Bachelor's thesis - proposal talk:

Organizing a Library of Higher Order Problems

by Julian Backes on December 12, 2008

Advisor: Chad Brown
Supervisor: Gert Smolka

Contents

- Recap from the first talk
  - Our problem
  - Signature/Presentation/Provability
- A closer look at morphisms
  - Signature morphisms
  - Theory morphisms
- Status of the implementation
  - Imports

Recap

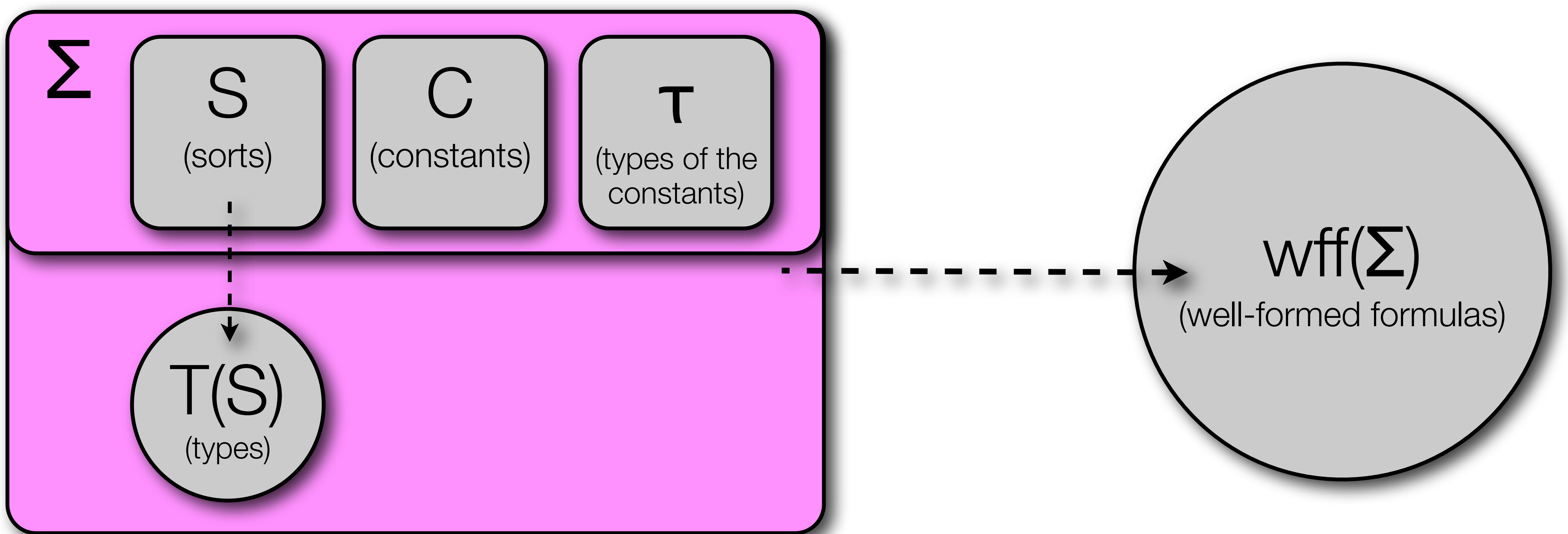
The story so far...

