A Framework for Proof-carrying Logical Transformations

Seventh Workshop on Proof eXchange for Theorem Proving

July 11, 2021

Quentin Garchery

Context

Deductive systems with large trusted base:

PVS, F*, Why3, Dafny, Viper, ...

- call to external provers

Z3, Yices, Vampire, Alt-Ergo, ...

- internal *logical transformations* (a.k.a. tactics)
  proof task $\rightsquigarrow$ simpler proof tasks
  - ↳ Simplify, decompose proof tasks
  - ↳ Convert to the logic of an external tool

Our objective

Improve trust in logical transformations

Example: `rewrite` Transformation

$$\vdots$$
$$H : \Pi\alpha.\ \forall x : \alpha.\ \forall l_1\ l_2 : \mathit{list}(\alpha).\ \mathit{concat}\ (\mathit{cons}\ x\ l_1)\ l_2 = \mathit{cons}\ x\ (\mathit{concat}\ l_1\ l_2)$$
$$\overline{\qquad\qquad}$$
$$G : \mathit{length}\ (\mathit{concat}\ (\mathit{cons}\ 1\ \mathit{nil})\ (\mathit{cons}\ 2\ \mathit{nil})) \geq 1$$

$$\rightsquigarrow \quad \texttt{rewrite H in G}$$

$$\vdots$$
$$H : \Pi\alpha.\ \forall x : \alpha.\ \forall l_1\ l_2 : \mathit{list}(\alpha).\ \mathit{concat}\ (\mathit{cons}\ x\ l_1)\ l_2 = \mathit{cons}\ x\ (\mathit{concat}\ l_1\ l_2)$$
$$\overline{\qquad\qquad}$$
$$G : \mathit{length}\ (\mathit{cons}\ 1\ (\mathit{concat}\ \mathit{nil}\ (\mathit{cons}\ 2\ \mathit{nil}))) \geq 1$$

Overview

Our approach

Production and verification of *certificates*

- *Robustness* w.r.t. change in the code of transformations
- *Incrementality*: gradually add support for various cases

Additional needs:

- *Modularity*: composability of transformations
- Support for *expressive logics*
  - higher-order logic
  - polymorphism
  - theories: equality, integers

Implemented inside Why3 

Correctness of Transformations

Correctness of transformation application $T \rightsquigarrow T_1, \ldots, T_n$

Validity of $T_1, \ldots, T_n$ implies validity of T

Transformation `rewrite` example:

$$H : \Pi\alpha.\ \forall x : \alpha.\ \forall l_1\ l_2 : \mathit{list}(\alpha).\ \mathit{concat}\ (\mathit{cons}\ x\ l_1)\ l_2 = \mathit{cons}\ x\ (\mathit{concat}\ l_1\ l_2)$$

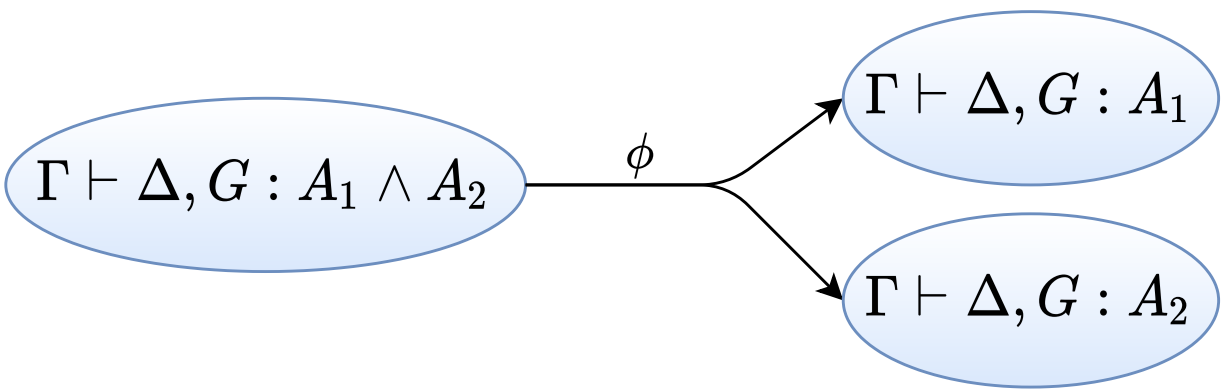
$$G : \mathit{length}\ (\mathit{concat}\ (\mathit{cons}\ 1\ \mathit{nil})\ (\mathit{cons}\ 2\ \mathit{nil})) \geq 1$$

