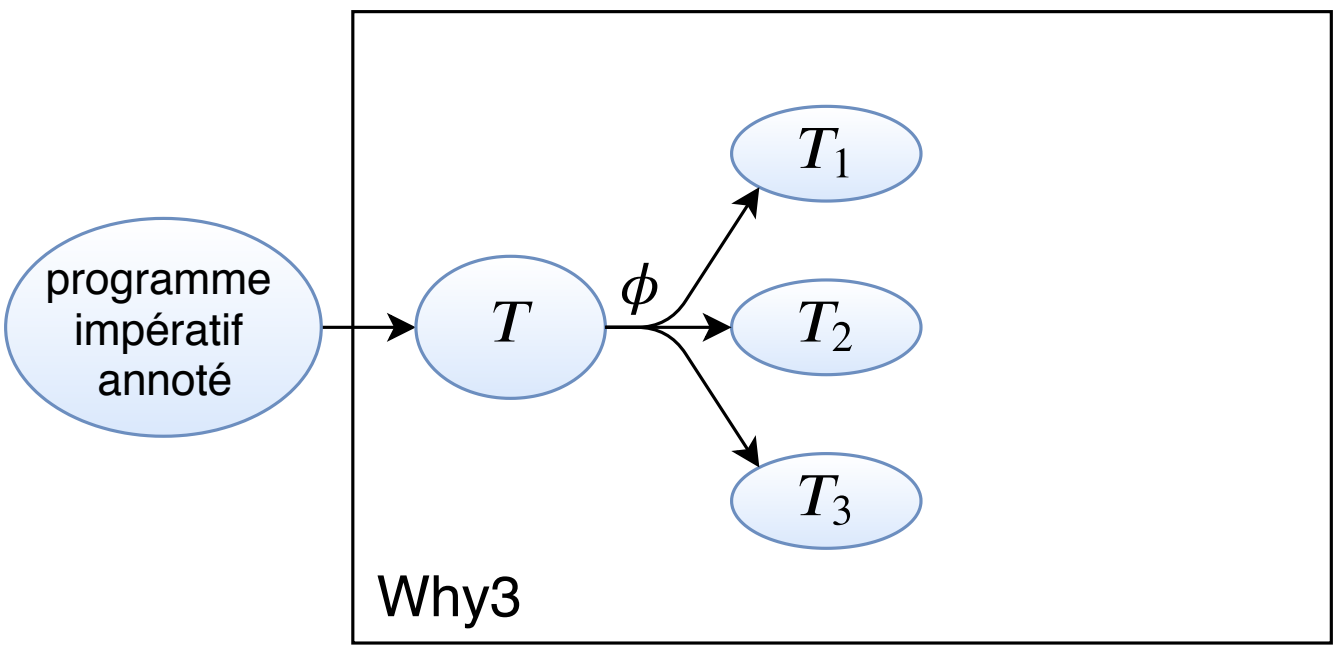
- création de la tâche initiale
- transformations logiques
- appel aux prouveurs



Why3 et motivations

Fonctionnement de Why3 :

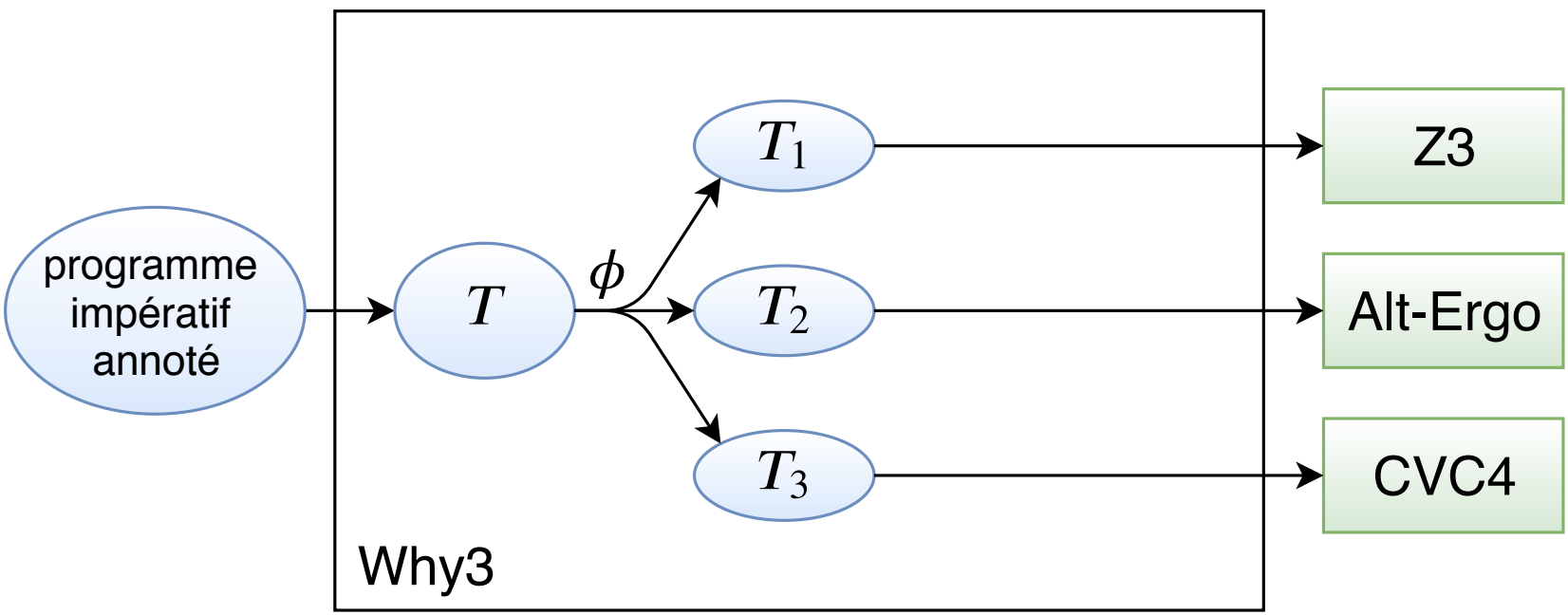
- création de la tâche initiale
- transformations logiques
- appel aux prouveurs



Why3 et motivations

Fonctionnement de Why3 :

