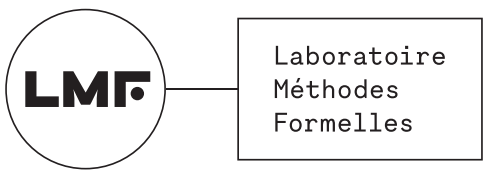
Certification de la transformation de tâches de preuve

Soutenance de thèse de doctorat

25 janvier 2022

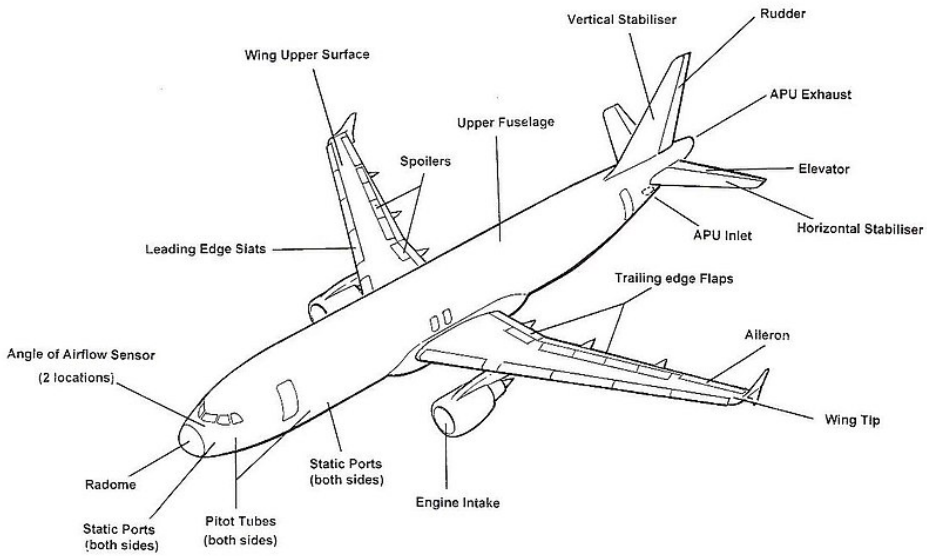


Quentin Garchery

Sûreté et confiance

Objectifs :

- Éviter les erreurs (*bugs*)
- Déterminer et réduire la base de confiance



vérification formelle, qui vérifie le vérificateur ?

Vérification de programmes

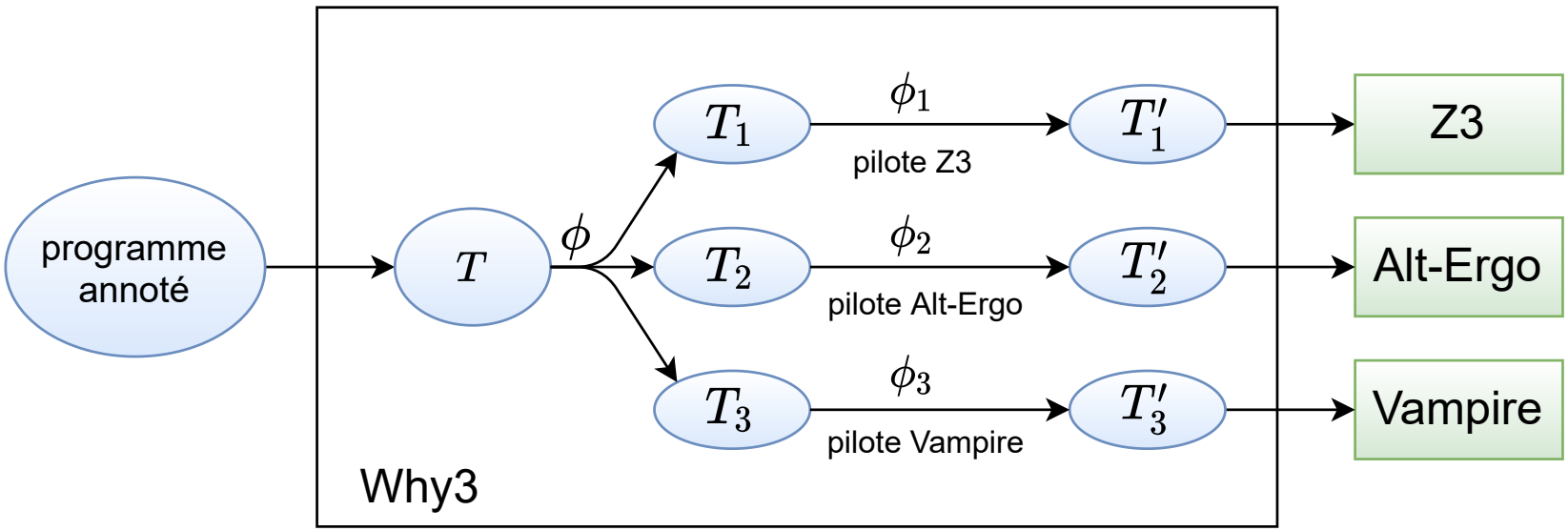
Motivation : systèmes critiques (médecine, aéronautique, ...)

Correction vis-à-vis d'une *spécification*

Vérification déductive : énoncé mathématique de la correction

Outils de vérification déductive (Why3, F*, Dafny, Viper, ...)

Why3 et transformations logiques



Paradigme Why3

1. génération de la tâche de preuve initiale T
2. applications de transformations logiques ϕ
3. appels de prouveurs automatiques

Tâches de preuve

$$\Gamma \vdash \Delta$$

Logique expressive (plus large que celle de Why3) :

- logique classique d'ordre supérieur
- polymorphisme de type
- théorie arithmétique et théorie de l'égalité

$$H : \Pi\alpha.\ \forall x : \alpha.\ \forall l : \mathit{list}(\alpha).\ \mathit{length}\ (\mathit{cons}\ x\ l) \geq 1$$
$$\vdash$$
$$G_1 : \mathit{length}\ (\mathit{cons}\ 1\ \mathit{nil}) \geq 1,$$
$$G_2 : \mathit{length}\ \mathit{nil} \geq 1$$

Exemple : transformation `instantiate`

$$I : y = 2x + 1,$$
$$J : \forall i : int.\ p\ (4i + 1)$$
$$\vdash$$
$$G : p\ (y^2)$$

`instantiate` $J\ (x^2 + x)$

$$I : y = 2x + 1,$$
$$J : \forall i : int.\ p\ (4i + 1),$$
$$J_1 : p\ (4(x^2 + x) + 1)$$
$$\vdash$$
$$G : p\ (y^2)$$

Approche

Approche par certificats

Production et vérification de *certificats*

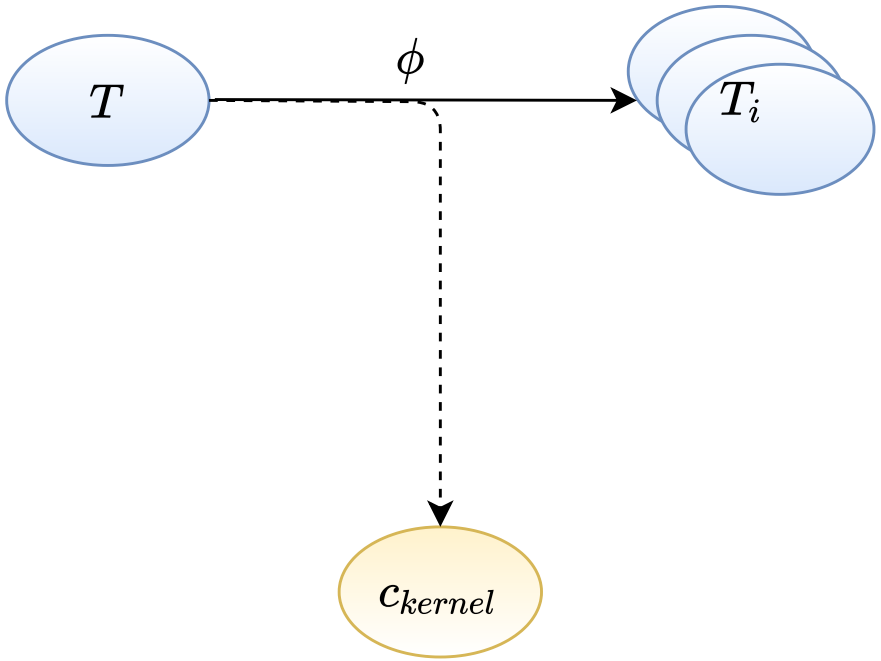
- *Robustesse* vis-à-vis des changements dans le code
- *Incrémentalité*, certification progressive

Autres particularités :

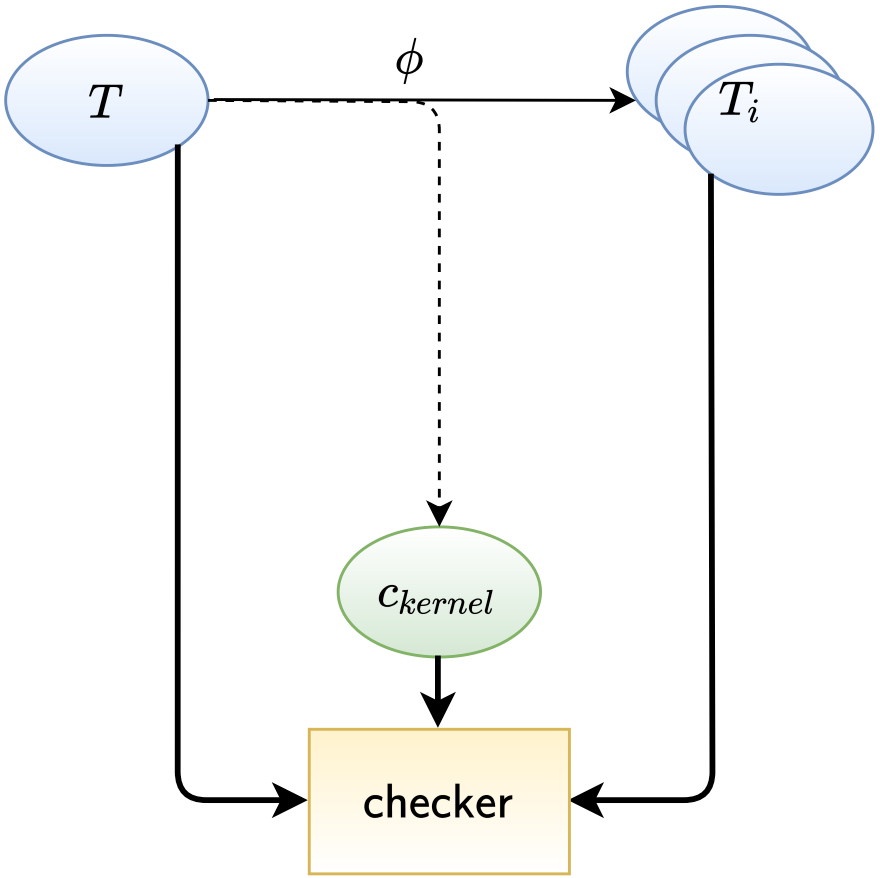
- *Généralité*, logique expressive
- *Modularité*, transformations composables