Our problem

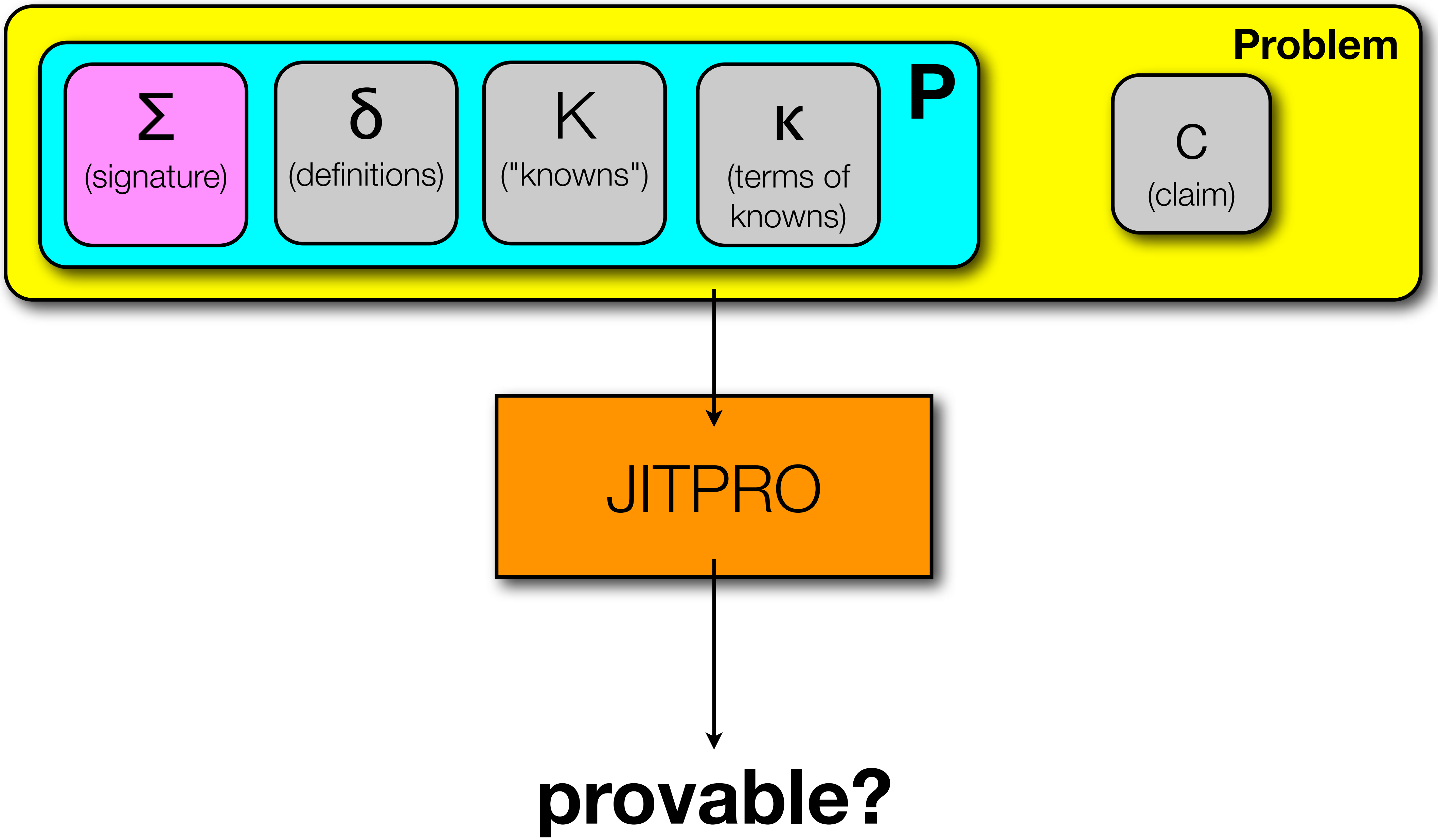
- The context: Proofs in Jitpro
- Goal: Reusing existing "theories" and proven claims
- Problem: Combining different small theories to bigger, more powerful theories
- Example:

```
sort I; // set elements
var x: I;
var S, T: I B; // subsets
term union = \S T x.S x | T x; // definition of union

sort V; // vertices
var v1, v2, v3: V;
const E: V V B; // edges
axiom !v1 v2. (E v1 v2) -> (E v2 v1); // undirected graph

claim !v1, v2, v3. (E v1 v3) ->
                   (union (E v1) (E v2)) v3
```