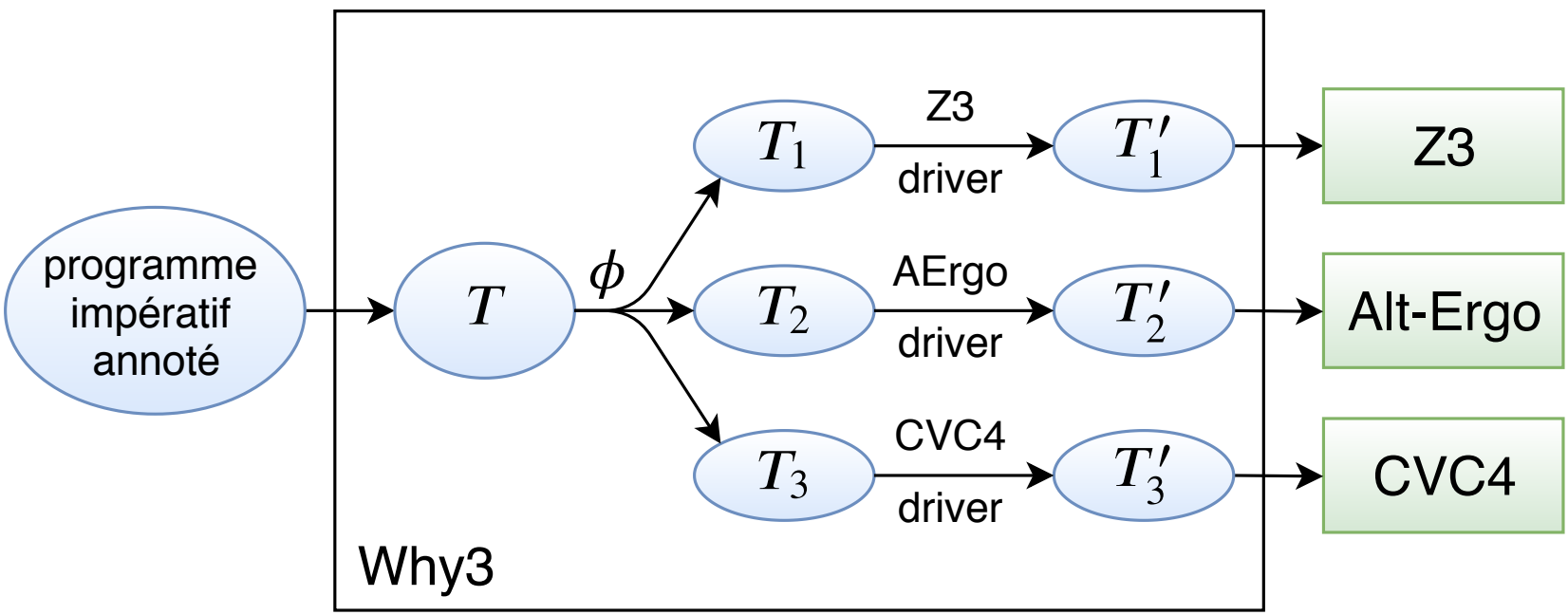
- création de la tâche initiale
- transformations logiques
- appel aux prouveurs



Why3 et motivations

Fonctionnement de Why3 :