Appliquée à Why3

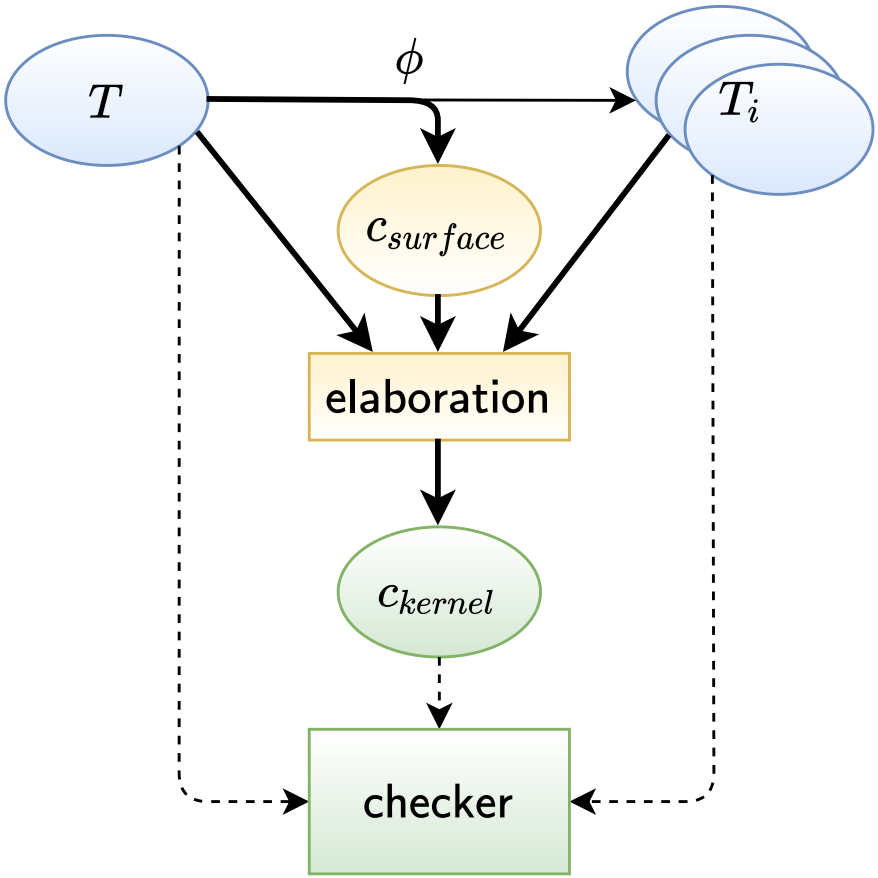
Plan



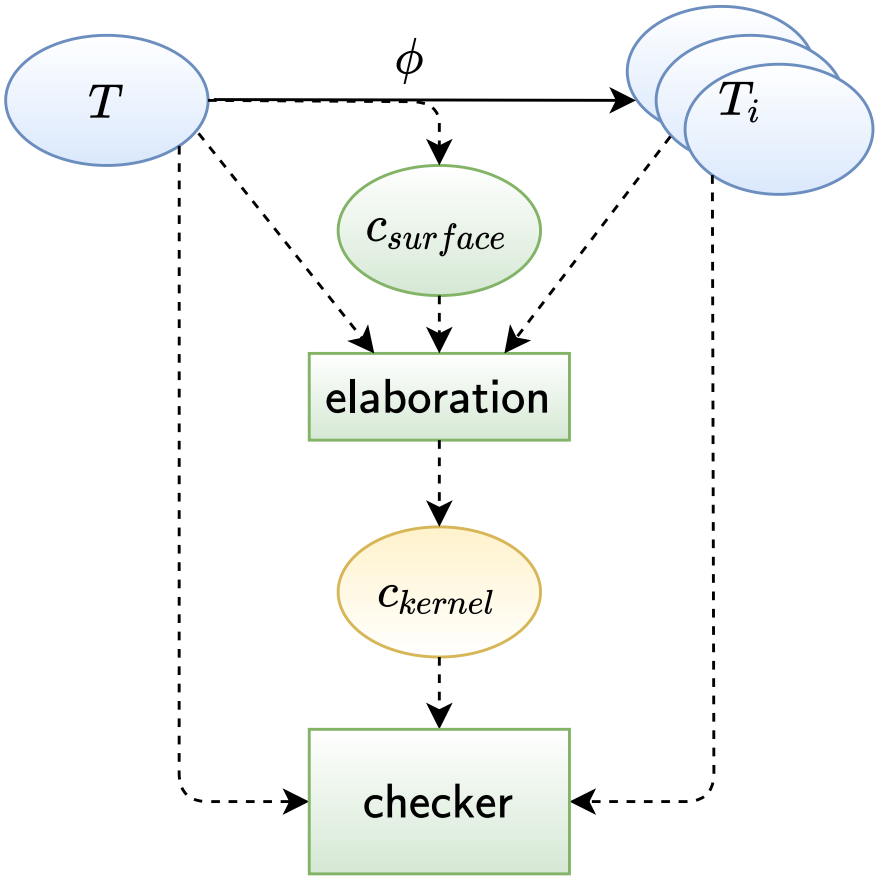
Plan



Plan



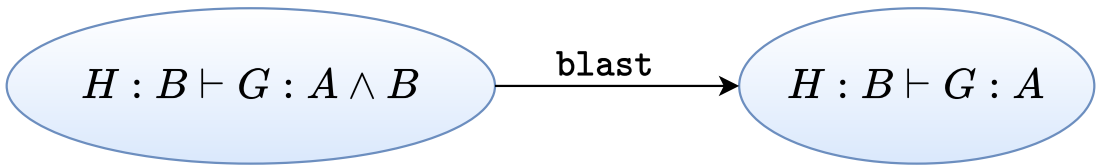
Certificats de noyau



Correction d'une transformation

Correction d'une application de transformation $T \rightsquigarrow T_1, \ldots, T_n$

La validité de $T_1, \ldots, T_n$ implique la validité de T.

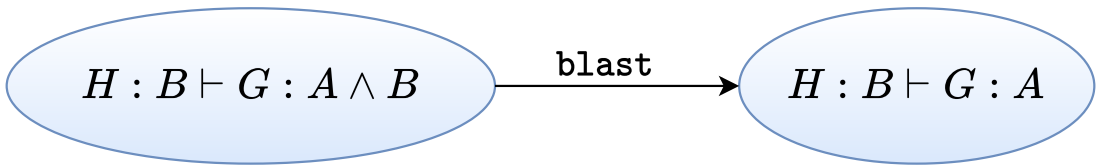


$$\frac{H : B \vdash G : A \qquad \overline{H : B \vdash G : B}}{H : B \vdash G : A \wedge B}$$

Correction d'une transformation

Correction d'une application de transformation $T \rightsquigarrow T_1, \ldots, T_n$

La validité de $T_1, \ldots, T_n$ implique la validité de T.

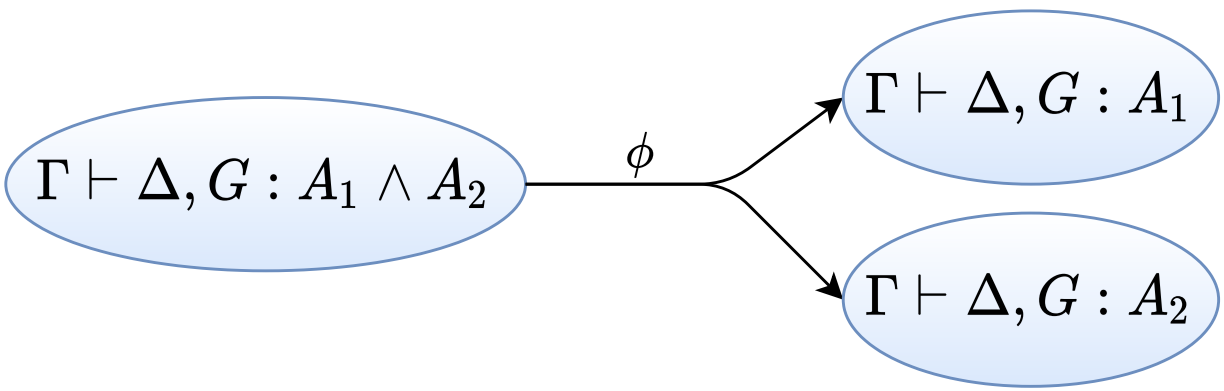


$$\frac{\texttt{KHole}(H : B \vdash G : A) \qquad \dfrac{}{H : B \vdash G : B}\ \texttt{KAxiom}(B, H, G)}{H : B \vdash G : A \wedge B}\ \texttt{KSplit}(G, \_, \_)$$

$$\texttt{KSplit}(G, \texttt{KHole}(H : B \vdash G : A), \texttt{KAxiom}(B, H, G))$$

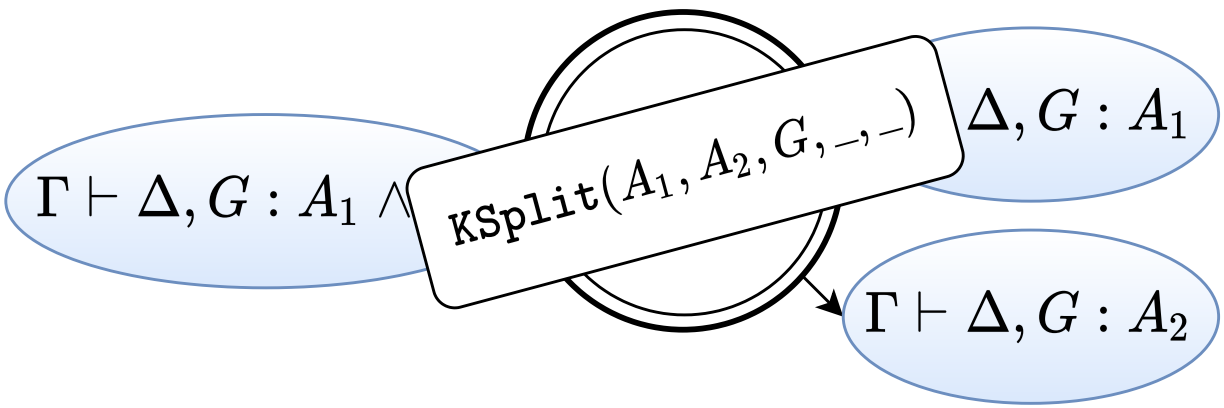
Définition des certificats de noyau

```
type kcert =
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ... (* 19 constructeurs de certificat *)
```



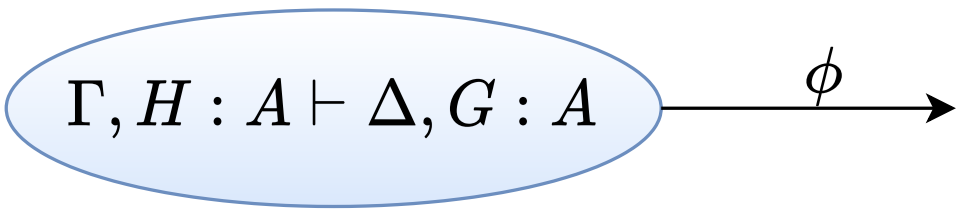
Définition des certificats de noyau

```
type kcert =
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ... (* 19 constructeurs de certificat *)
```