A possible certificate:

$$\begin{aligned}&\texttt{KInstType}(H, H', \mathit{int},\\ &\quad \texttt{KInstQuant}(H', H', 1,\\ &\quad \texttt{KInstQuant}(H', H', \mathit{nil},\\ &\quad \texttt{KInstQuant}(H', H', \mathit{cons}\ 2\ \mathit{nil},\\ &\quad \texttt{KRewrite}(H', G, (\lambda l.\ \mathit{length}\ l \geq 1),\\ &\quad \texttt{KClear}(H', \texttt{KHole}(T')))))))\end{aligned}$$

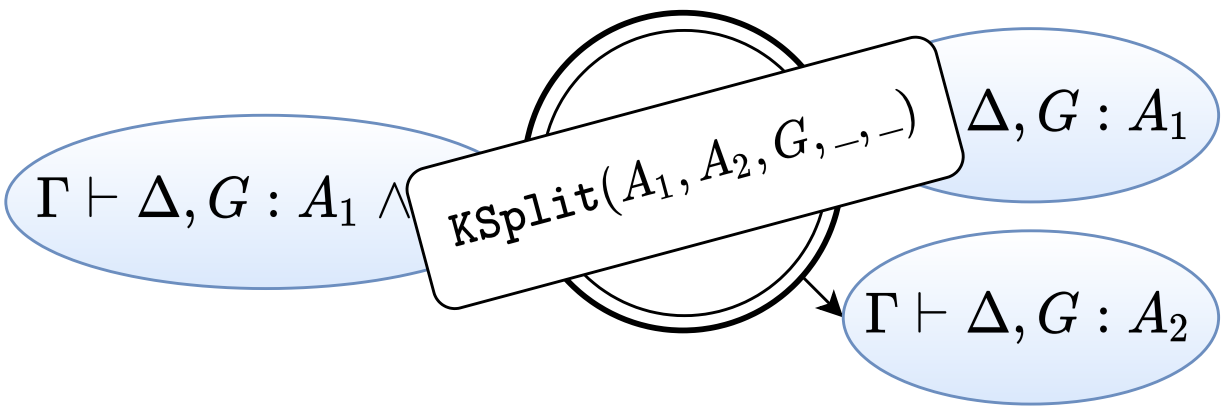
Kernel Certificate Definition

```
type kcert :=
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ...  (* 17 certificate constructors *)
```



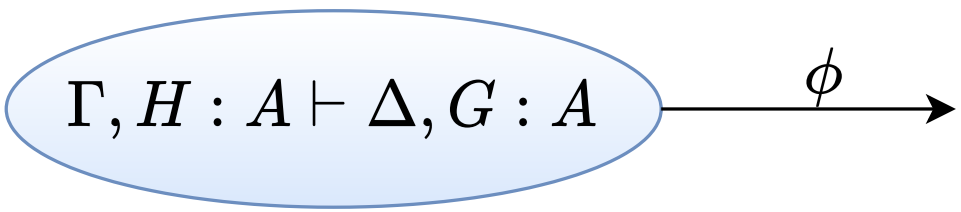
Kernel Certificate Definition

```
type kcert :=
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ...  (* 17 certificate constructors *)
```



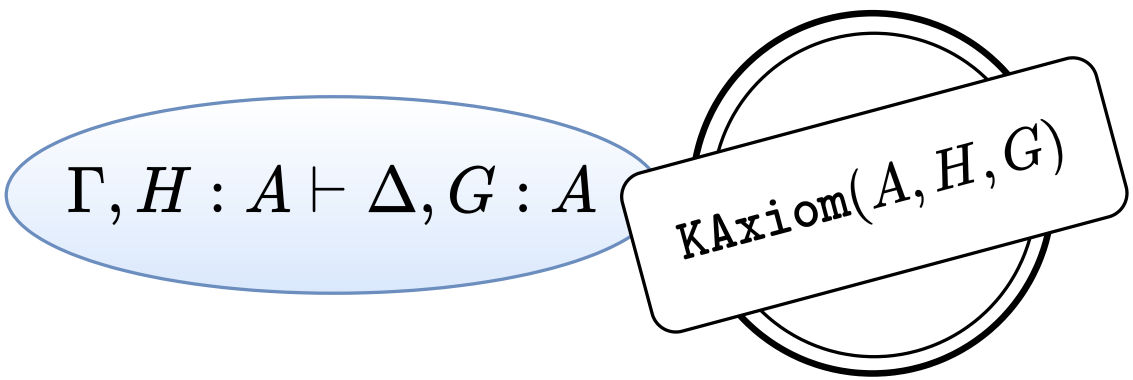
Kernel Certificate Definition

```
type kcert :=
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ...  (* 17 certificate constructors *)
```