- création de la tâche initiale
- transformations logiques
- appel aux prouveurs



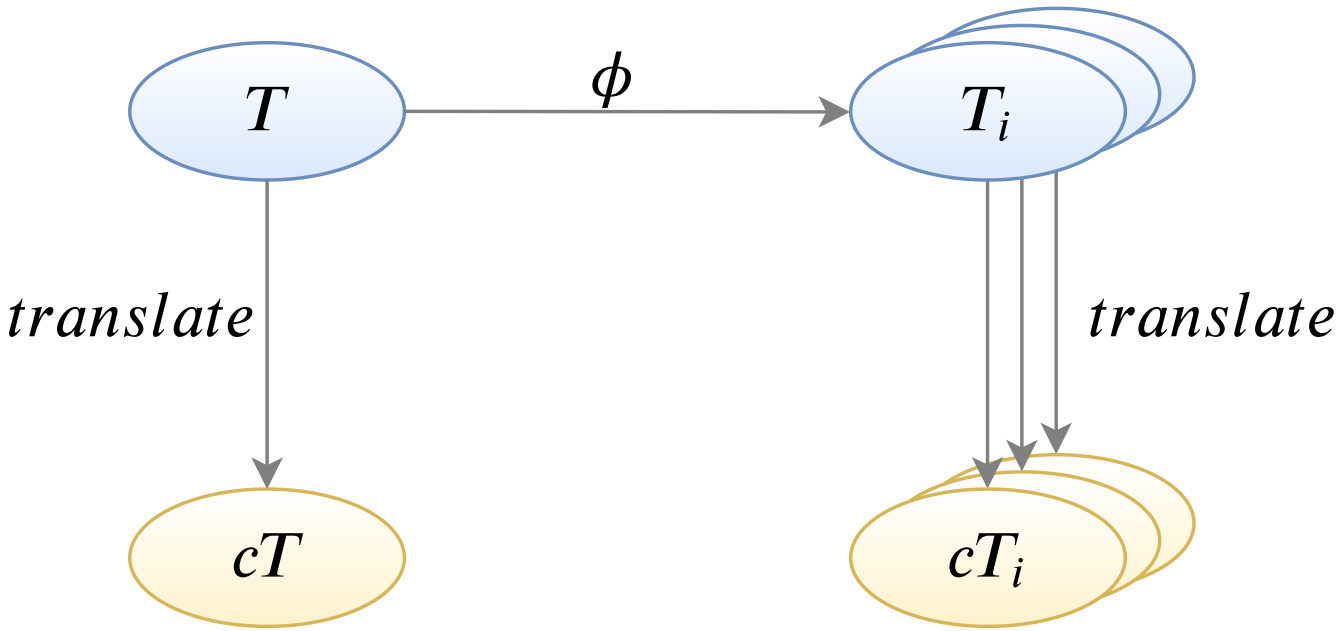
Plan

Abstraction des tâches : motivations

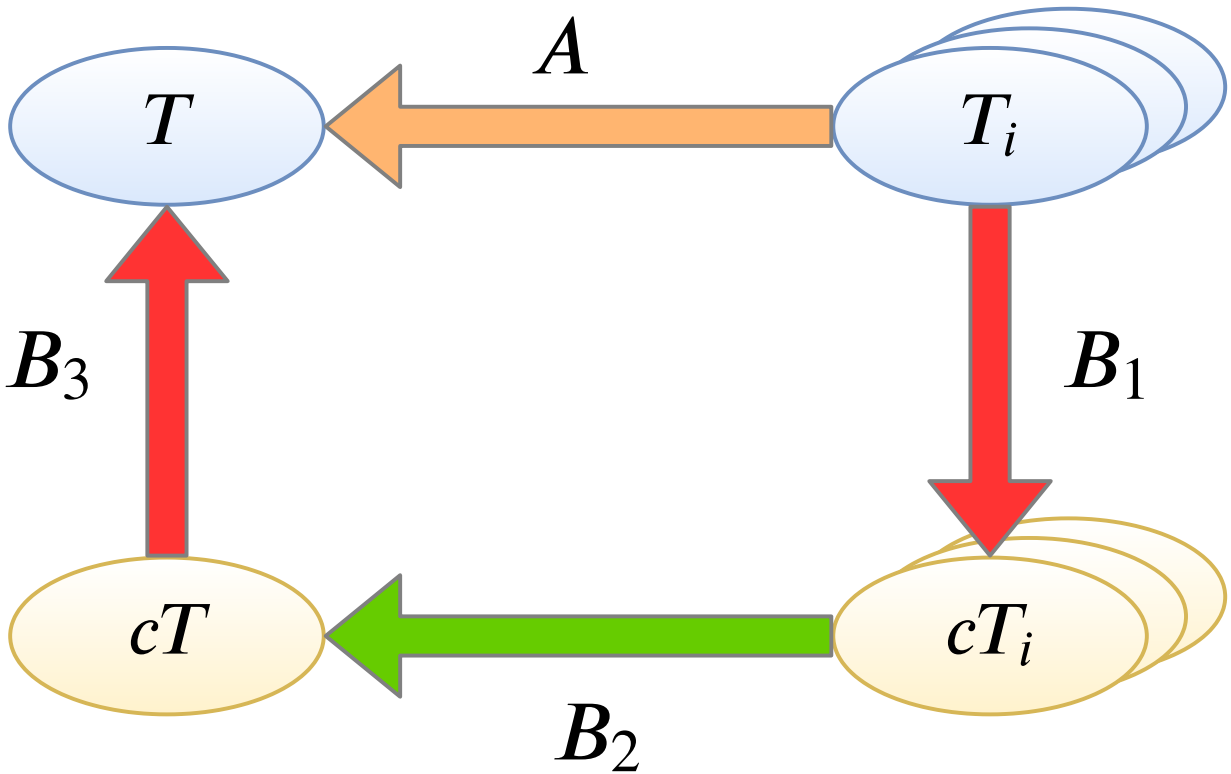
Tâches de preuves avec méta-données

$\hookrightarrow$ extraction du contenu logique

$H_1 : A_1, ..., H_m : A_m \vdash G_1 : B_1, ..., G_n : B_n$



Abstraction des tâches : correction

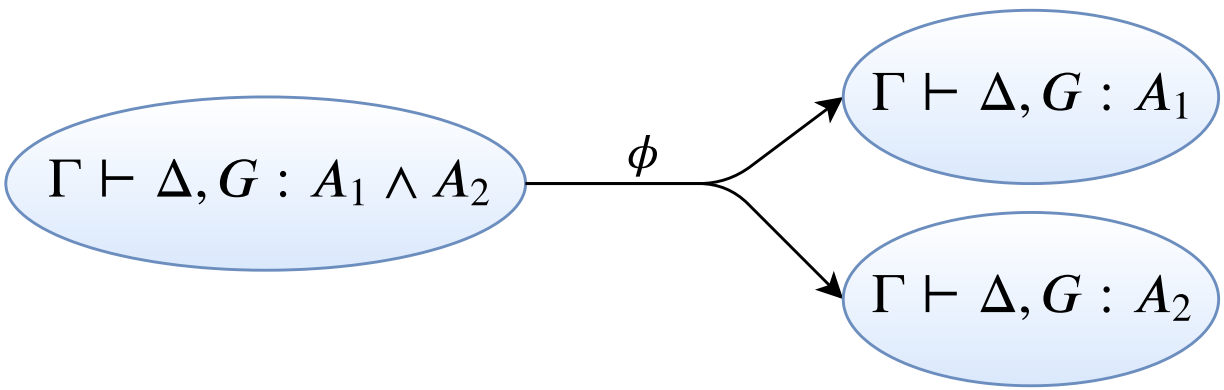


$\Longrightarrow$ nécessite l'équisatisfiabilité entre une tâche et son abstraction

Définition des certificats

```
type certif :=
 | Split of ident * certif * certif
 | Axiom of ident * ident
 | Hole
 | ...
```

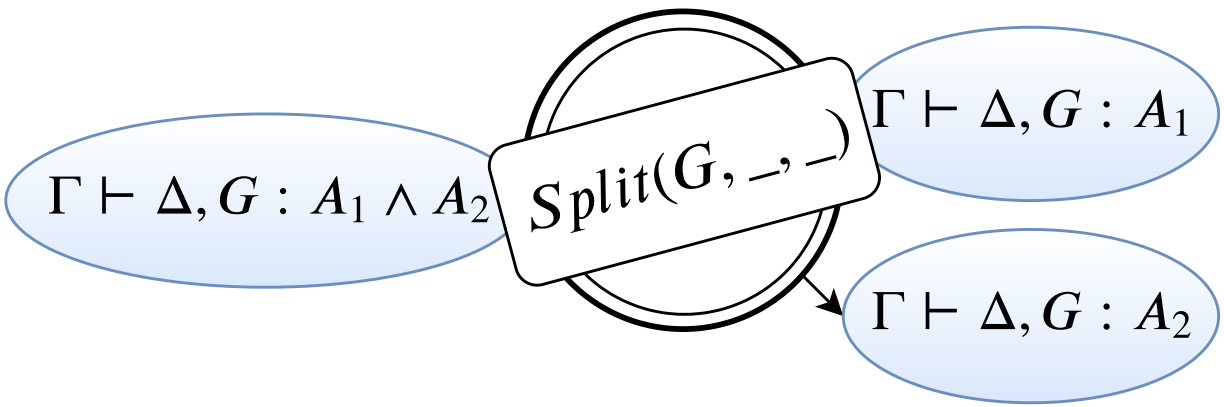
Un constructeur $\Leftrightarrow$ Une transformation élémentaire



Définition des certificats

```
type certif :=
 | Split of ident * certif * certif
 | Axiom of ident * ident
 | Hole
 | ...
```

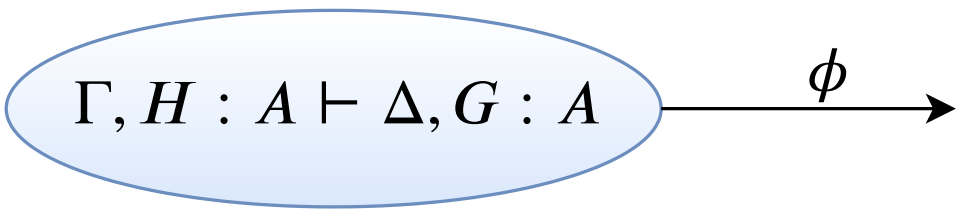
Un constructeur $\Leftrightarrow$ Une transformation élémentaire



Définition des certificats

```
type certif :=
 | Split of ident * certif * certif
 | Axiom of ident * ident
 | Hole
 | ...
```

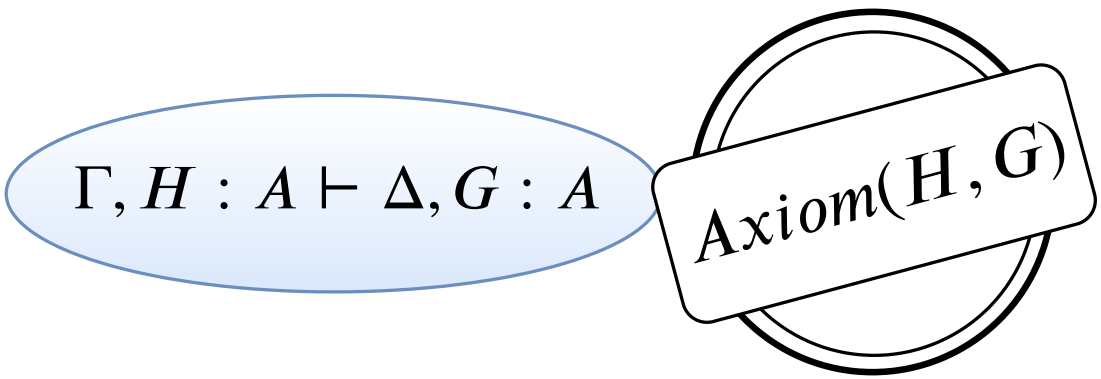
Un constructeur $\Leftrightarrow$ Une transformation élémentaire



Définition des certificats

```
type certif :=
 | Split of ident * certif * certif
 | Axiom of ident * ident
 | Hole
 | ...
```

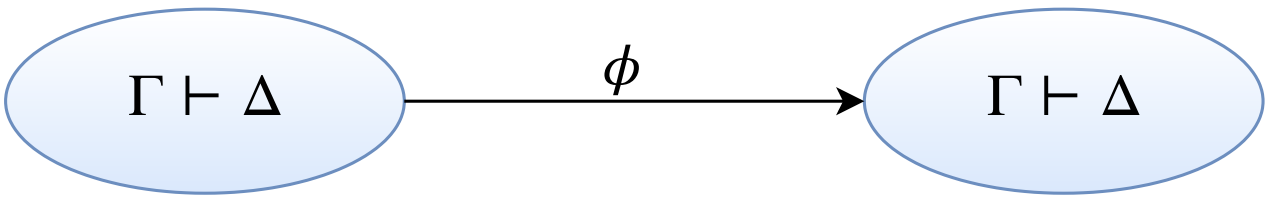
Un constructeur $\Leftrightarrow$ Une transformation élémentaire