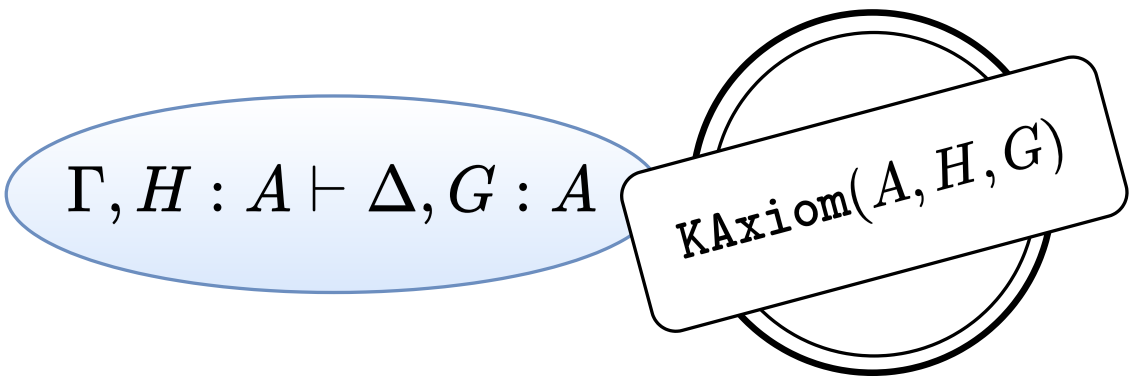
Définition des certificats de noyau

```
type kcert =
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ... (* 19 constructeurs de certificat *)
```



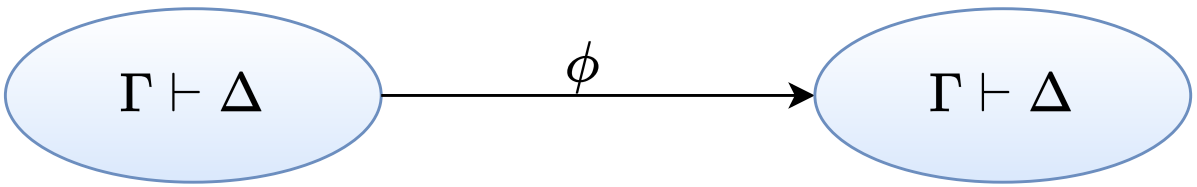
Définition des certificats de noyau

```
type kcert =
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ... (* 19 constructeurs de certificat *)
```



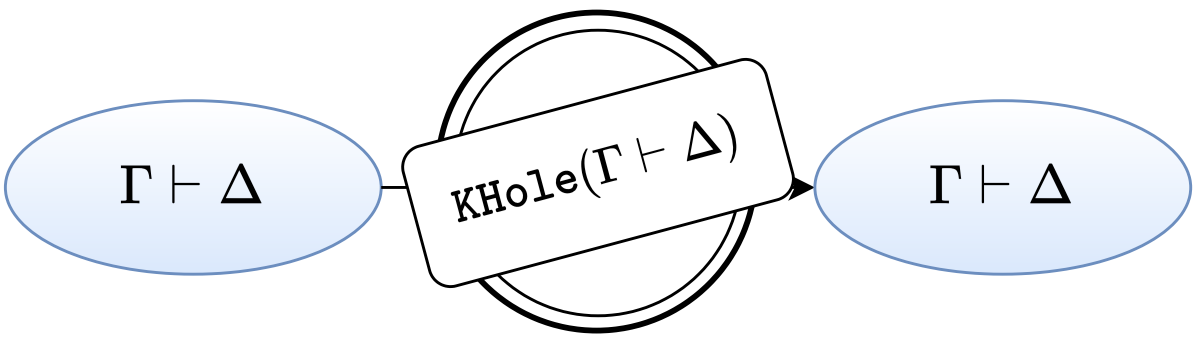
Définition des certificats de noyau

```
type kcert =
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ... (* 19 constructeurs de certificat *)
```



Définition des certificats de noyau

```
type kcert =
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ... (* 19 constructeurs de certificat *)
```



Exemple : transformation `blast`

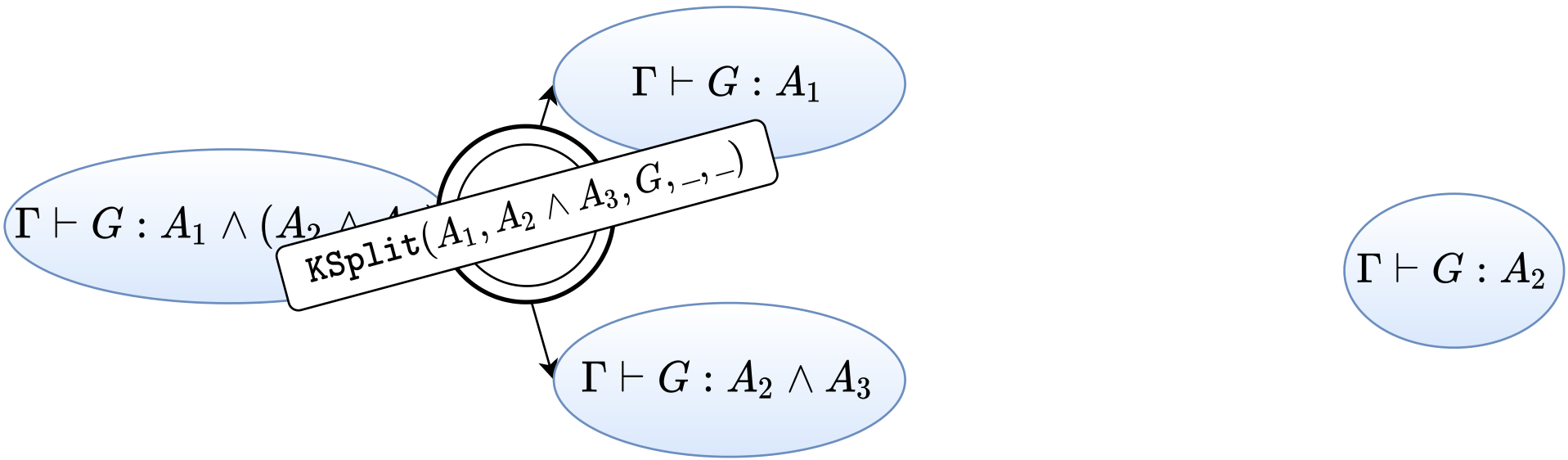
Posons : $\Gamma := H_1 : A_1, H_3 : A_3$

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

Exemple : transformation `blast`

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

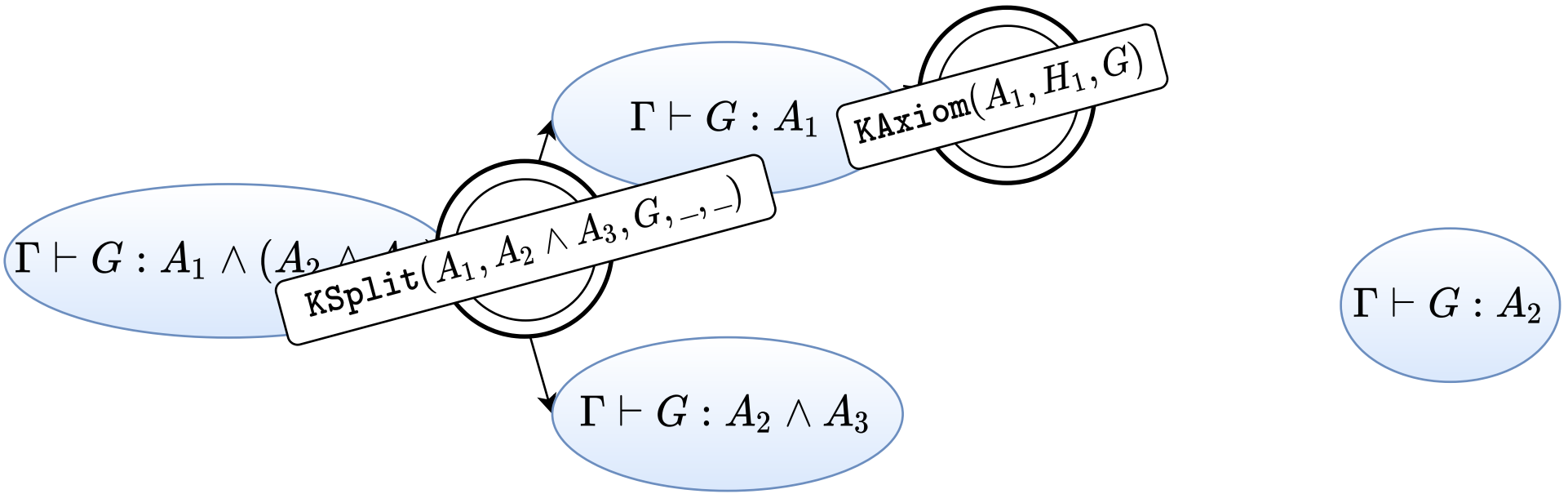
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

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

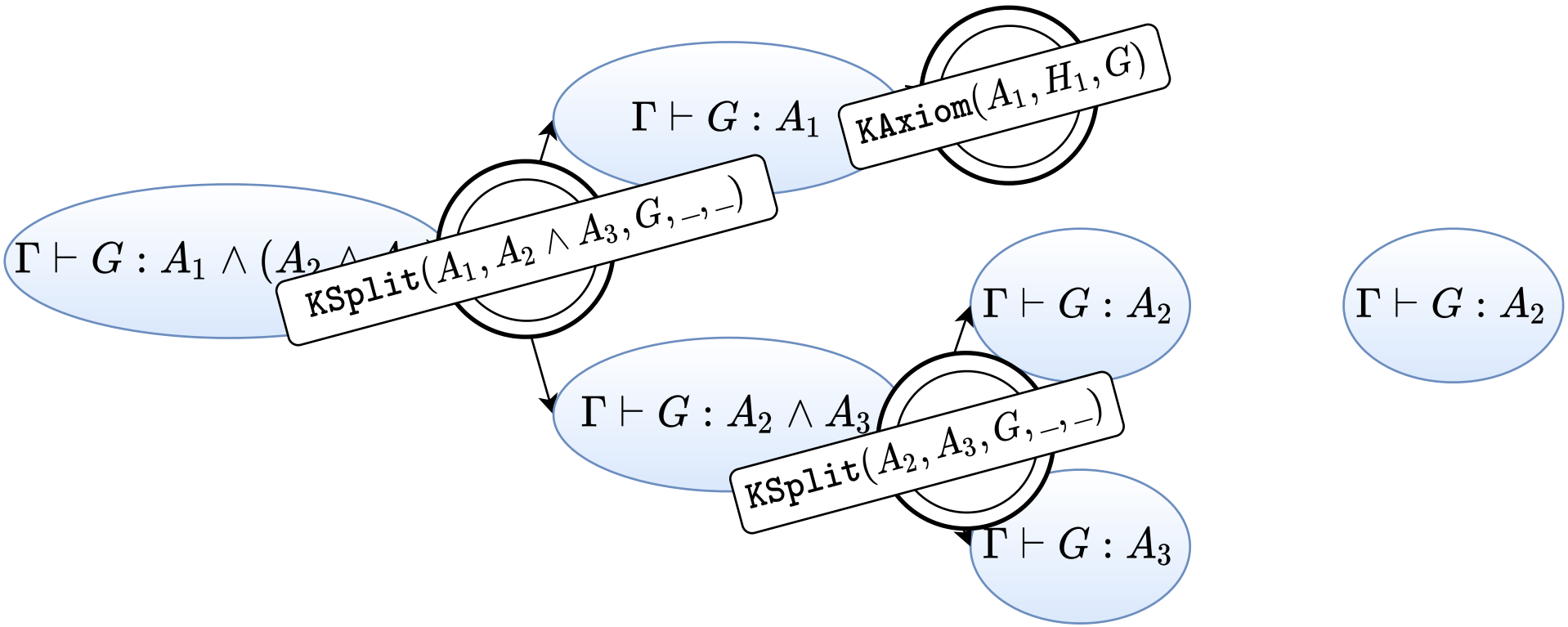
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