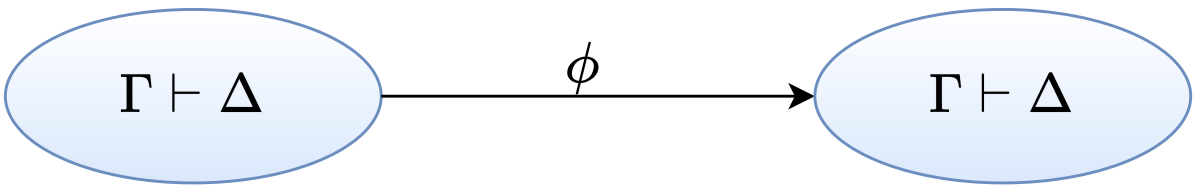
Kernel Certificate Definition

```
type kcert :=
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ...  (* 17 certificate constructors *)
```



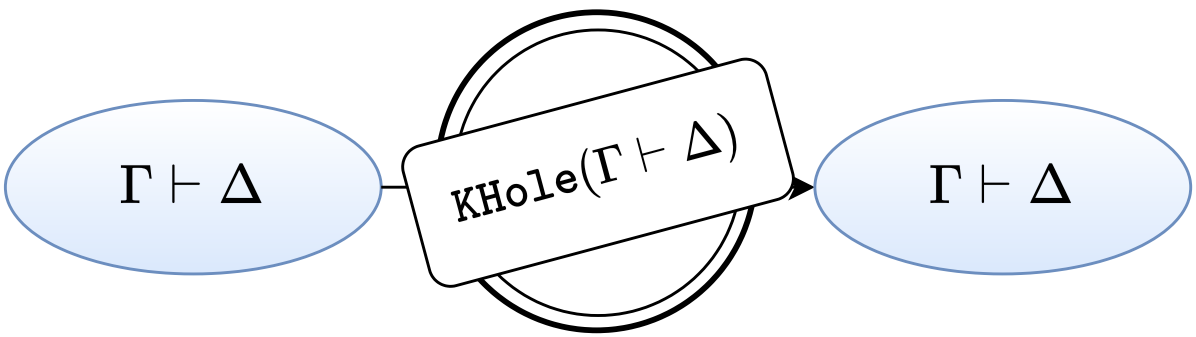
Kernel Certificate Definition

```
type kcert :=
 | KSplit of term * term * ident * kcert * kcert
 | KAxiom of term * ident * ident
 | KHole of task
 | ...  (* 17 certificate constructors *)
```



Kernel Certificate Definition

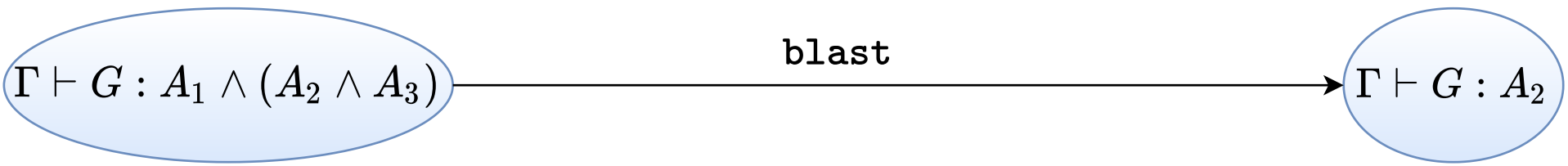
```
type kcert :=
  | KSplit of term * term * ident * kcert * kcert
  | KAxiom of term * ident * ident
  | KHole of task
  | ...  (* 17 certificate constructors *)
```



