Bachelor's thesis - first talk:

Organizing a Library of Higher Order Problems

by Julian Backes on November 21, 2008

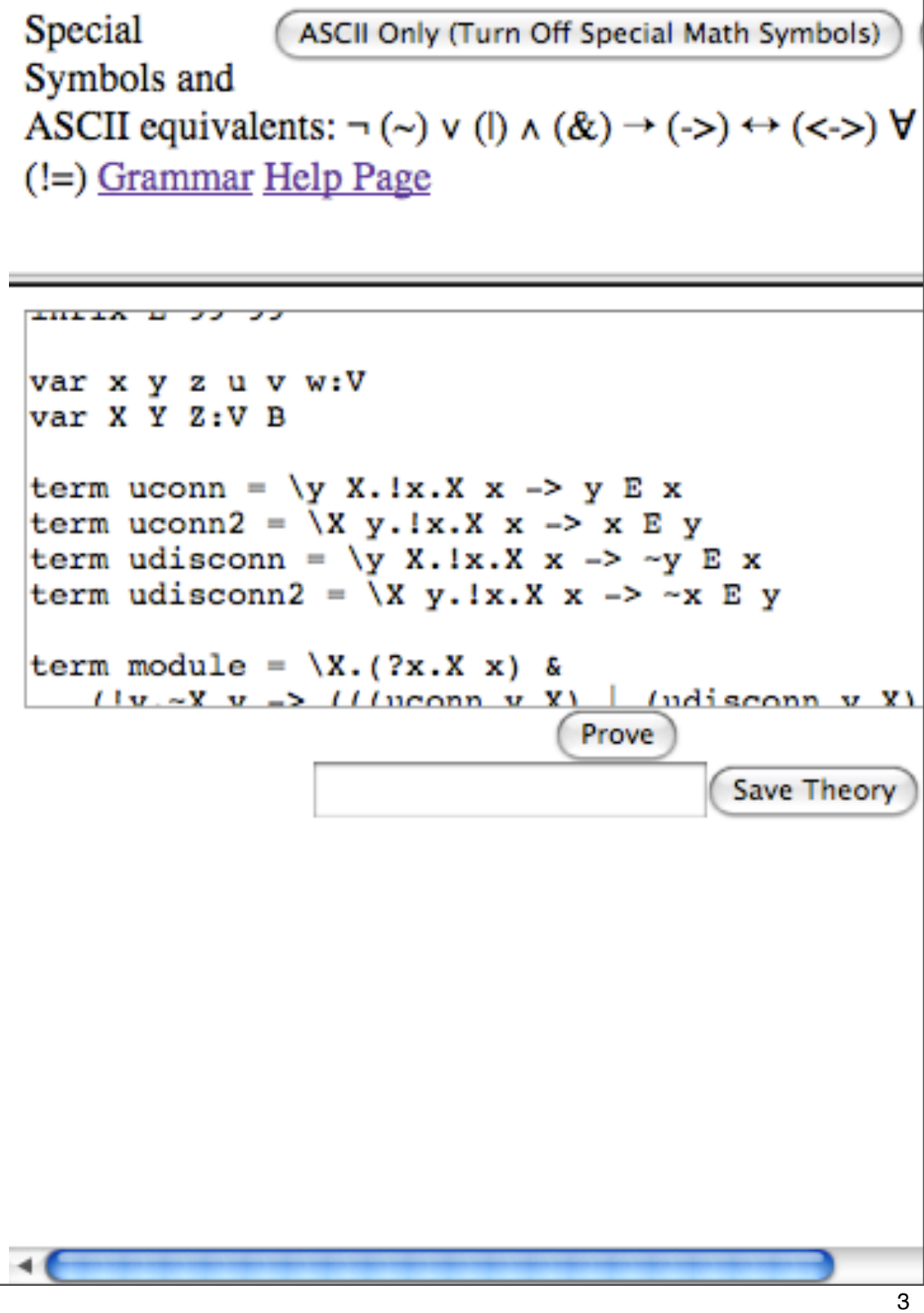
Advisor: Chad Brown
Supervisor: Gert Smolka

Contents

- Short introduction to JITPRO
- What's the problem about the problems?
- What has been done so far?
- What can we expect from this work?