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

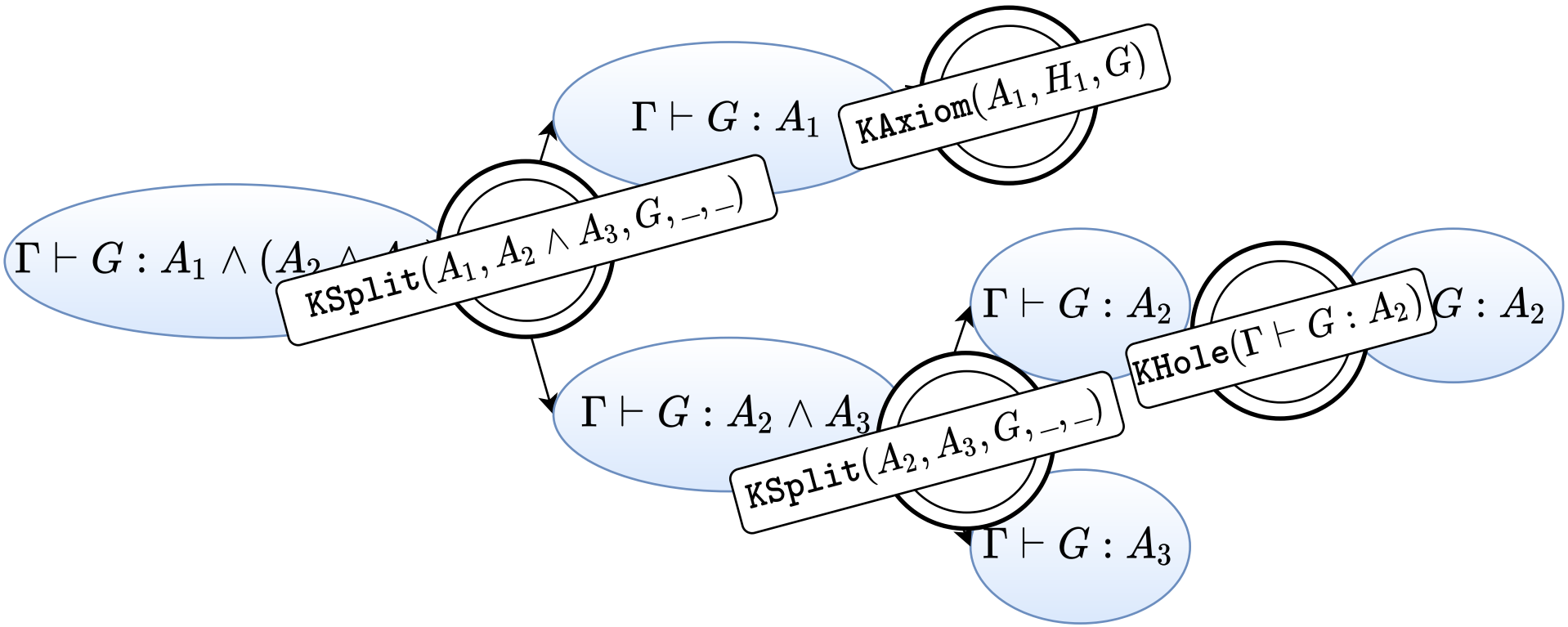
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

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

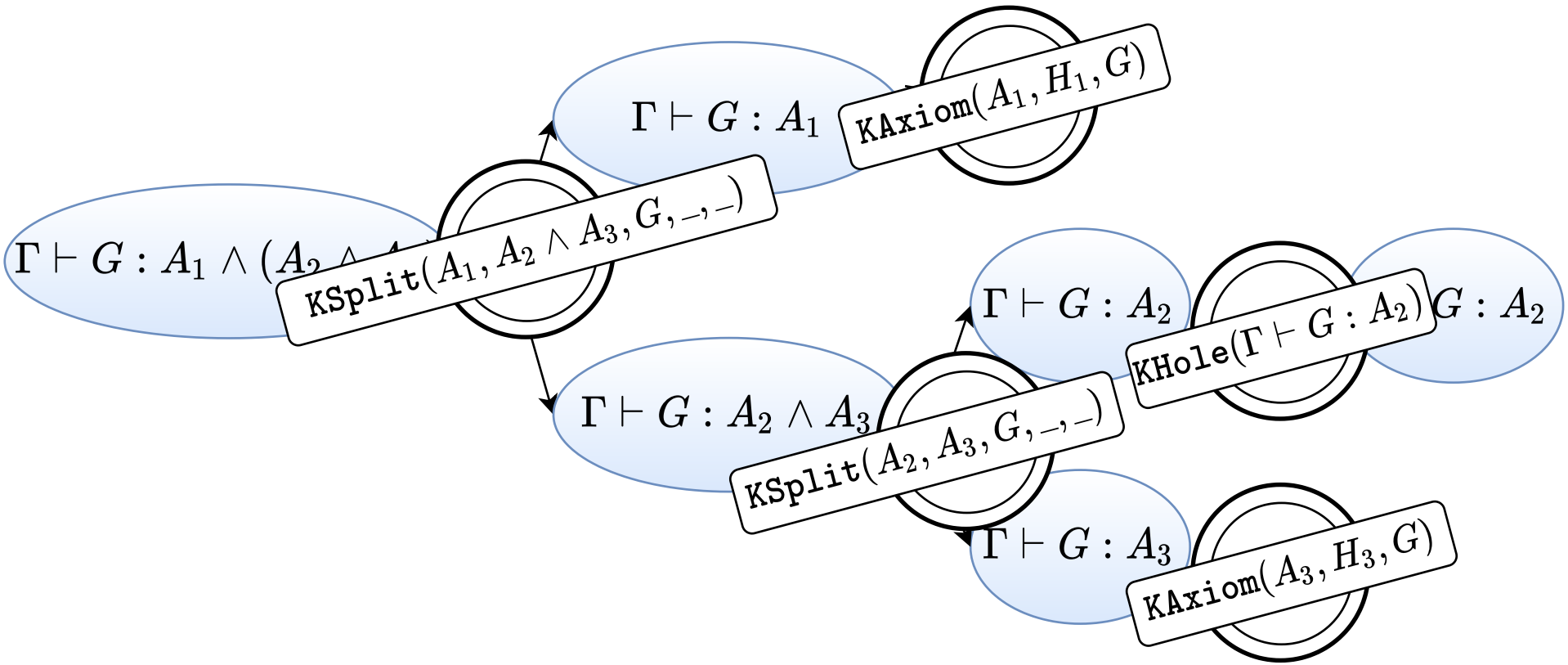
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

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

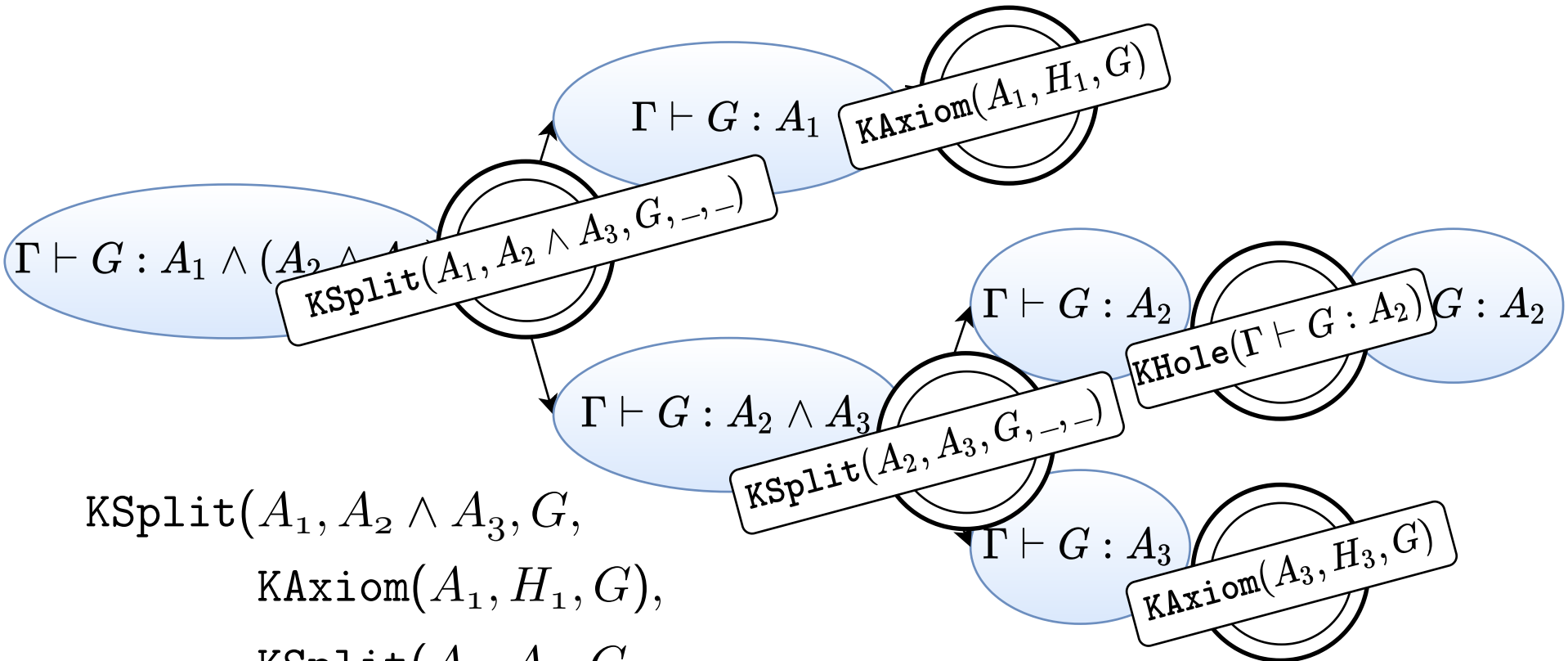
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Exemple : transformation `blast`

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

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



$\texttt{KSplit}(A_1, A_2 \wedge A_3, G,$
$\quad \texttt{KAxiom}(A_1, H_1, G),$
$\quad \texttt{KSplit}(A_2, A_3, G,$
$\quad\quad \texttt{KHole}(\Gamma \vdash G : A_2),$
$\quad\quad \texttt{KAxiom}(A_3, H_3, G)))$

Sémantique des certificats

Correction du prédicat $T \downarrow c$

Si $T \downarrow c$, alors la validité des tâches de c implique la validité de T.

$$\frac{}{\Gamma,\ H : A \vdash \Delta,\ G : A \downarrow \texttt{KAxiom}(A, H, G)}$$

$$\frac{}{\Gamma \vdash \Delta \downarrow \texttt{KHole}(\Gamma \vdash \Delta)}$$

$$\frac{\Gamma \vdash \Delta,\ G : A_1 \downarrow c_1 \qquad \Gamma \vdash \Delta,\ G : A_2 \downarrow c_2}{\Gamma \vdash \Delta,\ G : A_1 \wedge A_2 \downarrow \texttt{KSplit}(A_1, A_2, G, c_1, c_2)}$$

Théories interprétées

Motivation : transformations s'appuyant sur des théories interprétées

Formalisme *extensible* : nouvelles théories (*arithmétique, égalité*)