Définition des certificats

```
type certif :=
 | Split of ident * certif * certif
 | Axiom of ident * ident
 | Hole
 | ...
```

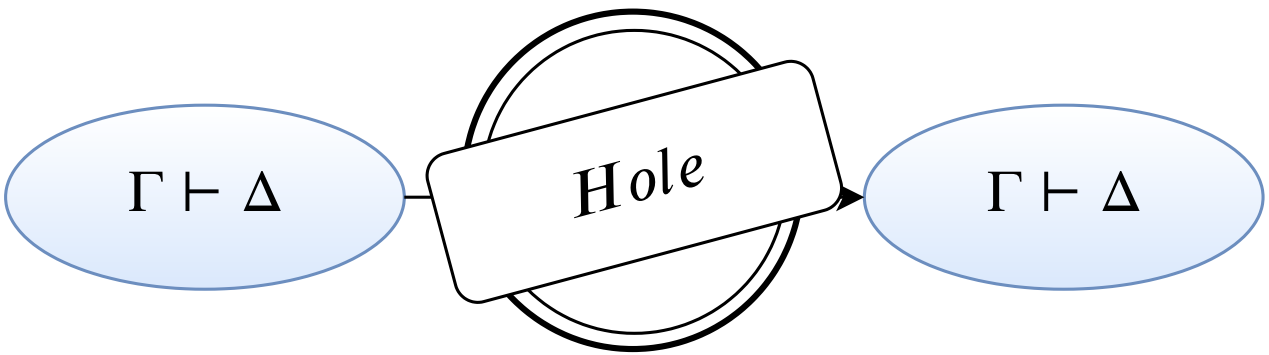
Un constructeur $\Leftrightarrow$ Une transformation élémentaire



Définition des certificats

```
type certif :=
 | Split of ident * certif * certif
 | Axiom of ident * ident
 | Hole
 | ...
```

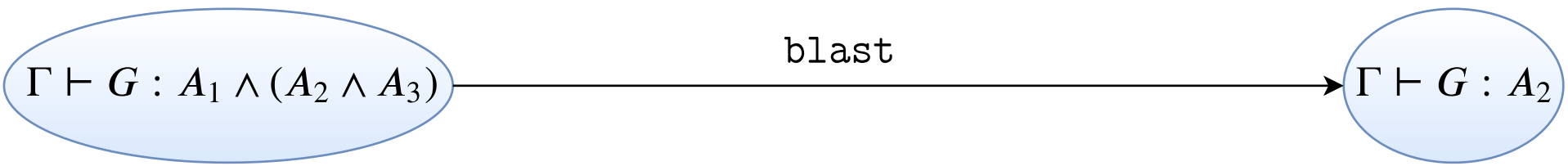
Un constructeur $\Leftrightarrow$ Une transformation élémentaire



Exemple : transformation `blast`

Posons : $\Gamma := H_1 : A_1, H_3 : A_3$

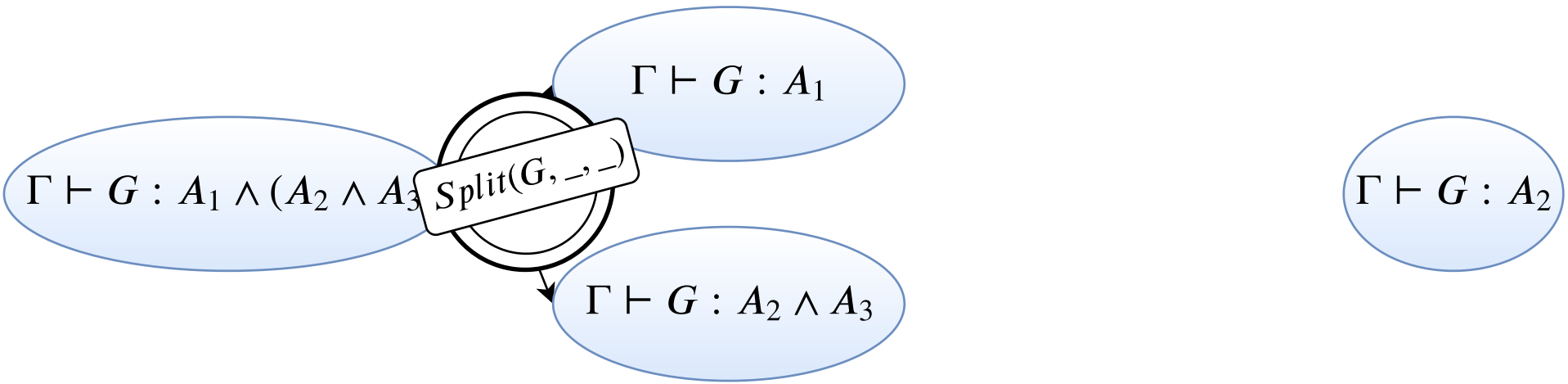
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

Posons : $\Gamma := H_1 : A_1, H_3 : A_3$

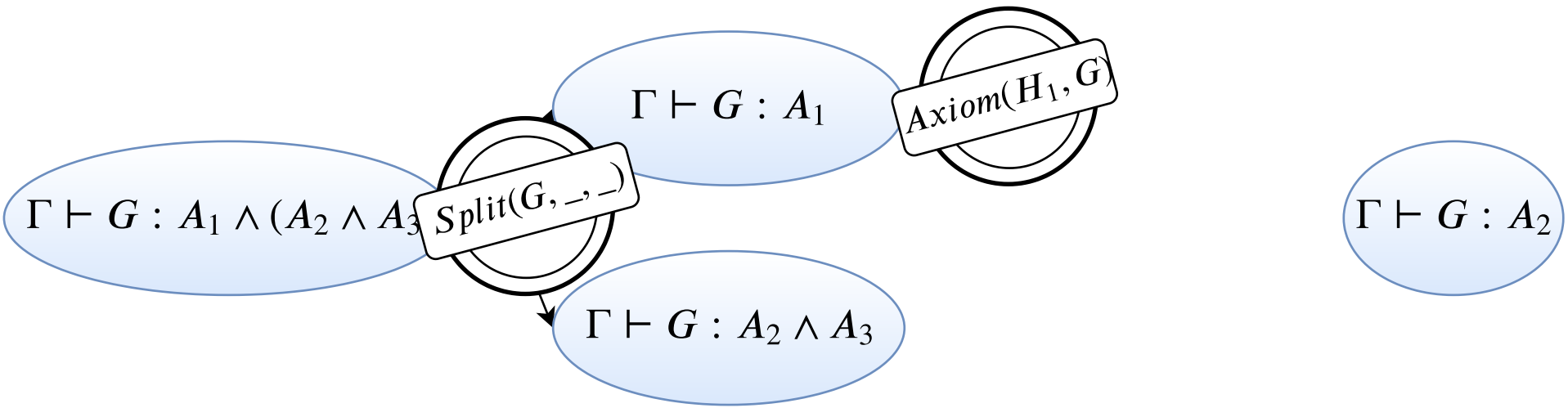
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