Example: `blast` Transformation

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

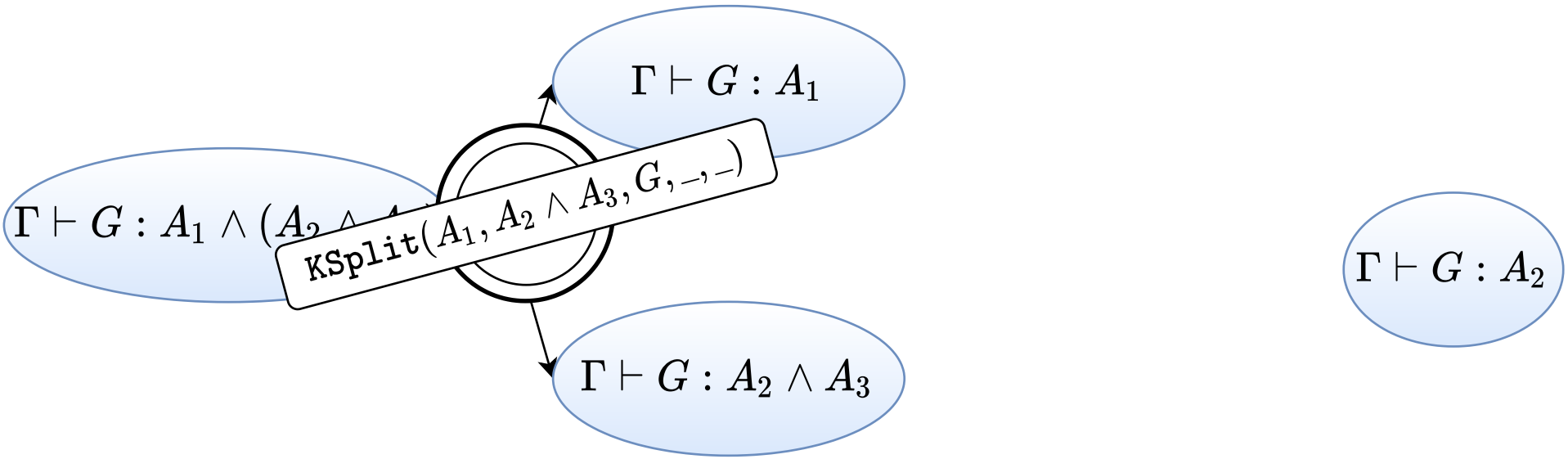
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Example: `blast` Transformation

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

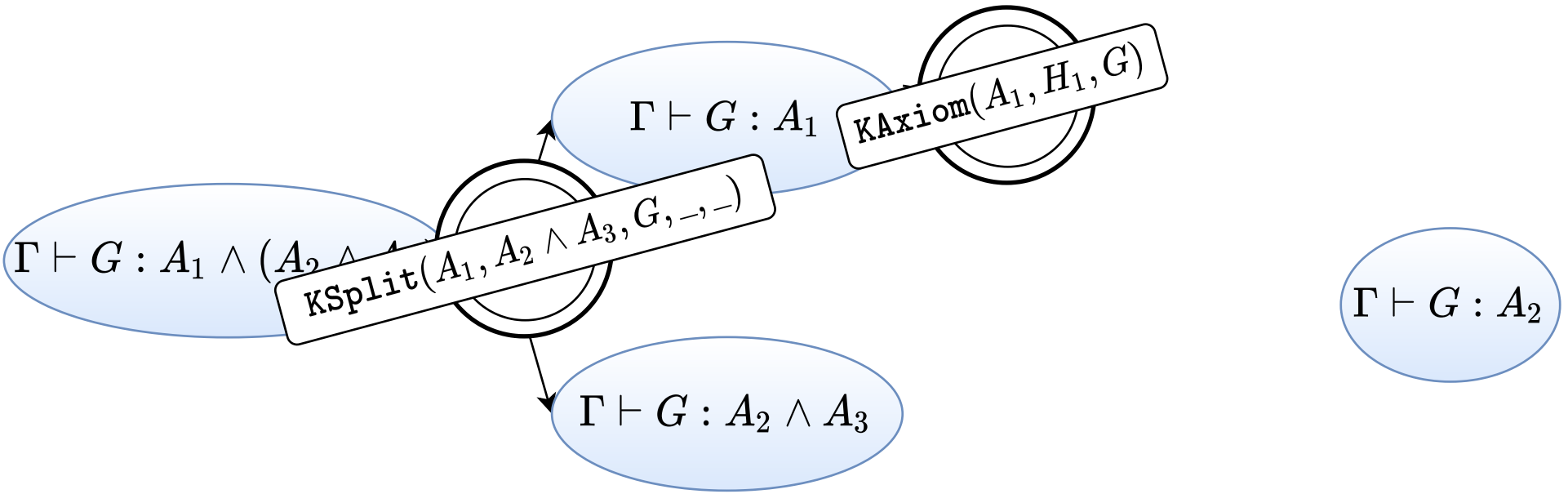
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Example: `blast` Transformation