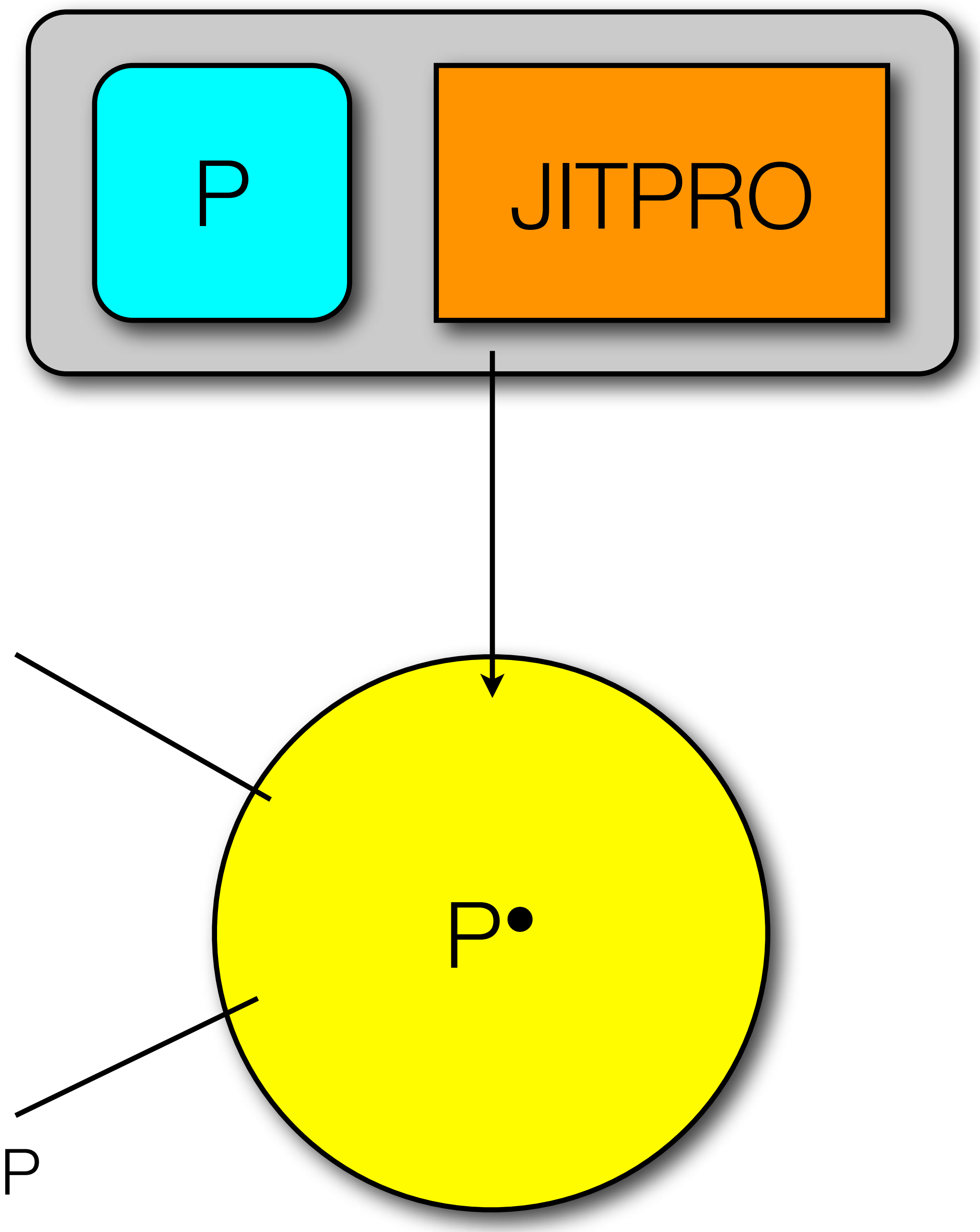
Signatures


Σ
S
(sorts)
C
(constants)
τ
(types of the constants)
T(S)
(types)
wff(Σ)
(well-formed formulas)