Posons : $\Gamma := H_1 : A_1, H_3 : A_3$

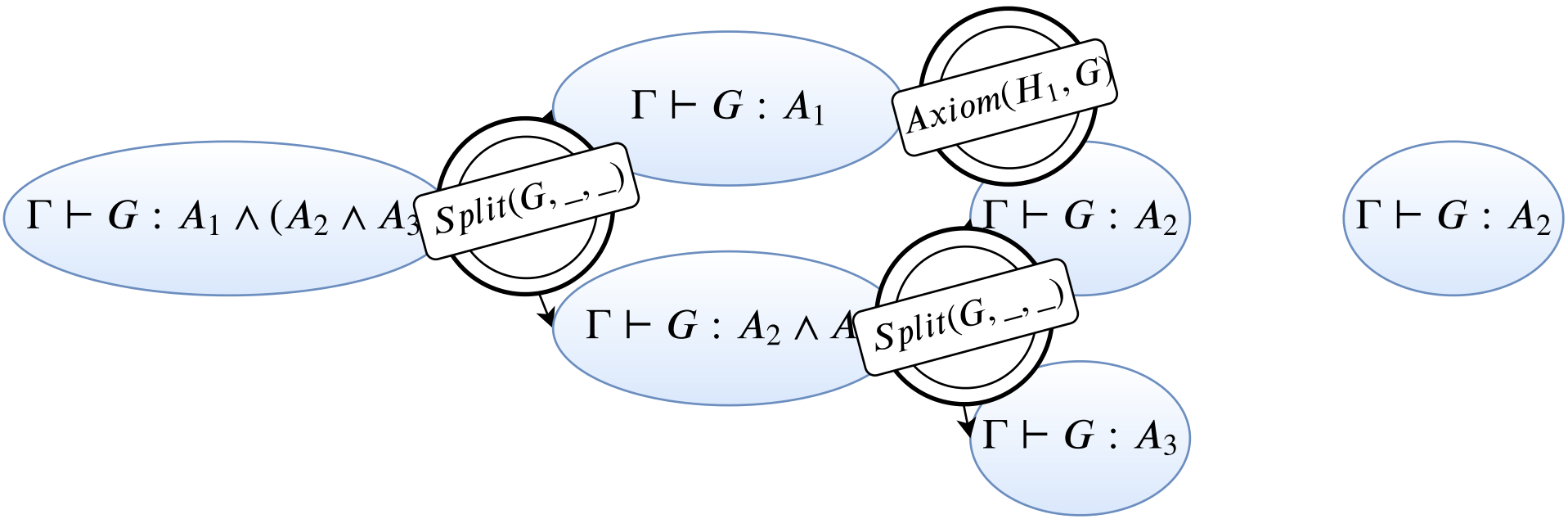
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

Posons : $\Gamma := H_1 : A_1, H_3 : A_3$

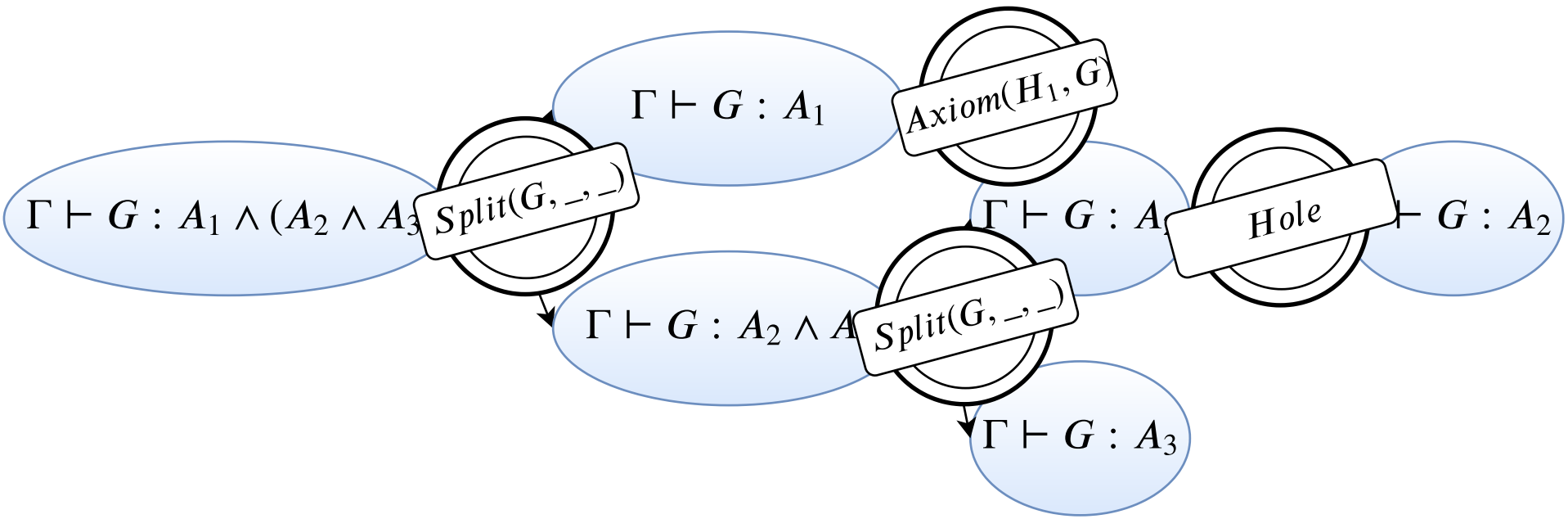
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

Posons : $\Gamma := H_1 : A_1, H_3 : A_3$

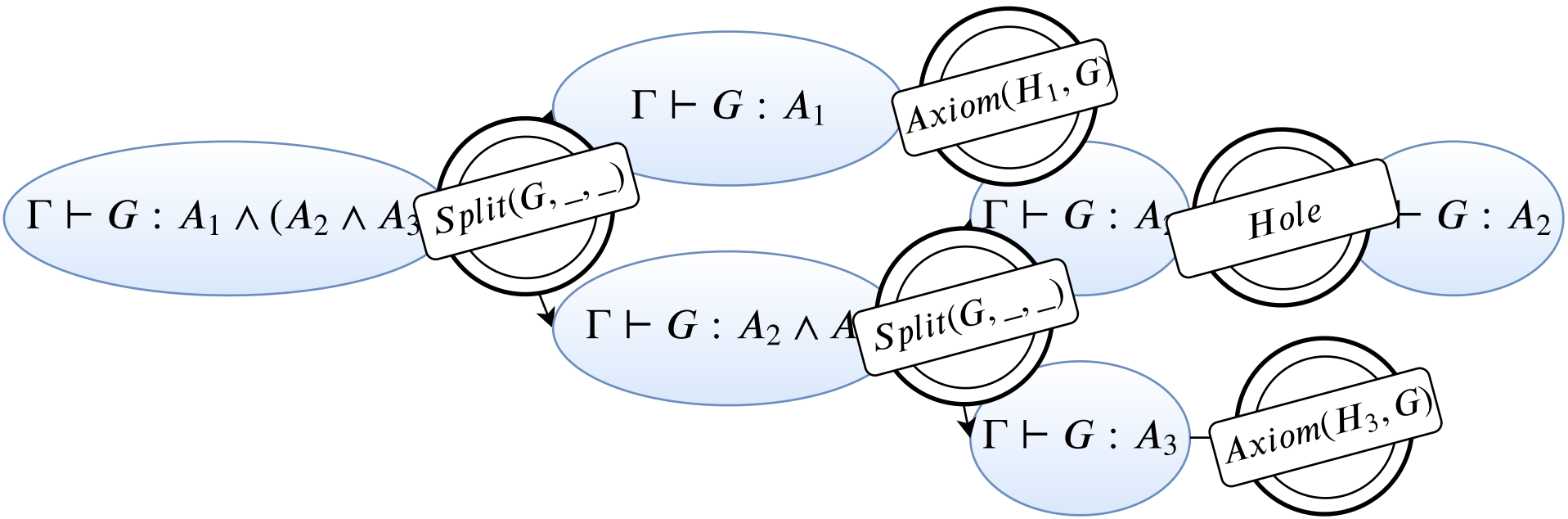
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