Short introduction to JITPRO

What? A prover in Javascript?



Short introduction to JITPRO

- JITPRO is a "**J**avaScript **I**nteractive Higher-Order **T**ableau **Pro**ver"
- Developed by Chad Brown
- Completely written in JavaScript (!)
- Based on a set of refutation rules (more about them in [1])
- "Interactive" means it doesn't prove anything for you, just assists
- User has to decide which rule he applies in which way
- Let's have an example....

An easy example

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

(Hint: `E v1` is the set of all nodes connected to v1)

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

Doesn't work: Not well typed

Another easy example

```
sort i
const in:i i B
term exu = (\phi:i B.?x:i.phi x & (!y:i.phi y -> x = y))
axiom setextAx:(!A:i.!B:i.(!x:i.in x A <-> in x B) -> A = B)
const emptyset:i
axiom emptysetAx:(!x:i.~ (in x emptyset))
const setadjoin:i i i
axiom setadjoinAx:(!x:i.!A:i.!y:i.in y (setadjoin x A) <-> y = x | in y A)
const powerset:i i
axiom powersetAx:(!A:i.!B:i.in B (powerset A) <-> (!x:i.in x B -> in x A))
const setunion:i i
axiom setunionAx:(!A:i.!x:i.in x (setunion A) <-> (?B:i.in x B & in B A))
const omega:i
axiom omega0Ax:in emptyset omega
axiom omegaSAx:(!x:i.in x omega -> in (setadjoin x x) omega)
axiom omegaIndAx:(!A:i.in emptyset A & (!x:i.in x omega & in x A -> in (setadjoin x x) A) -> (!x:i.in x omega -> in x A))
axiom replAx:(!phi:i i B.!A:i.(!x:i.in x A -> exu (\y:i.phi x y)) -> (?B:i.!x:i.in x B <-> (?y:i.in y A & phi y x)))
axiom foundationAx:(!A:i.(?x:i.in x A) -> (?B:i.in B A & ~ (?x:i.in x B & in x A)))
axiom wellorderingAx:(!A:i.?B:i.(!C:i.in C B -> (!x:i.in x C -> in x A)) & (!x:i.!y:i.in x A & in y A -> (!C:i.in C B -> (in x C <-> in y C)) -> x = y) & (!C:i.!D:i.in C B & in D B -> (!x:i.in x C ->
in x D) | (!x:i.in x D -> in x C)) & (!C:i.(!x:i.in x C -> in x A) & (?x:i.in x C) -> (?D:i.?x:i.in D B & in x C & ~ (?y:i.in y D & in y C) & (!E:i.in E B -> (!y:i.in y E -> in y D) | in x E))))
const descr:(i B) i
axiom descrp:(!phi:i B.exu (\x:i.phi x) -> phi (descr (\x:i.phi x)))
const dsetconstr:i (i B) i
axiom dsetconstrI:(!A:i.!phi:i B.!x:i.in x A -> phi x -> in x (dsetconstr A (\y:i.phi y)))
axiom dsetconstrEL:(!A:i.!phi:i B.!x:i.in x (dsetconstr A (\y:i.phi y)) -> in x A)
axiom dsetconstrER:(!A:i.!phi:i B.!x:i.in x (dsetconstr A (\y:i.phi y)) -> phi x)