Presentations / Problems / Provability



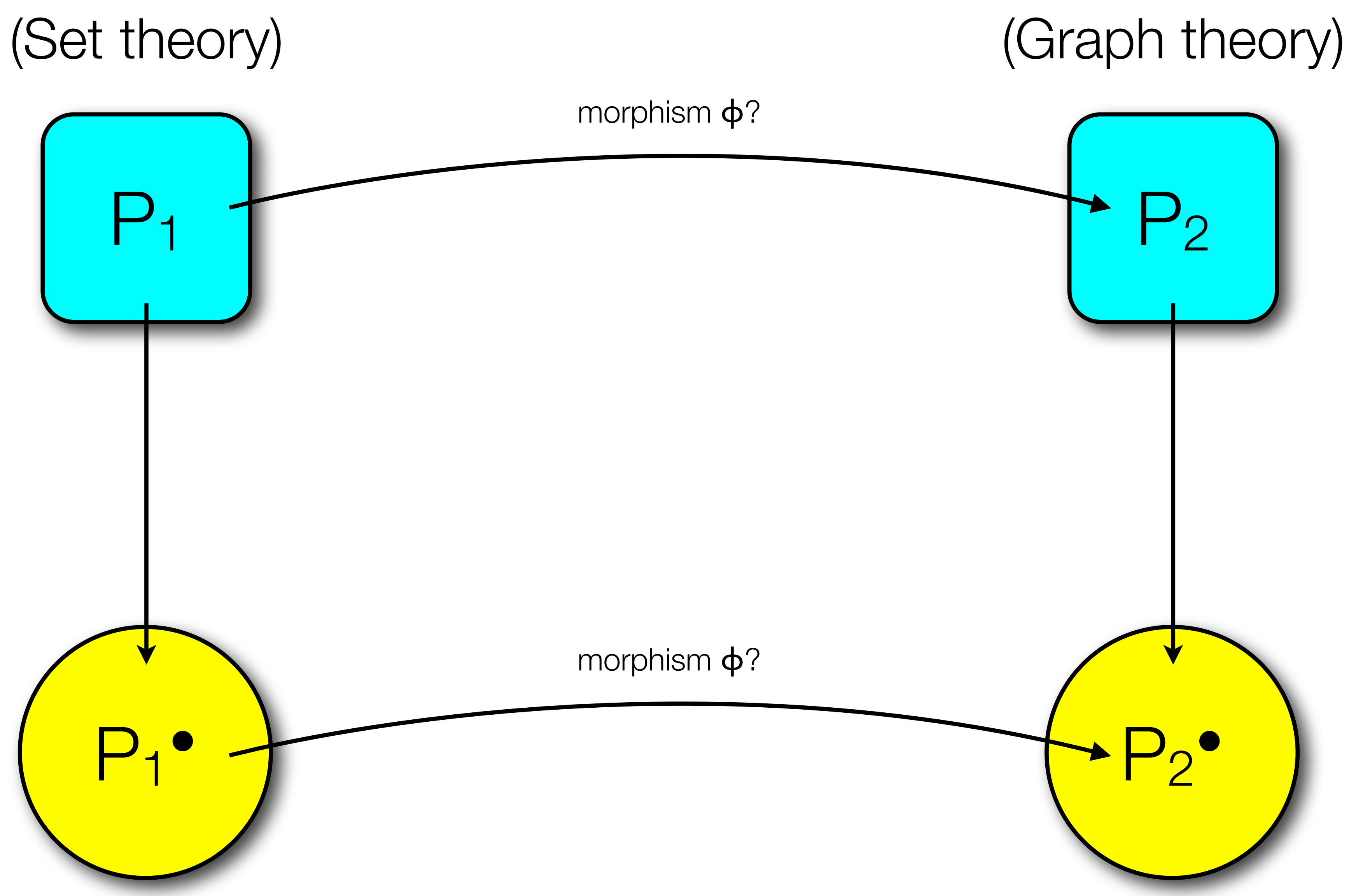
Closure / Theory



all claims provable in