Posons : $\Gamma := H_1 : A_1, H_3 : A_3$

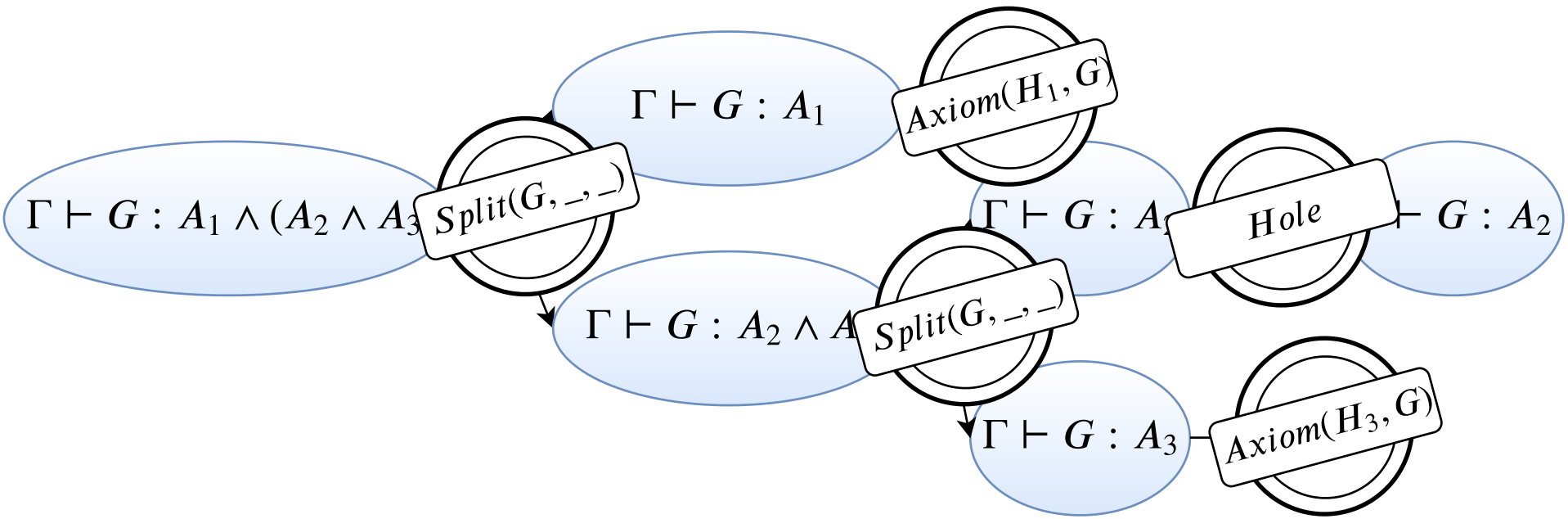
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

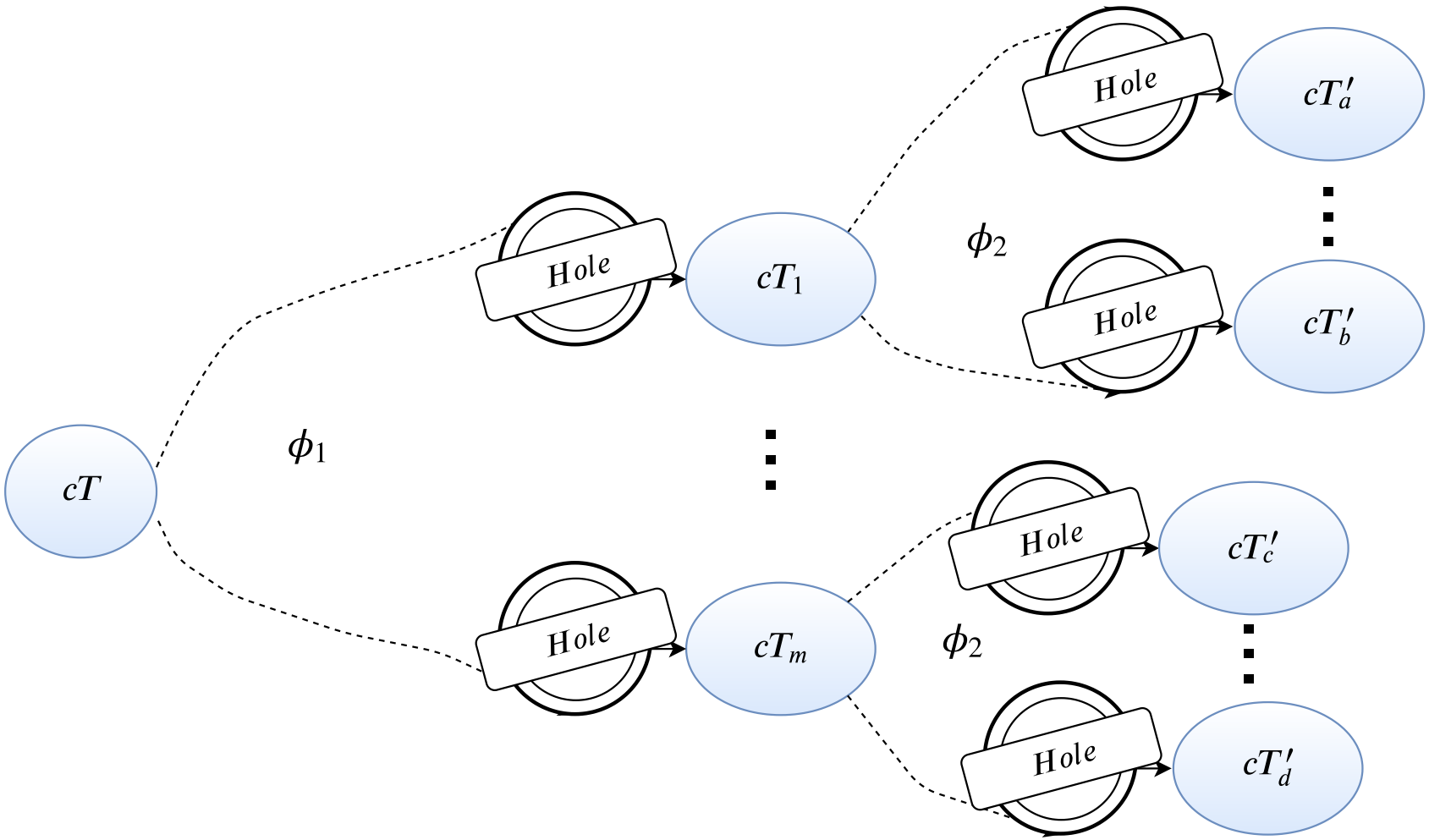
Posons : $\Gamma := H_1 : A_1, H_3 : A_3$