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

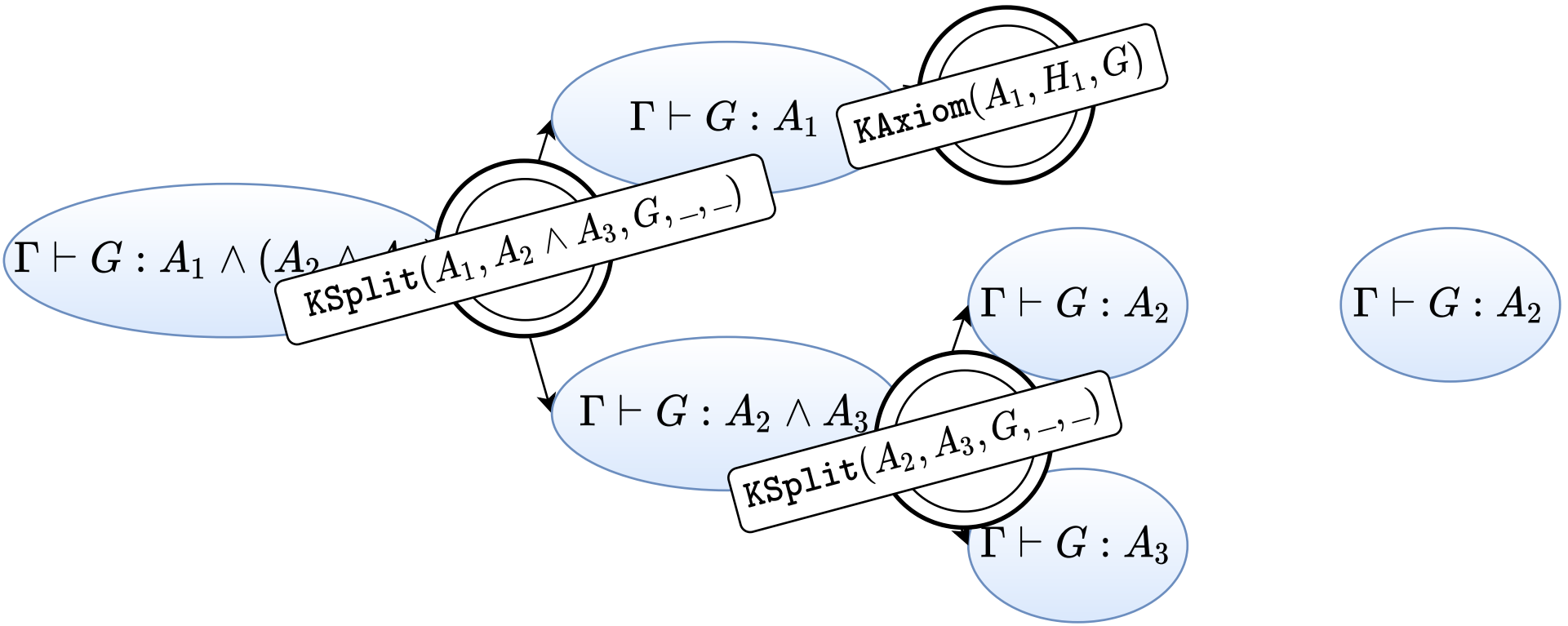
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Example: `blast` Transformation