1. définition de symboles, symboles communs aux tâches
2. définition de certificats, efficacité

`KConv` : conversion selon un ensemble $\mathcal{R}$ de règles de réduction

$$\left.\begin{array}{c}(\lambda x.\ t)\ u \hookrightarrow t[x \mapsto u] \\ \vdots\end{array}\right\} \mathcal{R}$$

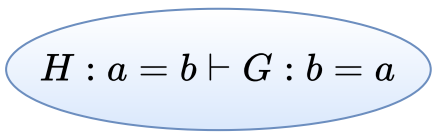
$$\frac{\Gamma, H : A_2 \vdash \Delta \downarrow c \qquad A_1 \equiv_{\mathcal{R}} A_2}{\Gamma, H : A_1 \vdash \Delta \downarrow \mathtt{KConv}(H, A_1, A_2, c)}$$

Théorie de l'égalité : certificats

Nouveau symbole (noté "=")

`KEqRefl` : réflexivité de l'égalité

`KRewrite` : réécriture avec une hypothèse d'égalité

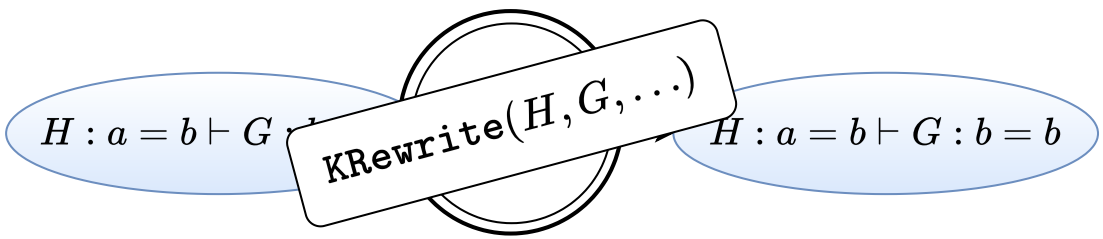


Théorie de l'égalité : certificats

Nouveau symbole (noté "=")

`KEqRefl` : réflexivité de l'égalité

`KRewrite` : réécriture avec une hypothèse d'égalité

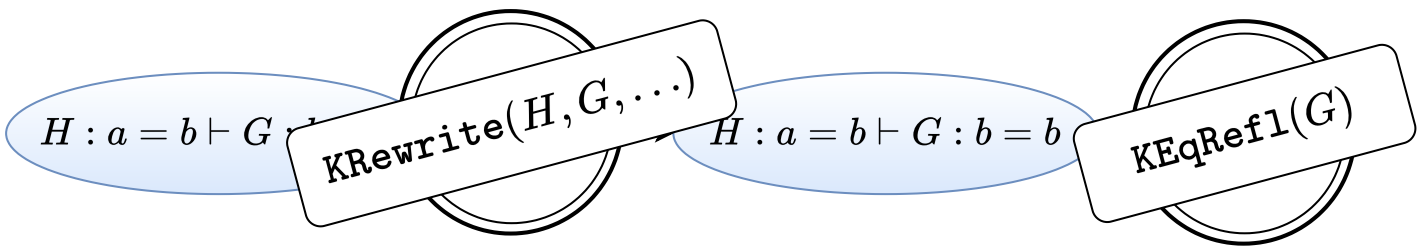


Théorie de l'égalité : certificats

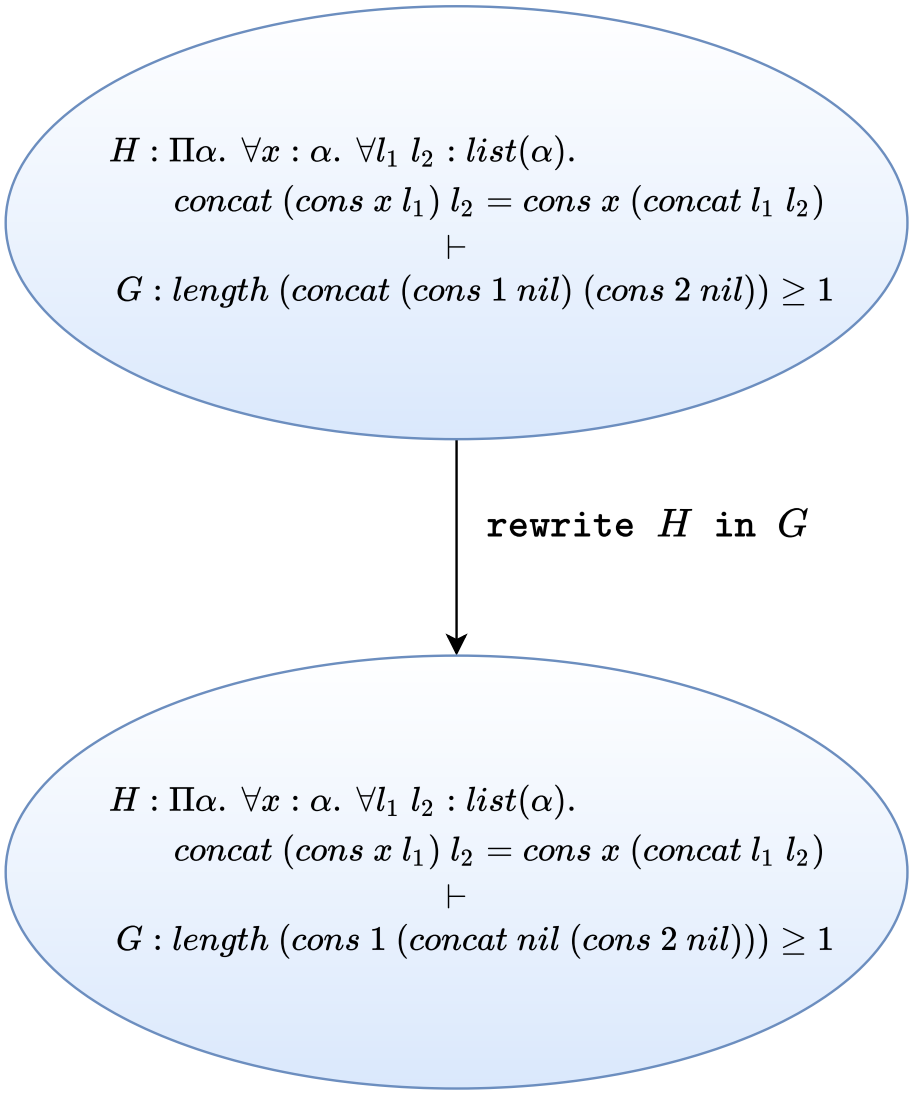
Nouveau symbole (noté "=")

`KEqRefl` : réflexivité de l'égalité

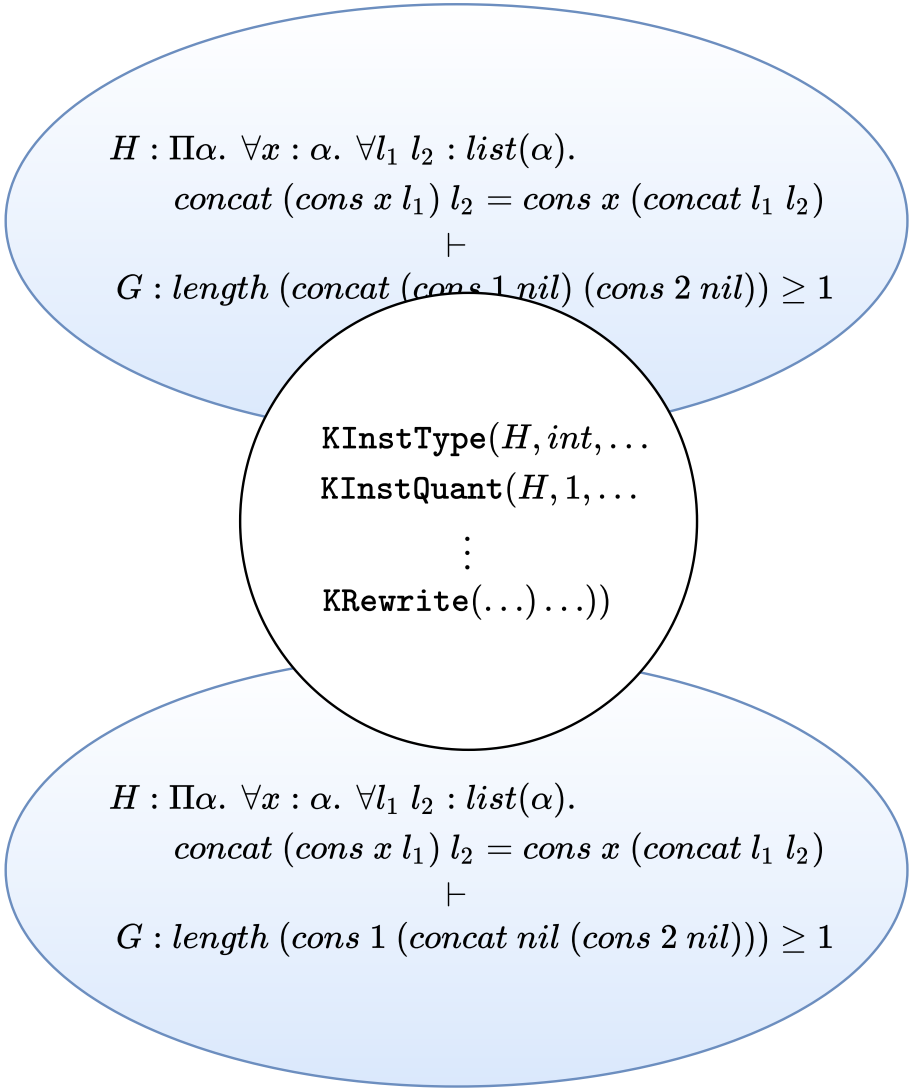
`KRewrite` : réécriture avec une hypothèse d'égalité