JITPRO using P

closure of P /
theory presented by P

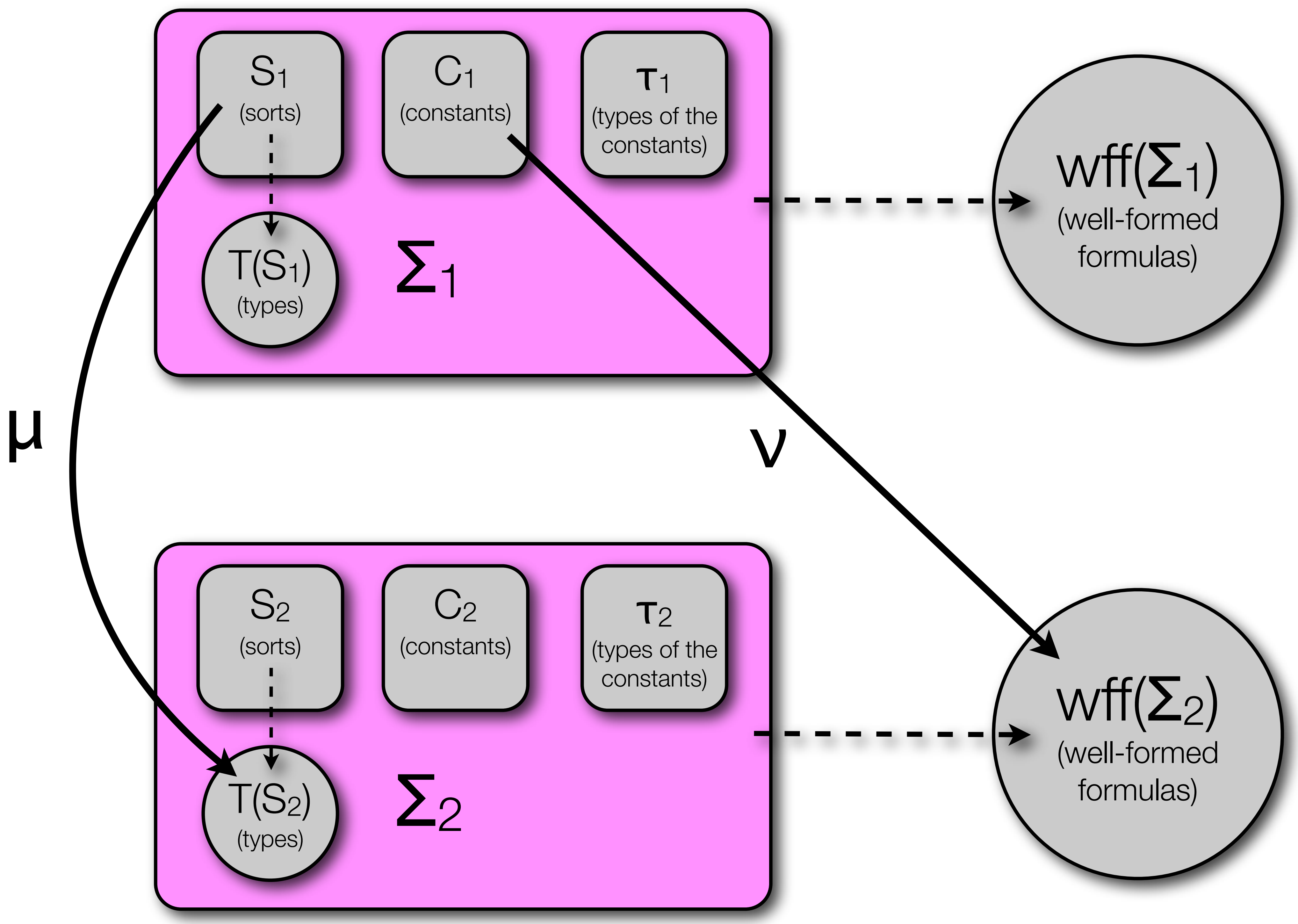
Morphisms (idea)