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

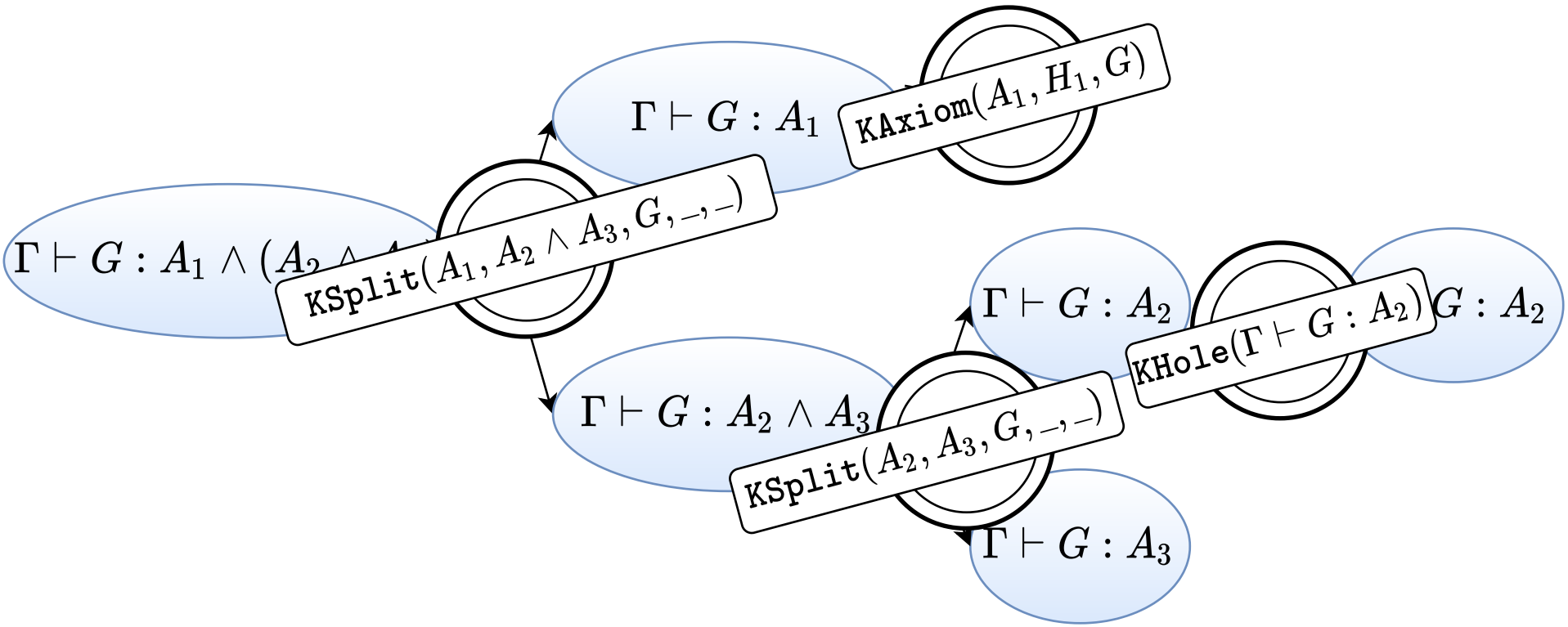
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Example: `blast` Transformation