Théorie de l'égalité : exemple



Théorie de l'égalité : exemple

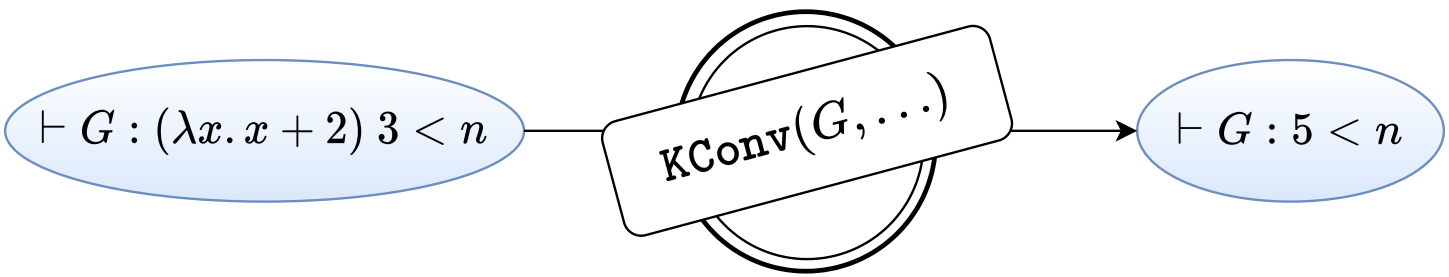


Arithmétique des entiers relatifs

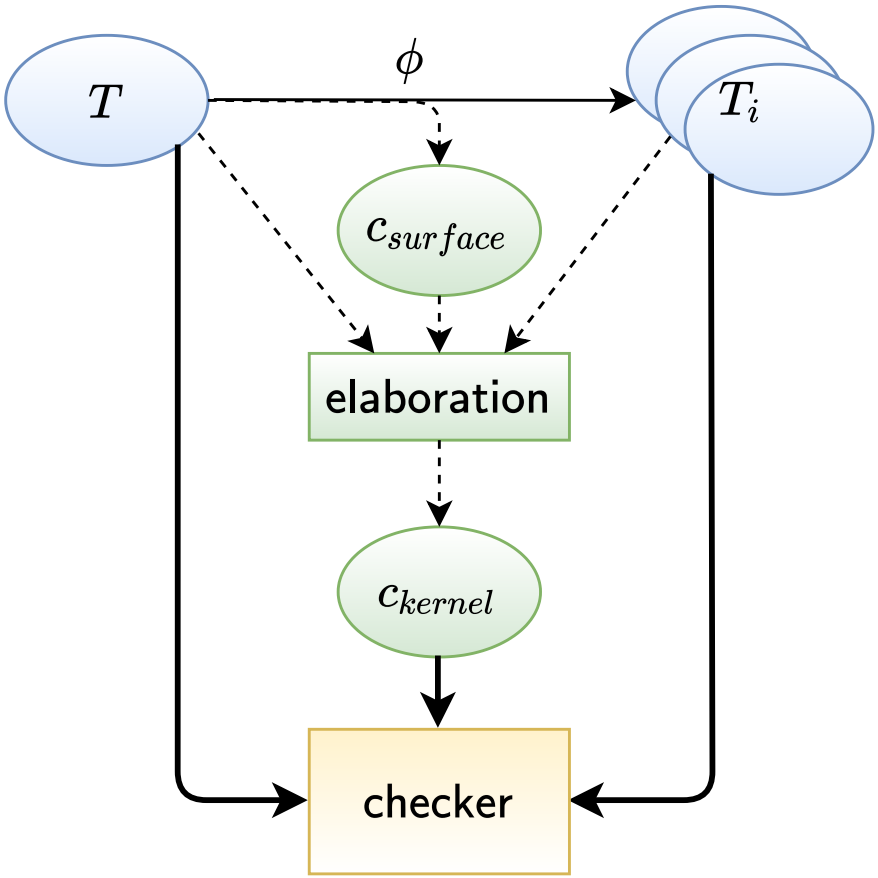
Nouveaux symboles : littéraux entiers, $+$, $\times$, $<$, $\ldots$

`KInduction` : induction forte sur les entiers relatifs

`KConv` : ensemble de règles de réduction étendu

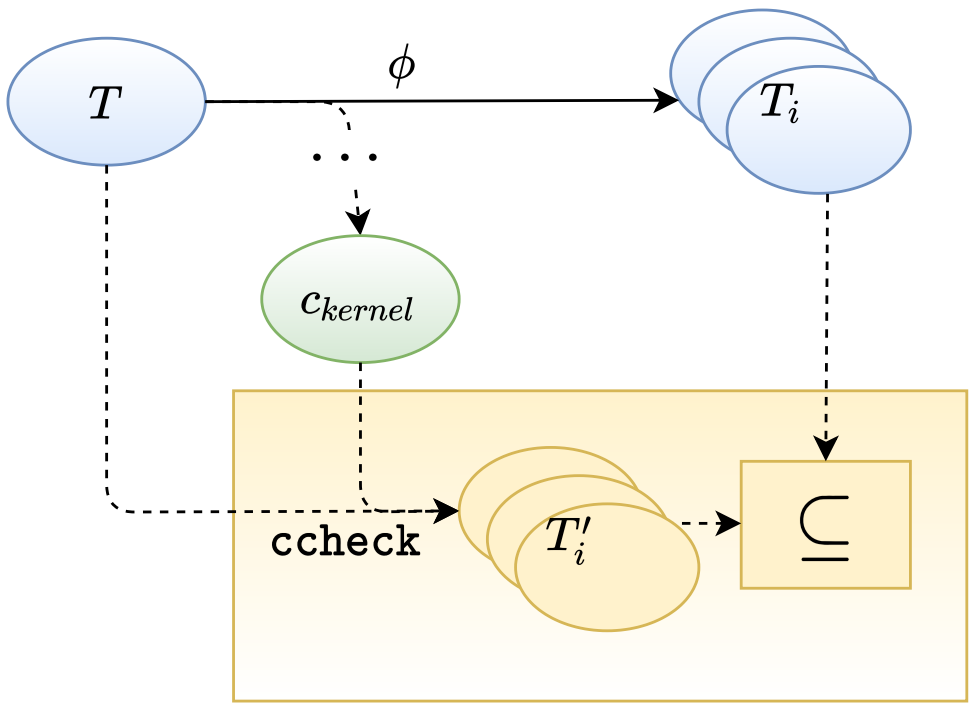


Intégration d'un vérificateur



Vérificateur avec approche calculatoire

```
ccheck : kcert -> task -> task set
```



Vérificateur avec approche calculatoire : implémentation

Cas $\texttt{KSplit}(A_1, A_2, G, c_1, c_2)$:

- s'applique à $T \triangleq \Gamma \vdash \Delta, G : A_1 \wedge A_2$
- produit $T_1 \triangleq \Gamma \vdash \Delta, G : A_1$ et $T_2 \triangleq \Gamma \vdash \Delta, G : A_2$
- renvoie $\texttt{ccheck}\ c_1\ T_1 \cup \texttt{ccheck}\ c_2\ T_2$

Correction de la fonction *ccheck*

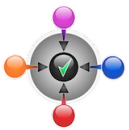
Si $\texttt{ccheck}\ c_{kernel}\ T = \{T'_1, \ldots, T'_n\}$, alors la validité de $T'_1, \ldots, T'_n$ implique la validité de T.

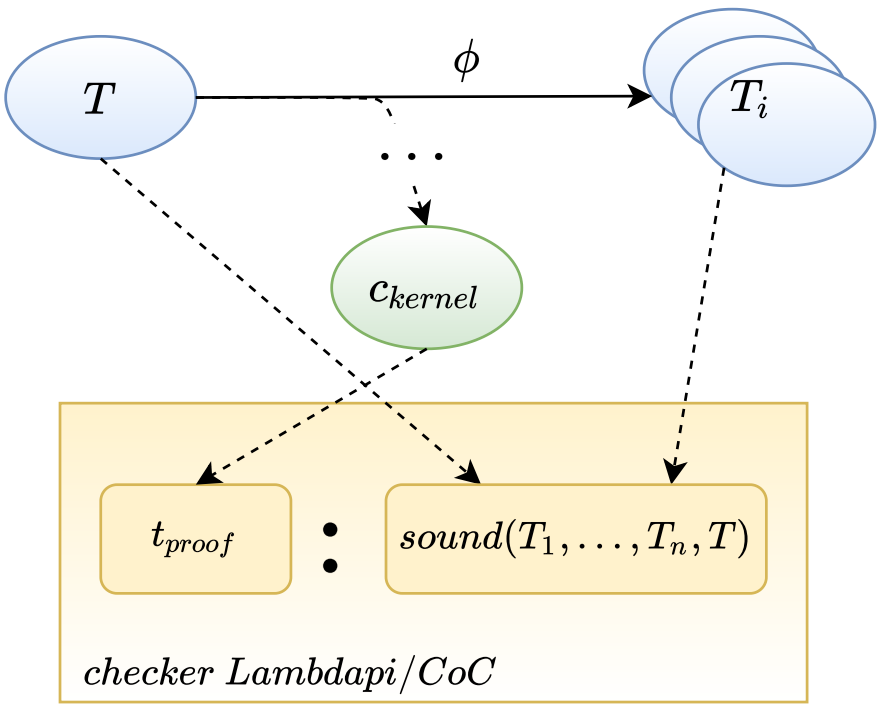
Base de confiance, vérification de l'implémentation ?

Vérificateur Lambdapi/CoC

Objectif : renforcer la confiance accordée à notre méthode

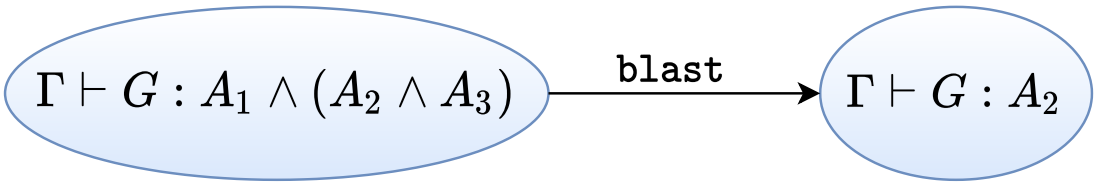
- encoder les tâches dans une logique connue, CoC++
- encoder les certificats, utiliser un vérificateur tiers

Lambdapi  : framework logique, $\lambda\Pi$-calcul modulo réécriture



Vérificateur Lambdapi/CoC : exemple

Exemple `blast` avec $\Gamma := H_1 : A_1,\ H_3 : A_3$



Encodage des tâches dans Lambdapi/CoC :

```
symbol ToVerify : ℰ (
  (∀ A1 A2 A3, A1 ⇒ A3 ⇒ ¬A2 ⇒ ⊥) ⇒
  (∀ A1 A2 A3, A1 ⇒ A3 ⇒ ¬(A1 ∧ (A2 ∧ A3)) ⇒ ⊥))
:= λ T2 A1 A2 A3 H1 H3 G, ... // terme généré à partir du certificat
```

Vérificateur Lambdapi/CoC : étapes de certificat

Définition et preuve en Lambdapi/CoC

Définition de l'égalité + preuve de ses propriétés

```
symbol RewriteHyp a b P : 𝓔 (
  (a = b ⇒ P b ⇒ ⊥) ⇒
  (a = b ⇒ P a ⇒ ⊥))
:= λ T eq pa, T eq (LeibnizEquality a b eq P pa);
```

Encodage binaire + preuve de résultats arithmétiques

```
symbol StrongInduction a P : 𝓔 (
  (∀ x, x ≤ a ⇒ P x) ⇒
  (∀ x, a < x ⇒ (∀ n, n < x ⇒ P n) ⇒ P x) ⇒
  ∀ x, P x)
:= ...
```