lemma exuE1:(!phi:i B.exu (\x:i.phi x) -> (?x:i.phi x & (!y:i.phi y -> x = y)))
const prop2set:B i
lemma prop2setE:(!phi:B.!x:i.in x (prop2set phi) -> phi)
lemma emptysetE:(!x:i.in x emptyset -> (!phi:B.phi))
lemma emptysetimpfalse:(!x:i.in x emptyset -> false)
lemma notinemptyset:(!x:i.~ (in x emptyset))
lemma exuE3e:(!phi:i B.exu (\x:i.phi x) -> (?x:i.phi x))
lemma setext:(!A:i.!B:i.(!x:i.in x A -> in x B) -> (!x:i.in x B -> in x A) -> A = B)
lemma emptyI:(!A:i.(!x:i.~ (in x A)) -> A = emptyset)
lemma noeltsimpempty:(!A:i.(!x:i.~ (in x A)) -> A = emptyset)
lemma setbeta:(!A:i.!phi:i B.!x:i.in x A -> (in x (dsetconstr A (\y:i.phi y)) <-> phi x))
term nonempty = (\x:i.~ (x = emptyset))
lemma nonemptyE1:(!A:i.nonempty A -> (?x:i.in x A))
lemma nonemptyI:(!A:i.!phi:i B.!x:i.in x A -> phi x -> nonempty (dsetconstr A (\y:i.phi y)))
lemma nonemptyI1:(!A:i.(?x:i.in x A) -> nonempty A)
lemma setadjoinIL:(!x:i.!y:i.in x (setadjoin x y))
lemma emptyinunitempty:in emptyset (setadjoin emptyset emptyset)
lemma setadjoinIR:(!x:i.!A:i.!y:i.in y A -> in y (setadjoin x A))
lemma setadjoinE:(!x:i.!A:i.!y:i.in y (setadjoin x A) -> (!phi:B.(y = x -> phi) -> (in y A -> phi) -> phi))
lemma setadjoinOr:(!x:i.!A:i.!y:i.in y (setadjoin x A) -> y = x | in y A)
lemma setoftrueEq:(!A:i.dsetconstr A (\x:i.true) = A)
lemma powersetI:(!A:i.!B:i.(!x:i.in x B -> in x A) -> in B (powerset A))
lemma emptyinPowerset:(!A:i.in emptyset (powerset A))
lemma emptyInPowerset:(!A:i.in emptyset (powerset A))
lemma powersetE:(!A:i.!B:i.!x:i.in B (powerset A) -> in x B -> in x A)
lemma setunionI:(!A:i.!x:i.!B:i.in x B -> in B A -> in x (setunion A))
lemma setunionE:(!A:i.!x:i.in x (setunion A) -> (!phi:B.(!B:i.in x B -> in B A -> phi) -> phi))
lemma subPowSU:(!A:i.!x:i.in x A -> in x (powerset (setunion A)))
lemma exuE2:(!phi:i B.exu (\x:i.phi x) -> (?x:i.!y:i.phi y <-> y = x))
lemma nonemptyImpWitness:(!A:i.nonempty A -> (?x:i.in x A & true))
lemma uniqinunit:(!x:i.!y:i.in x (setadjoin y emptyset) -> x = y)
lemma notinsingleton:(!x:i.!y:i.~ (x = y) -> ~ (in y (setadjoin x emptyset)))
lemma eqinunit:(!x:i.!y:i.x = y -> in x (setadjoin y emptyset))..........................
```

```
claim woz2A:(!A:i.!R:i.breln1 A R -> (!S:i.breln1 A S -> (!T:i.breln1 A T ->
breln1compset A (binunion R S) T = binunion (breln1compset A R T) (breln1compset A S
T))))
```

What's the problem about the problems?

What is a problem?