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

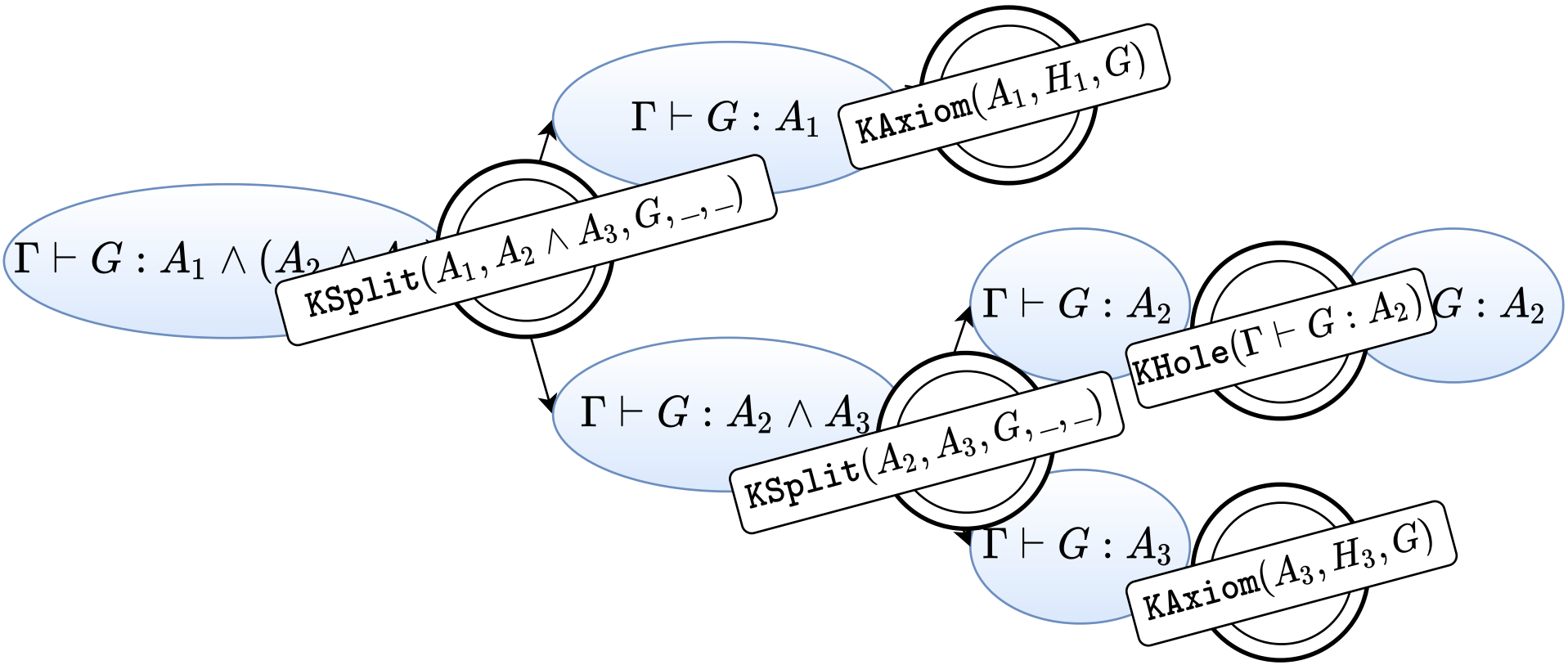
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Example: `blast` Transformation

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

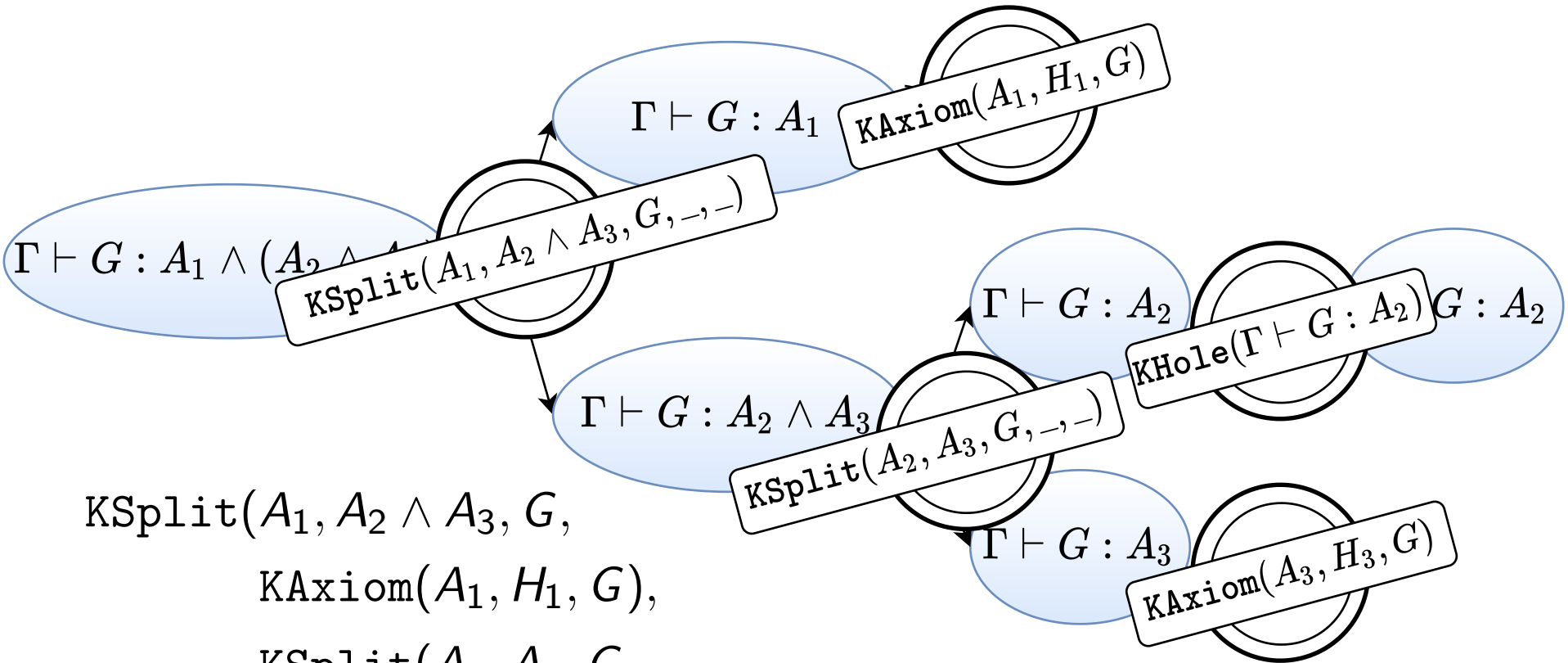
$T := \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3)$