A closer look at morphisms

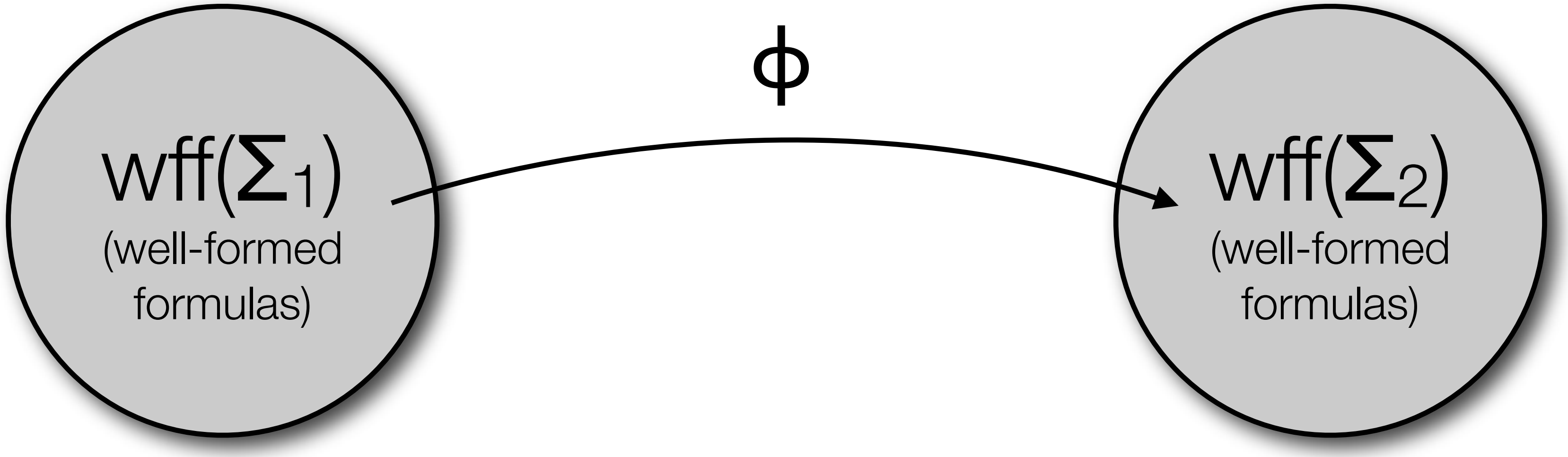
"Truth is invariant under change of notation"

Signature morphisms



Signature morphisms ctd

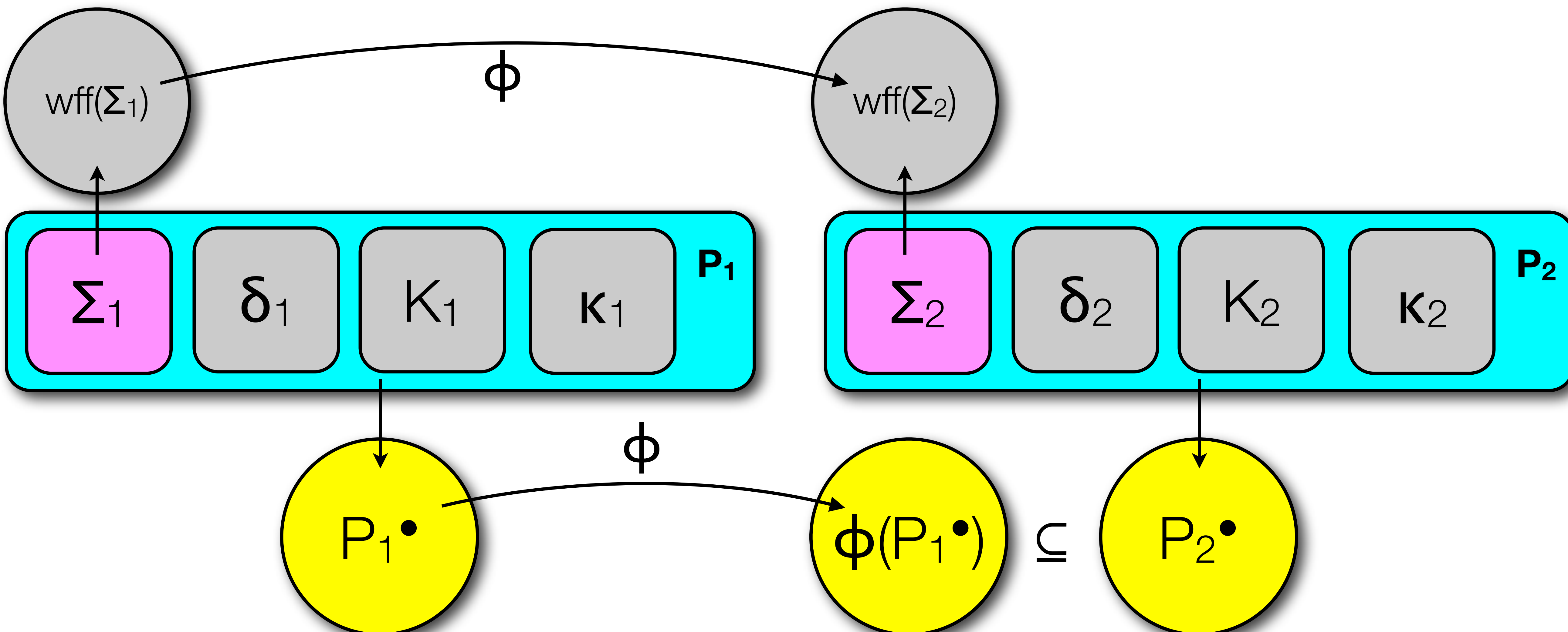
- Let Σ_1, Σ_2 and $\phi = (\mu, \nu)$ be given
- Recursively define $\mu^\bullet$ on types using μ
- Recursively define $\nu^\bullet$ on terms using $\mu^\bullet$ and ν