Some Definitions

- Let S be a set of sorts with B (boolean sort) ∈ S
  - induces a set T(S) of possible types

`sort V;` ⇒ S = {V, B}

T(S) = {V, B, B→B, V→V, V→B, V→V→B....}

- Let C be a set of names ("constants) and τ: C→T(S)

`const E: V V B;` ⇒ C = {E}, τ(E) = V→V→B

- Σ = (S, C, τ) called **signature**

- Σ induces a set of well typed, closed terms wff(Σ)

wff(Σ) = {λx:B.x, λx:V y:V. E x y...}

Some Definitions ctd

- In the following, let some Σ be given

- Let K be a set of names ("knowns") and $\kappa: K \to wff_B(\Sigma)$

```
axiom undir = !x y. E x y -> E y x
```

⇒ K = {undir}, κ(undir) = ∀x, y. (E x y) implies (E y x)

- Let $\delta: C \supseteq D \to wff(\Sigma)$ give us definitions for some constants

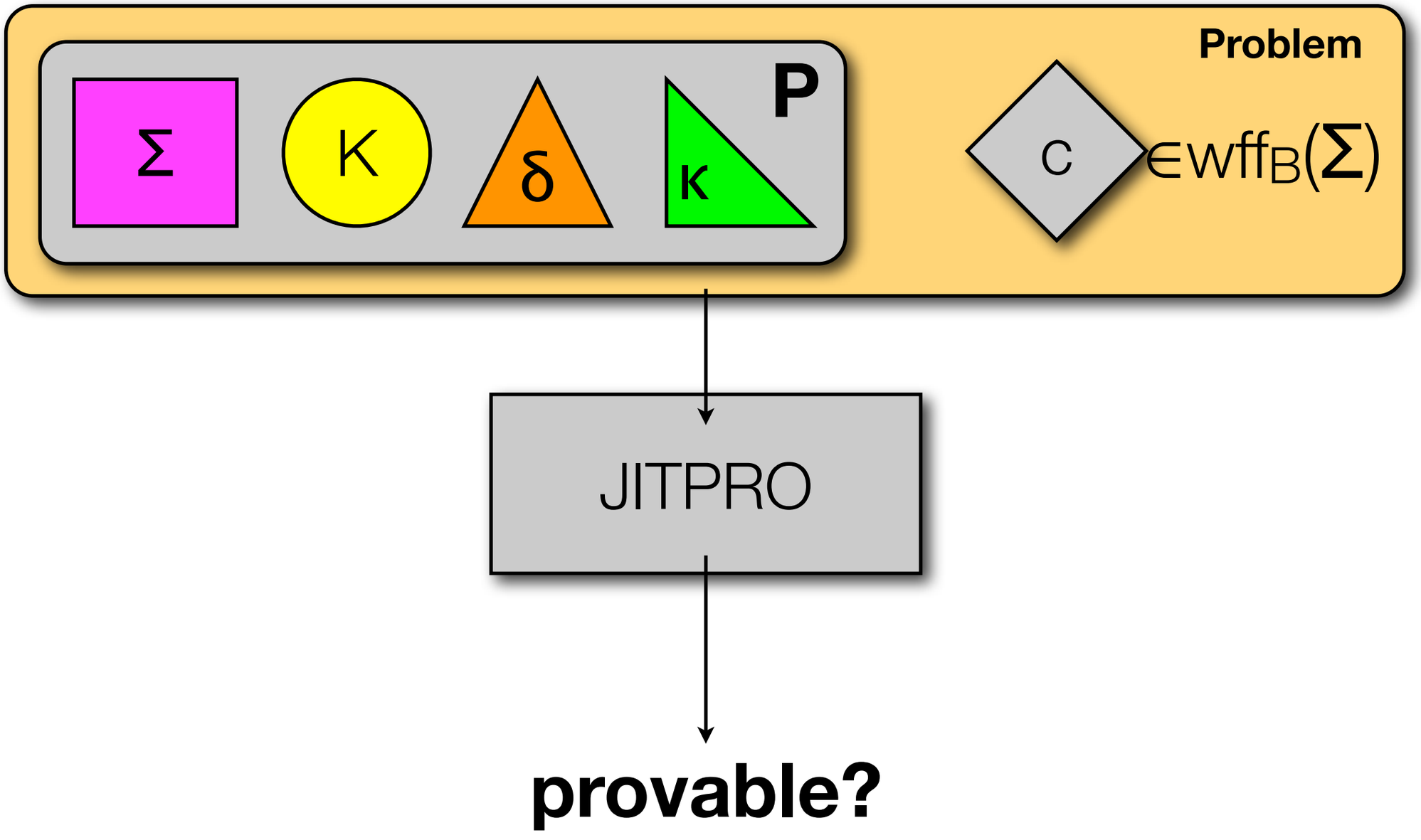
```
term union = \A B x.A x | B x
```

⇒ (C = {E, union}, τ(union) = (V→B)→(V→B)→V→B,)

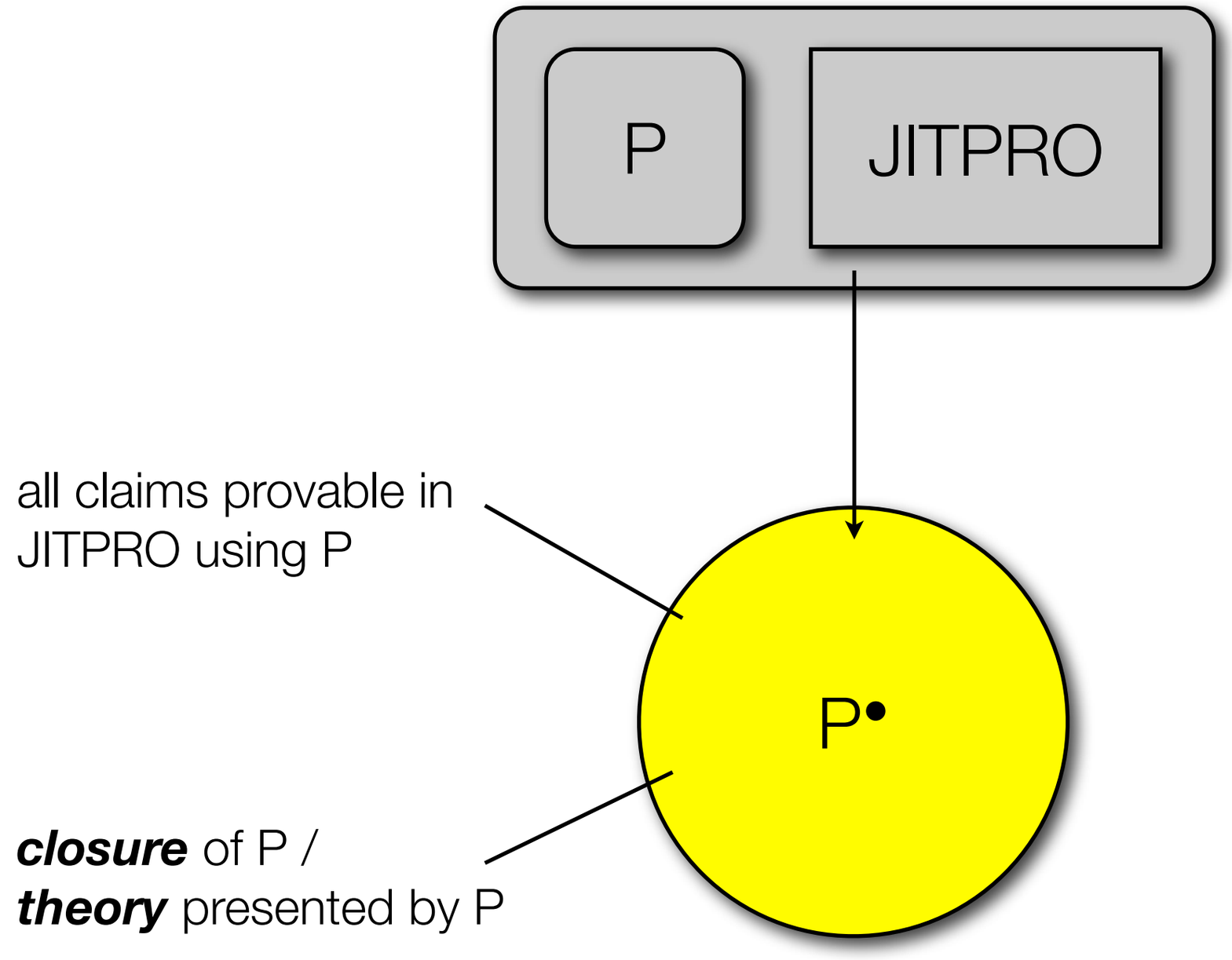
δ(union) = λA:V→B, B:V→B, x:V. (A x) or (B x)