$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$

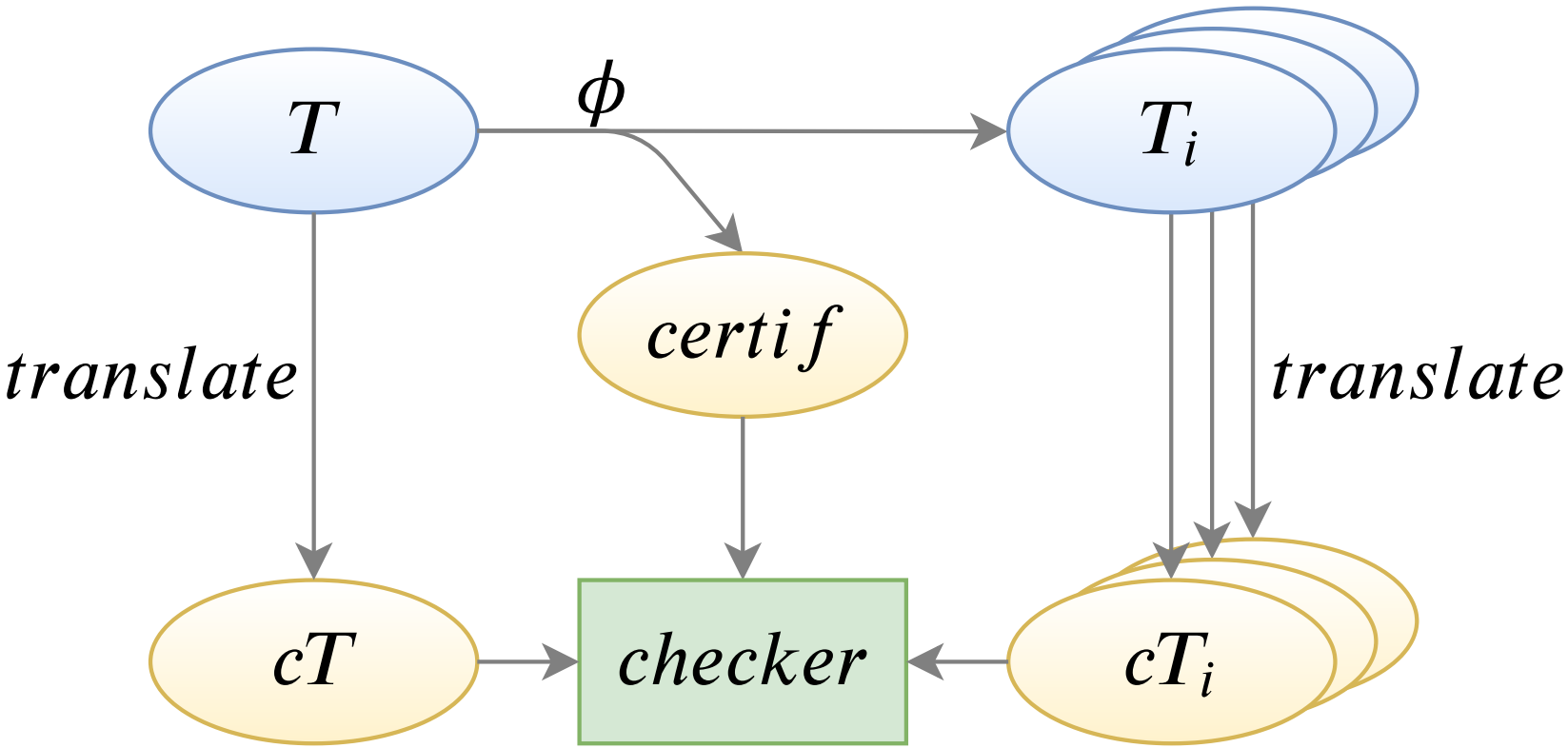


$Split\ (G, Axiom(H_1, G), Split(G, Hole, Axiom(H_3, G)))$

Composition de transformations certifiantes

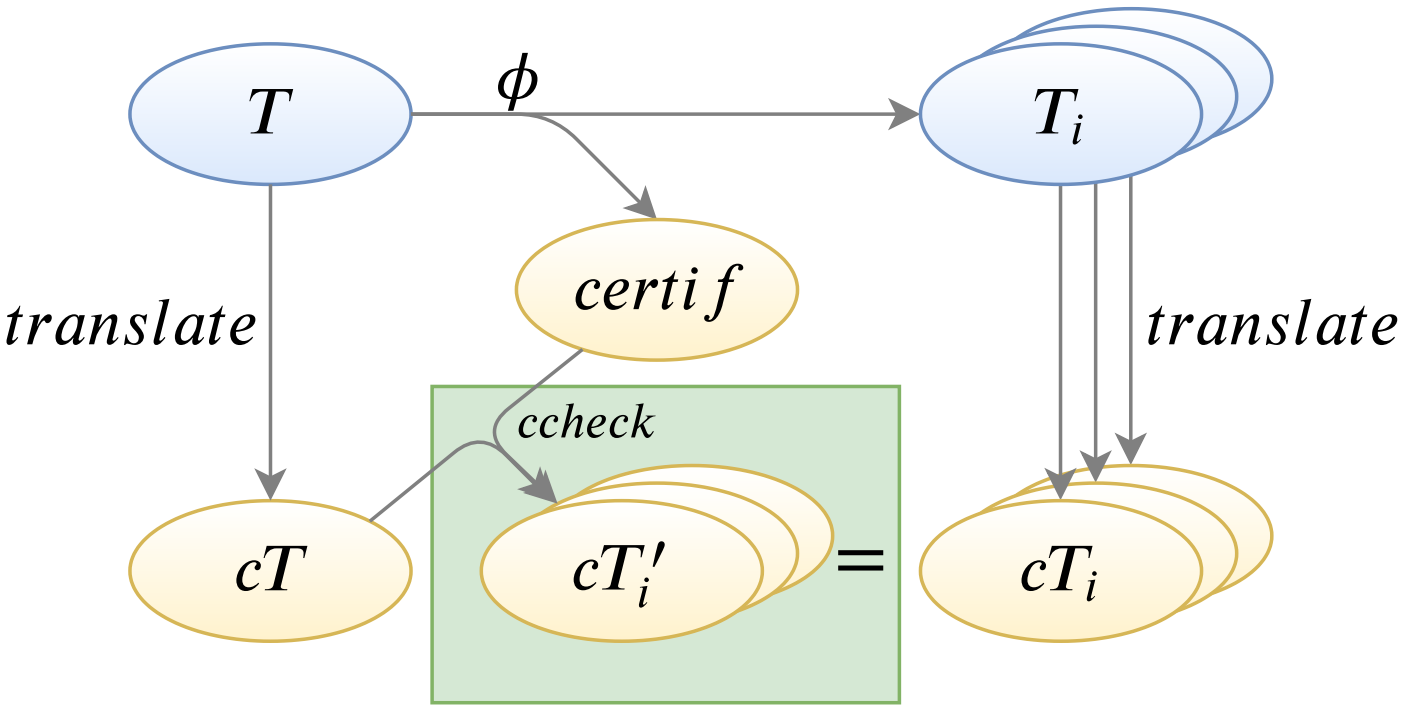


Implémenter un vérificateur



Approche réflexive

ccheck : `certif -> task -> task list`



Réalisation de *ccheck*

Exemple de `blast` :

$$\Gamma := H_1 : A_1, H_3 : A_3$$
$$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$$
$$\mathit{certif} := \mathit{Split}\ (G, \mathit{Axiom}(H_1, G), \mathit{Split}(G, \mathit{Hole}, \mathit{Axiom}(H_3, G)))$$

Réalisation de *ccheck*

Exemple de `blast` :

$$\Gamma := H_1 : A_1, H_3 : A_3$$
$$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$$
$$\mathit{certif} := \mathit{Split}\ (G, \mathit{Axiom}(H_1, G), \mathit{Split}(G, \mathit{Hole}, \mathit{Axiom}(H_3, G)))$$

On calcule :

$$\begin{aligned}&\mathit{ccheck}\ \mathit{certif}\ T\\ &\quad \equiv \mathit{ccheck}\ (\mathit{Axiom}(H_1, G))\ (\Gamma \vdash G : A_1)\\ &\qquad @\ \mathit{ccheck}\ (\mathit{Split}(G, \mathit{Hole}, \mathit{Axiom}(H_3, G)))\ (\Gamma \vdash G : A_2 \wedge A_3)\end{aligned}$$