Vérificateur Lambdapi/CoC : résultats

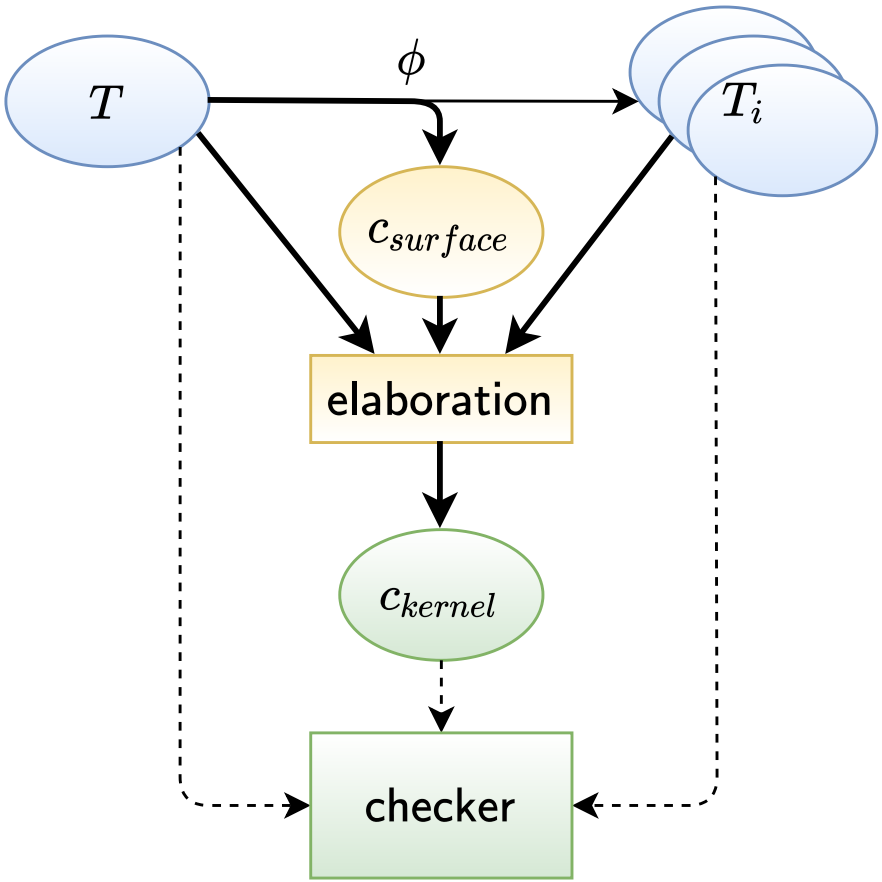
Base de confiance du vérificateur Lambdapi/CoC :

- contient la traduction des tâches (encodage dans CoC)
- contient Lambdapi/CoC
- contient le vérificateur de type de Lambdapi
- ne contient pas la production du terme de preuve

Correction du vérificateur Lambdapi/CoC

Si le type $sound(T_1, \ldots, T_n, T)$ est habité, alors la validité de $T_1, \ldots, T_n$ implique la validité de T.

Certificats de surface



Élaboration de certificats

Certificat de noyau

$$\texttt{KSplit}(A_1, A_2, G,\ \texttt{KHole}(T_1),\ \texttt{KHole}(T_2))$$

Élaboration de certificats

Certificat de surface

$\lambda c_1\, c_2.\ \texttt{SSplit}(G, c_1, c_2)$

Certificat de noyau

$\texttt{KSplit}(A_1, A_2, G,$
$\texttt{KHole}(T_1),$
$\texttt{KHole}(T_2))$

- moins de paramètres (A_1, A_2)
  ↳ plus facile à produire
- abstraction des tâches résultantes
  ↳ réutilisable
- facilite la composition de certificats
  ↳ plus modulaire

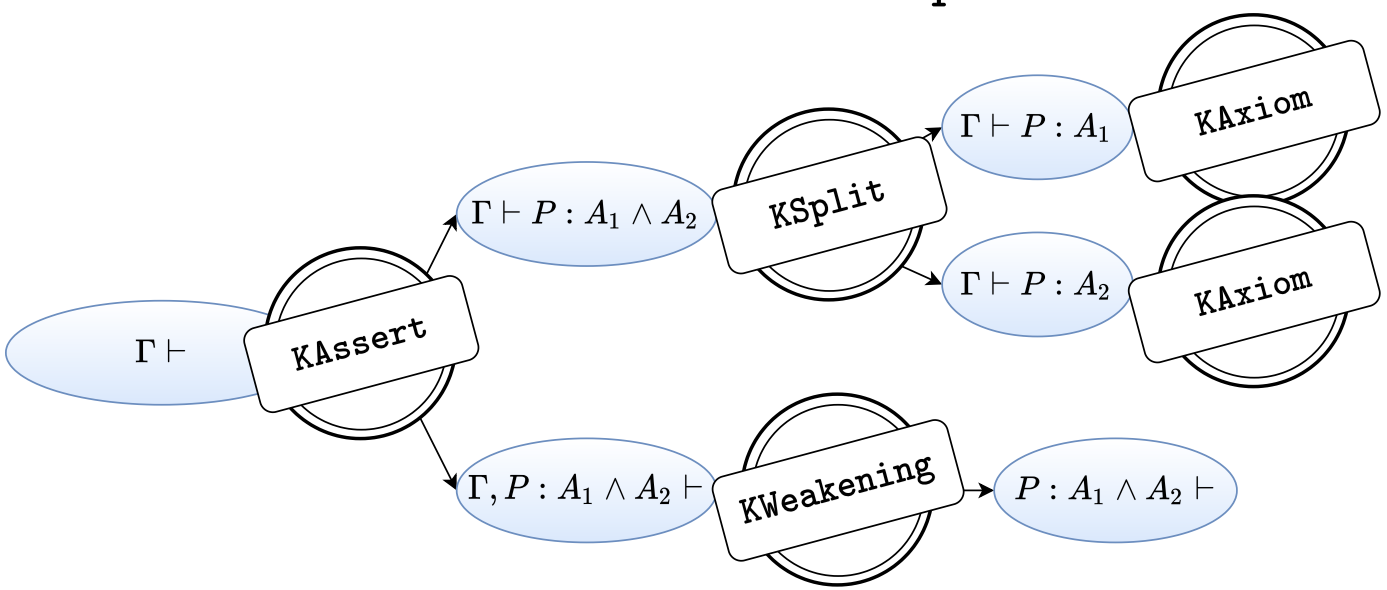
Interface

```
type scert
val axiom : ident -> ident -> scert
val split : ident -> scert
[...]
```

Interface

```
type scert
val axiom : ident -> ident -> scert
val split : ident -> scert
[...]
val construct : ident -> ident -> ident -> scert
```

construct h1 h2 p

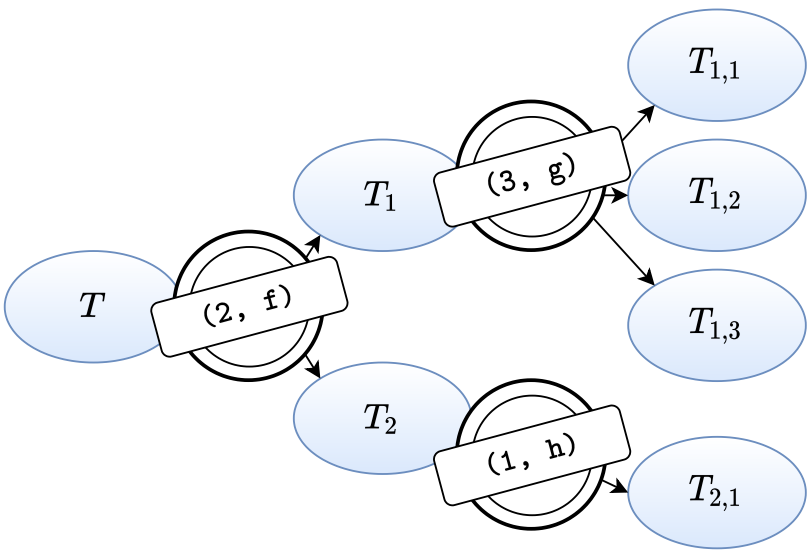


$$\Gamma := H_1 : A_1, H_2 : A_2$$

Composition de certificats

```
type sc = SSplit of ident * sc | ...
type scert = int * (sc list -> sc)
val ( +++ ) : scert -> scert list -> scert
```

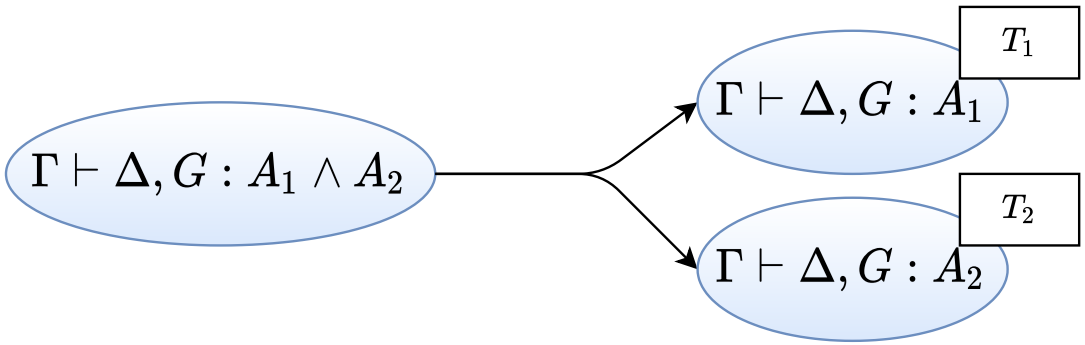
```
(2, f) +++ [(3, g); (1, h)]
```



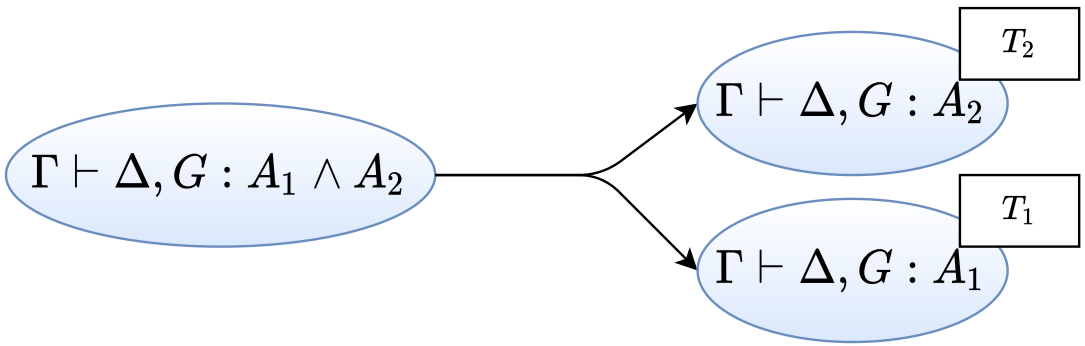

```
4,
fun [c11; c12; c13; c21] ->
    f [g [c11;
          c12;
          c13];
       h [c21]]
```

Réordonner les tâches résultantes

$$c = \texttt{KSplit}(A_1, A_2, G, \texttt{KHole}(T_1), \texttt{KHole}(T_2))$$



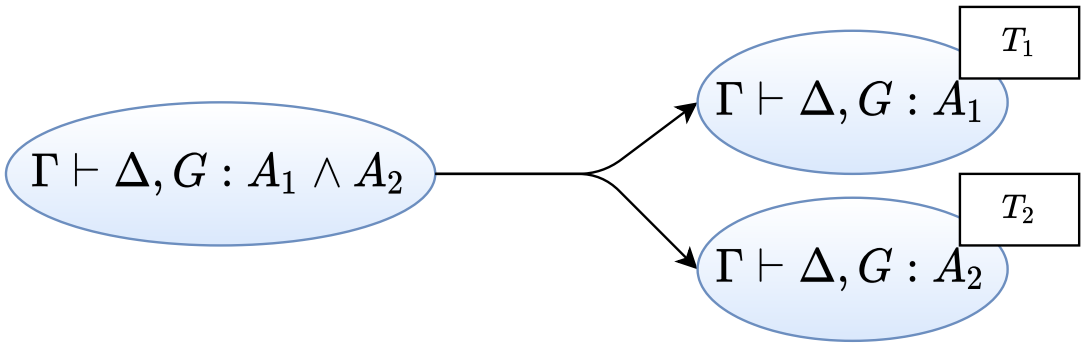
$$c = \texttt{KSplit}(A_1, A_2, G, \texttt{KHole}(T_1), \texttt{KHole}(T_2))$$



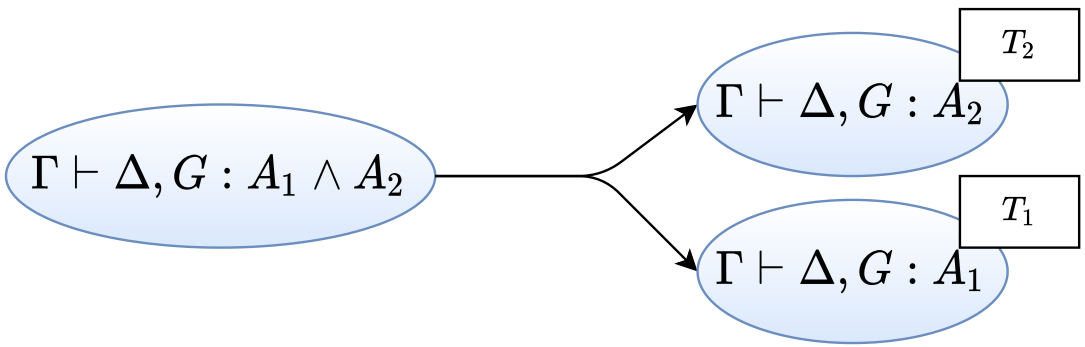
Composition de c avec une liste $[c_1; c_2]$?