- P = (Σ, δ, K, κ) called **presentation**

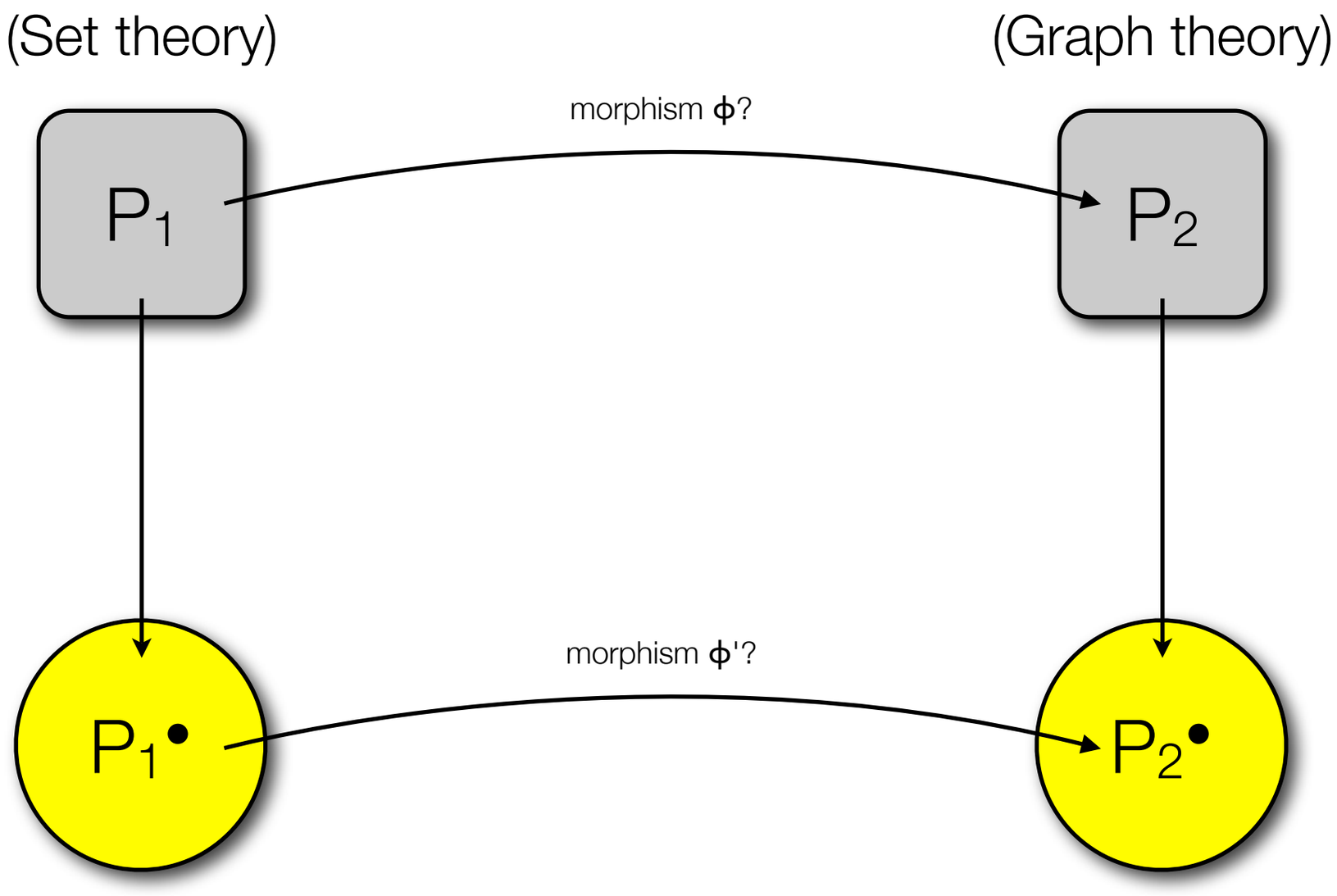
Problem / Provability



Closure / Theory



Morphisms



What has been done so far?

a lot...

What has been done so far?

- R.M. Burstall, J.A. Goguen: "Putting theories together to make specifications" (1977)
- J.A. Goguen, R.M. Burstall: "Institutions: Abstract Model Theory for Specification and Programming" (1992)
- W.M. Farmer, J.D. Guttman, F.J. Thayer: "Little theories" (1992)
- W.M. Farmer, J.D. Guttman, F.J. Thayer: "IMPS: An Interactive Mathematical Proof System" (1993)
- D. Hutter: "Management of Change in Structured Verification" (2000)
- G. Sutcliffe, C. Suttner: "TPTP: Thousands of Problems for Theorem Provers"

What can we expect from this work?

The example from the beginning

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

This is the way it has been done so far!

The second example from the beginning

```
sort i
const in:i i B
term exu = (\phi:i B.?x:i.phi x & (!y:i.phi y -> x = y))
axiom setextAx:(!A:i.!B:i.(!x:i.in x A <-> in x B) -> A = B)
const emptyset:i
axiom emptysetAx:(!x:i.~ (in x emptyset))
const setadjoin:i i i