- ϕ is a ***signature morphism*** from Σ_1 and Σ_2

Theory morphisms

- Let $P_1 = (\Sigma_1, \delta_1, K_1, \kappa_1)$, $P_2 = (\Sigma_2, \delta_2, K_2, \kappa_2)$ and $\phi: \Sigma_1 \rightarrow \Sigma_2$ be given



- ϕ is a ***theory morphism*** iff $\phi(P_1^\bullet) \subseteq P_2^\bullet$ (preservation of provability)

Theory morphisms ctd

- Let $P_1 = (\Sigma_1, \delta_1, K_1, \kappa_1)$, $P_2 = (\Sigma_2, \delta_2, K_2, \kappa_2)$ and $\phi: \Sigma_1 \to \Sigma_2$ be given

- Problem: If we want to show that ϕ is a theory morphism, i.e. that we can reuse existing proofs, we first have to reprove everything which can be quite a lot of work.

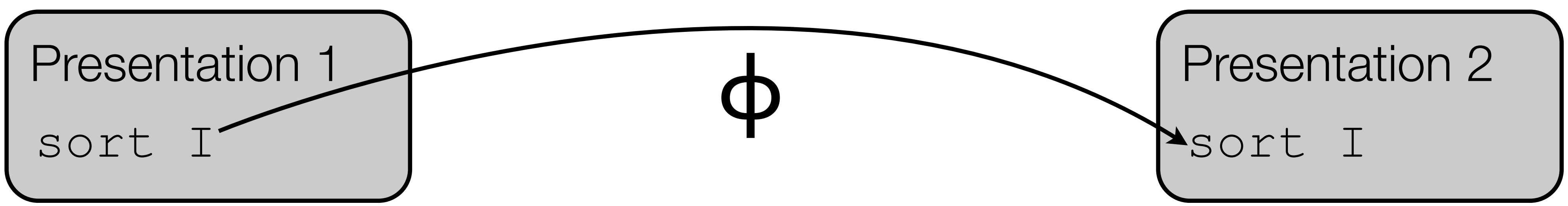
- Fortunately: **Presentation Lemma**: If $\phi(\kappa_1(k)) \in P_2^\bullet$ for all $k \in K_1$ and $(\phi(d) = \phi(\delta_1(d))) \in P_2^\bullet$ for all $d \in Dom(\delta)$ then ϕ is a theory morphism.
    - Proof: In my Bachelor's thesis ;-)

- => It is enough to check all knowns and definitions (which can be trivial as we will later see)

Status of the implementation

Morphisms in Jitpro

- Unfortunately, using only some implementation of pure morphisms is not very useful in practice:



- Assume, we want to reuse sort I in Presentation 2. Using morphisms, this would work as follows:
  - Define a sort I in Presentation 2
  - Map sort I of Presentation 1 to sort I of Presentation 2
- Quite useless, similar with constants, definitions...