Réordonner les tâches résultantes

```
split g ≡ lambda2 (fun c1 c2 -> split g +++ [c1; c2])
```




```
lambda2 (fun c2 c1 -> split g +++ [c1; c2])
```




```
val lambda2 : (scert -> scert -> scert) -> scert
```

Contributions

Cadre dans une logique d'ordre supérieur avec théories interprétées

Contributions générales :

- certificats de noyau avec trous
- vérificateur avec approche calculatoire
- traduction des tâches en Lambdapi/CoC
- traduction des certificats, bibliothèque arithmétique Lambdapi
- certificats de surface, élaboration, composition

Applications :

- traduction des tâches de Why3 dans notre formalisme
- *bug* de correction dans les transformations `case` et `destruct`
- $\sim$ 15 transformations certifiantes simples
- + `blast`, `rewrite`, `induction`, `split`, `compute`

Évaluation expérimentale

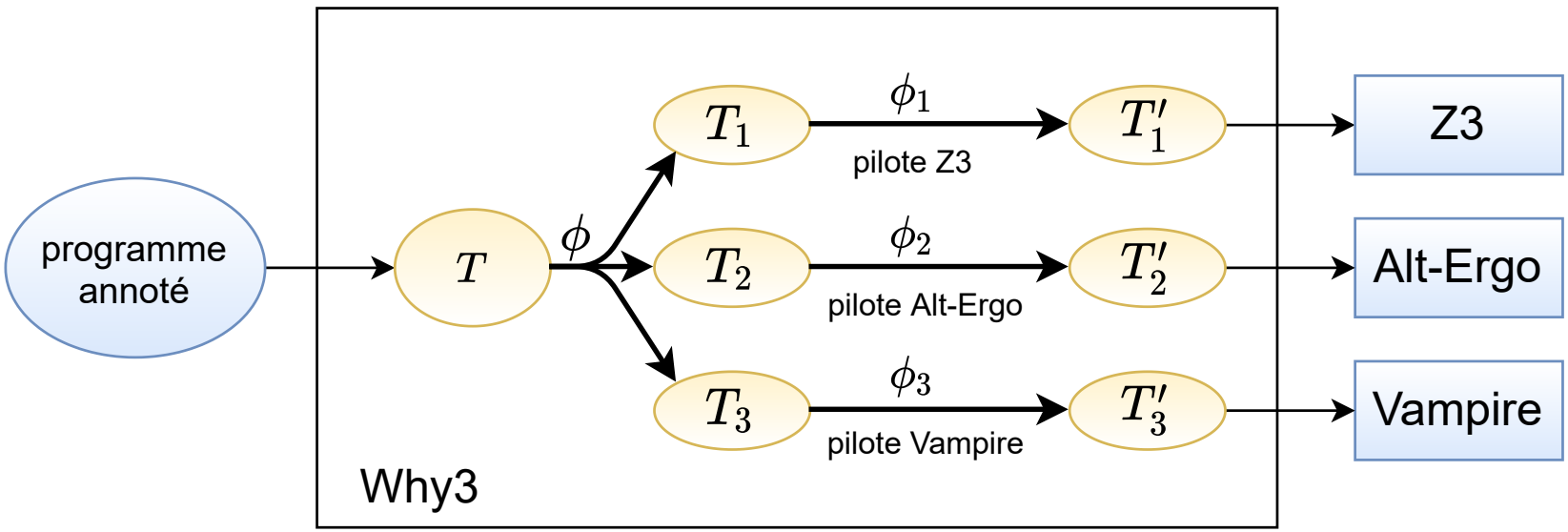
Évaluation de `blast` sur :

$$\vdash G : A_1 \Rightarrow (A_1 \Rightarrow A_2) \Rightarrow \ldots (A_{n-1} \Rightarrow A_n) \Rightarrow A_n$$

nombre de variables	5	10	15	20	25	50	100	200	400	800
taille des certificats de surface (kB)	4	4	8	12	16	32	64	128	260	516
taille des certificats de noyau (kB)	8	16	28	36	52	124	344	1200	4000	15000
temps de la transformation (s)	~ 0	~ 0	0.01	0.01	0.02	0.06	0.29	1.22	5.4	25
temps de la transformation certifiante (s)	~ 0	~ 0	0.01	0.01	0.02	0.08	0.32	1.34	5.9	28
temps du vérificateur OCaml (s)	~ 0	~ 0	~ 0	~ 0	~ 0	~ 0	0.02	0.07	0.28	1.1
temps du vérificateur Lambdapi (s)	0.07	0.21	0.64	1.6	3.2	35	450	-	-	-

Perspectives

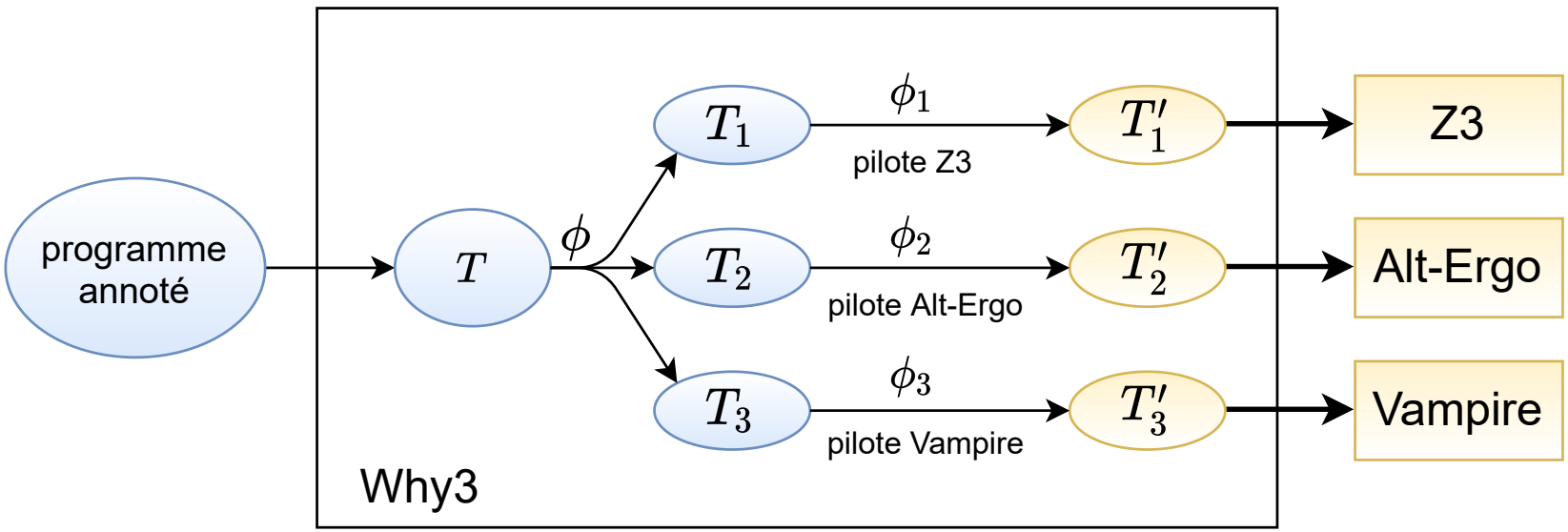
Vérification des transformations logiques



Élimination du polymorphisme, élimination des types algébriques

Perspectives

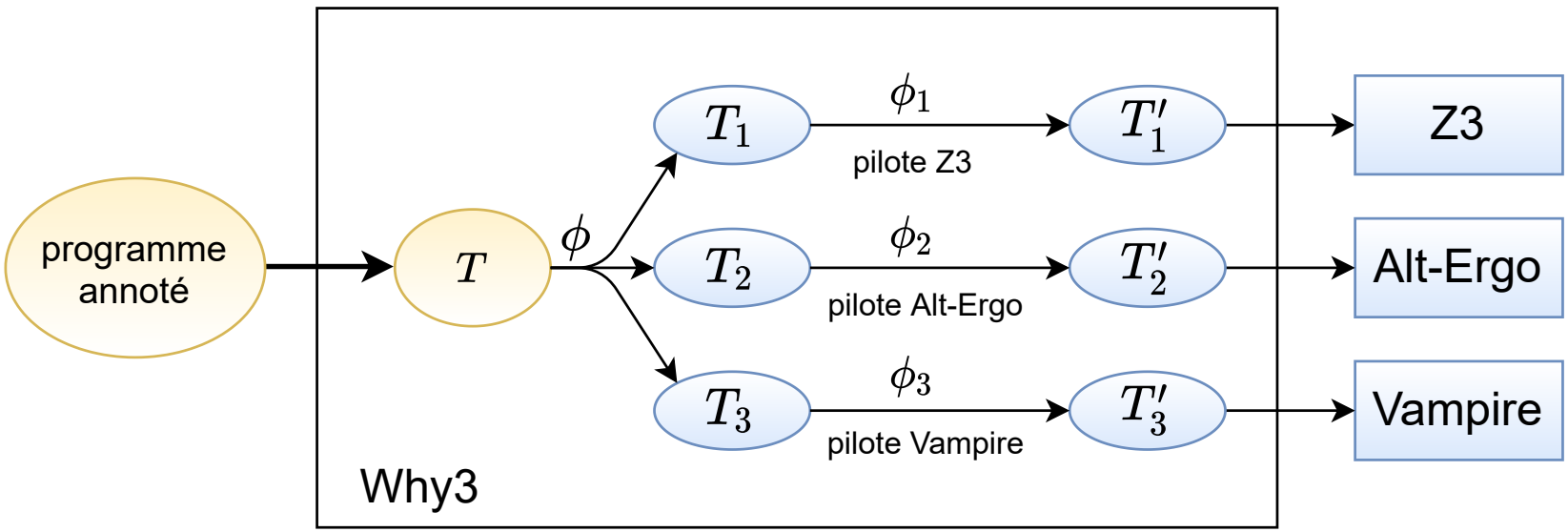
Vérification des appels des prouveurs automatiques



ekstrakto, SMTCoq

Perspectives

Vérification de la génération de la tâche initiale



Interprétation dans CoC

Utilisation de *transformations* dans d'autres contextes