Example: `blast` Transformation

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

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



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

Native Checker

Based on function `ccheck : kcert -> task -> bool`

Case `KSplit`(A_1, A_2, G, c_1, c_2):

- Applies to $\Gamma \vdash \Delta, G : A_1 \wedge A_2$
- Creates $T_1 \triangleq \Gamma \vdash \Delta, G : A_1$ and $T_2 \triangleq \Gamma \vdash \Delta, G : A_2$
- Returns `ccheck` c_1 T_1 `&&` `ccheck` c_2 T_2

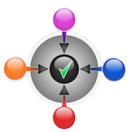
Correctness of *ccheck*

If *ccheck certif* T = *true* where *certif* has holes $T_1, \ldots, T_n$, then the validity of $T_1, \ldots, T_n$ implies the validity of T.

Lambdapi/CoC Checker

Objective: improve trust in our framework

- encode certificate steps in a trusted logical system, CoC
- translate tasks and certificates, use checker for this system

Lambdapi  logical framework, $\lambda\Pi$-calculus modulo rewriting

Lambdapi/CoC checker overview:

1. from $T \rightsquigarrow T_1, \ldots, T_n$, generate a Lambdapi type

$$sound(T_1, \ldots, T_n, T)$$

2. from the certificate, generate a Lambdapi term t
3. ask Lambdapi to check that $t : sound(T_1, \ldots, T_n, T)$

Lambdapi/CoC Checker: Example

Recall the application of `blast` $T \rightsquigarrow T_2$, where:

$$\begin{aligned} \Gamma &:= H_1 : A_1, H_3 : A_3 \\ T &:= \Gamma \vdash G : A_1 \wedge (A_2 \wedge A_3) \\ T_2 &:= \Gamma \vdash G : A_2 \end{aligned}$$

Produce the following Lambdapi instruction

```
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, ...
```

Lambdapi/CoC Checker: Results

Trusted code base:

- contains translation of tasks (CoC encoding)
- contains Lambdapi itself
- does *not* contain the proof term generation

Certificate steps are *defined* (instead of posited) in Lambdapi/CoC:

```
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)
```

Correctness of the Lambdapi/CoC checker

If $\mathit{sound}(T_1, \ldots, T_n, T)$ is inhabited, then the validity of $T_1, \ldots, T_n$ implies the validity of T.

Certificate Elaboration

Kernel certificates

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

Certificate Elaboration

Surface certificates

$\lambda\, T_1\ T_2.\ \texttt{SSplit}(G, T_1, T_2)$

- fewer parameters (A_1, A_2)
  ↳ easier to produce
- abstract over resulting tasks
  ↳ reusable
- "big" certificate steps
  ↳ easier to produce

Kernel certificates

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

Contributions

Framework for higher-order logic with interpreted theories

Core development:

- kernel certificates with holes
- Native checker
- translation procedure into Lambdapi/CoC
- library for certificates, integer library in Lambdapi
- surface certificates, elaboration, composition

Applications:

- translation of Why3 tasks into our framework
- $\sim$ 15 simple certifying transformations
- `blast`, `rewrite`, `induction`, `split`, `compute`

Future Work

Improve trust in Why3:

- certify other Why3 transformations
- integrate proofs by automated provers
- improve trust in other parts of Why3

Allow reuse of transformations in other contexts

`https://gitlab.inria.fr/why3/why3/tree/cert_pxtp/README_PXTP.md`