Morphisms in Jitpro ctd

- We need a possibility to define a presentation and morph another presentation at the same time, so called ***imports***

- Imports are more powerful practical counterparts to the theory of morphisms

Presentation 1

```
sort I
term union = \C, D:I B.\x:I.(C x) | (D x)
```

Presentation 2

```
import "Presentation 1"
end
sort M
...
```

- Implicitly defines sort I and definition union and applies identity morphism

More complex import

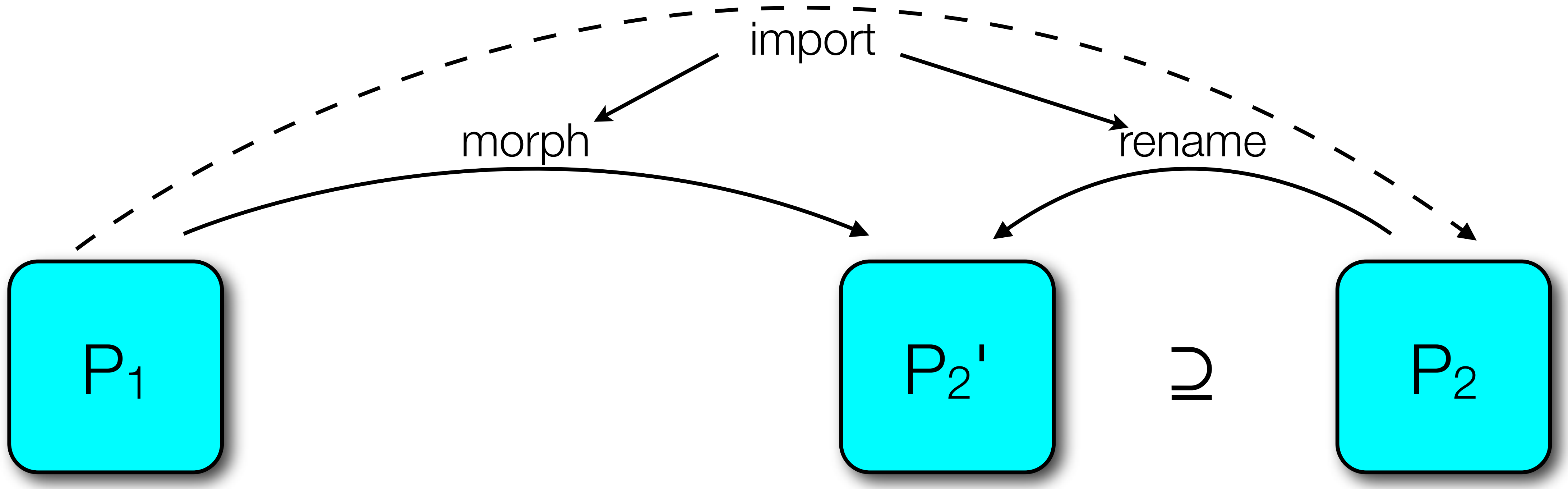
Presentation 1

```
sort I
term union = \C, D:I B.\x:I.(C x) | (D x)
```

Presentation 2

```
sort V; // vertices
var v1, v2, v3: V;
const E: V V B; // edges
axiom !v1 v2. (E v1 v2) -> (E v2 v1); // undirected graph

import "Presentation 1"
    morph sort I = V // morphs sort I to sort V
    rename term union union_vertices // redefines union, renames it to union_vertices
                                     // and applies morphism (union->union_vertices)
end

claim !v1, v2, v3. (E v1 v3) ->
                   (union_vertices (E v1) (E v2)) v3
```

How imports work



Preservation of provability

- What about the obligations for a theory morphisms?
  - Morphed knowns must be provable
  - Morphed constant = morphed definition must be provable
- When using `rename` for knowns or definitions, these proofs become trivial
- Otherwise: The corresponding obligation becomes a claim in the new presentation and has to be proven by the user
- Default import mode is `rename`

Questions?

Thank you for your attention
and enjoy your weekend!