Réalisation de *ccheck*

Exemple de `blast` :

$$\Gamma := H_1 : A_1, H_3 : A_3$$
$$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$$
$$certif := Split\ (G, Axiom(H_1, G), Split(G, Hole, Axiom(H_3, G)))$$

On calcule :

$$\begin{aligned} & ccheck\ certif\ T \\ & \quad \equiv ccheck\ (Axiom(H_1, G))\ (\Gamma \vdash G : A_1) \\ & \qquad @\ ccheck\ (Split(G, Hole, Axiom(H_3, G)))\ (\Gamma \vdash G : A_2 \wedge A_3) \\ & \quad \equiv [] \\ & \qquad @\ ccheck\ Hole\ (\Gamma \vdash G : A_2) \\ & \qquad @\ ccheck\ (Axiom(H_3, G))\ (\Gamma \vdash G : A_3) \end{aligned}$$

Réalisation de *ccheck*

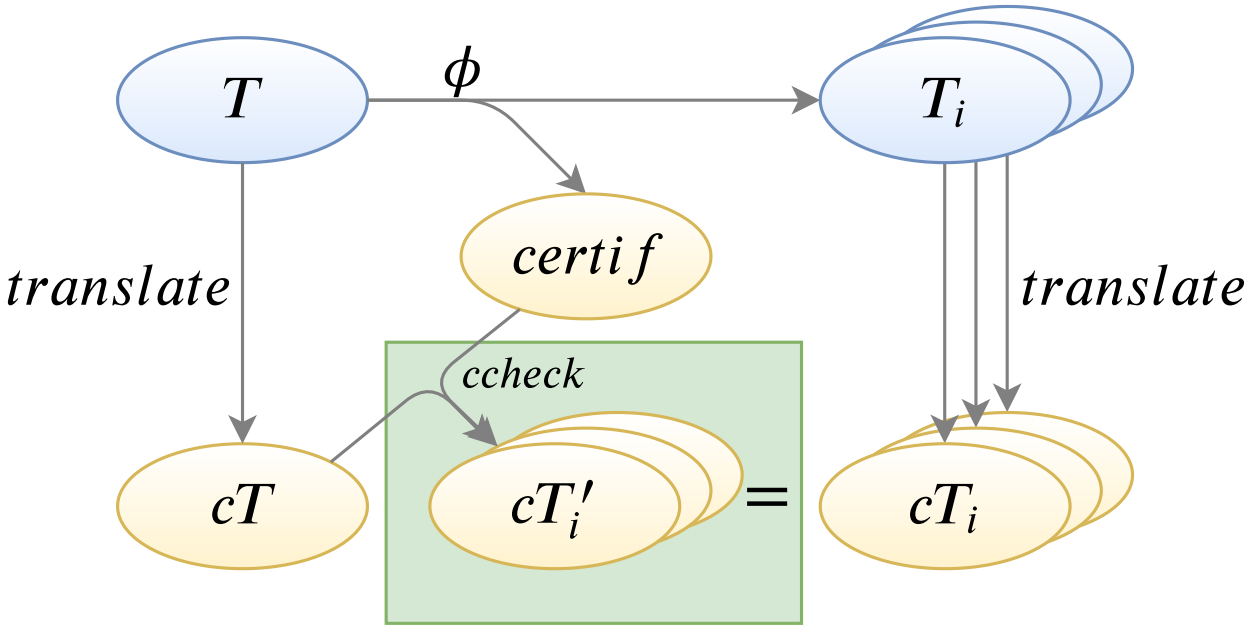
Exemple de `blast` :

$$\Gamma := H_1 : A_1, H_3 : A_3$$
$$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$$
$$certif := Split\ (G, Axiom(H_1, G), Split(G, Hole, Axiom(H_3, G)))$$

On calcule :

$$\begin{aligned} & ccheck\ certif\ T \\ & \quad \equiv ccheck\ (Axiom(H_1, G))\ (\Gamma \vdash G : A_1) \\ & \qquad @\ ccheck\ (Split(G, Hole, Axiom(H_3, G)))\ (\Gamma \vdash G : A_2 \wedge A_3) \\ & \quad \equiv [] \\ & \qquad @\ ccheck\ Hole\ (\Gamma \vdash G : A_2) \\ & \qquad @\ ccheck\ (Axiom(H_3, G))\ (\Gamma \vdash G : A_3) \\ & \quad \equiv [\Gamma \vdash G : A_2] \end{aligned}$$

Approche réflexive



Correction de *ccheck*

Si le calcul *ccheck certif cT* donne cTi' alors la conjonction des cTi' implique cT.

Principe

Objectif : éviter la vérification formelle du code de *ccheck*

Dedukti : vérificateur de preuve universel, Curry-Howard, conversion extensible

Fonctionnement du vérificateur :

1. générer, à partir des tâches, un type *ty* dans Dedukti
2. générer, à partir du certificat, un terme *t* dans Dedukti
3. vérifier que $t : ty$ dans Dedukti

La deuxième étape n'est pas dans la base de confiance

Traduction d'une tâche de preuve

Encodage de la logique du premier ordre + tiers exclu

Encodage des séquents en Dedukti :

`hyp` A P qui se réécrit en $A \to P$

`goal` B P qui se réécrit en $\neg B \to P$

où $\to$ est la flèche de Dedukti

$$trad(H_1 : A_1, ..., H_m : A_m \vdash G_1 : B_1, ..., G_n : B_n)$$

$$\Vert\Vert$$

$$\texttt{hyp}\ A_1\ \texttt{(...(hyp}\ A_m\ \texttt{(goal}\ B_1\ \texttt{(...(goal}\ B_n\ \texttt{empty)...)}$$

$$\Downarrow$$

$$A_1 \to ... \to A_m \to \neg B_1 \to ... \to \neg B_n \to \bot$$

Traduction des tâches de preuve

$$trad(H_1 : A_1, ..., H_m : A_m \vdash G_1 : B_1, ..., G_n : B_n) = \\ A_1 \to ... \to A_m \to \neg B_1 \to ... \to \neg B_n \to \bot$$

Avec tâche initiale cT et tâches résultantes cT_i :

$$ty = trad\ (cT_1) \to ... \to trad\ (cT_n) \to trad\ (cT)$$

$\Rightarrow$ Utilisation systématique de la flèche de Dedukti

Traduction du certificat

`Split` :

$$\frac{\Gamma \vdash \Delta, G : A \qquad \Gamma \vdash \Delta, G : B}{\Gamma \vdash \Delta, G : A \wedge B}$$

On définit

$$\texttt{split\_goal} : (\neg A \to \bot) \to (\neg B \to \bot) \to \neg(A \wedge B) \to \bot$$

Difficultés rencontrées

- noms, génération et correspondance
- élaboration des certificats

Contributions

Méthode, *générique, à petits pas*, pour certifier des transformations

Développements logiciel :

- format de certificats
- infrastructure commune de vérification
- $\sim$ 15 transformations certifiantes simples
- composition de transformations

Expérimentations :

- `blast`, `rewrite`
- validation des transformations par Dedukti
- principe des tiroirs de Dirichlet

Perspectives

Améliorer la confiance, faciliter la certification de Why3 :

- instrumenter d'autres transformations de Why3
- calcul de WP, génération de la tâche initiale

Permettre l'utilisation des transformations dans d'autres contextes

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