axiom setadjoinAx:(!x:i.!A:i.!y:i.in y (setadjoin x A) <-> y = x | in y A)
const powerset:i i
axiom powersetAx:(!A:i.!B:i.in B (powerset A) <-> (!x:i.in x B -> in x A))
const setunion:i i
axiom setunionAx:(!A:i.!x:i.in x (setunion A) <-> (?B:i.in x B & in B A))
const omega:i
axiom omega0Ax:in emptyset omega
axiom omegaSAx:(!x:i.in x omega -> in (setadjoin x x) omega)
axiom omegaIndAx:(!A:i.in emptyset A & (!x:i.in x omega & in x A -> in (setadjoin x x) A) -> (!x:i.in x omega -> in x A))
axiom replAx:(!phi:i i B.!A:i.(!x:i.in x A -> exu (\y:i.phi x y)) -> (?B:i.!x:i.in x B <-> (?y:i.in y A & phi y x)))
axiom foundationAx:(!A:i.(?x:i.in x A) -> (?B:i.in B A & ~ (?x:i.in x B & in x A)))
axiom wellorderingAx:(!A:i.?B:i.(!C:i.in C B -> (!x:i.in x C -> in x A)) & (!x:i.!y:i.in x A & in y A -> (!C:i.in C B -> (in x C <-> in y C)) -> x = y) & (!C:i.!D:i.in C B & in D B -> (!x:i.in x C ->
in x D) | (!x:i.in x D -> in x C)) & (!C:i.(!x:i.in x C -> in x A) & (?x:i.in x C) -> (?D:i.?x:i.in D B & in x C & ~ (?y:i.in y D & in y C) & (!E:i.in E B -> (!y:i.in y E -> in y D) | in x E))))
const descr:(i B) i
axiom descrp:(!phi:i B.exu (\x:i.phi x) -> phi (descr (\x:i.phi x)))
const dsetconstr:i (i B) i
axiom dsetconstrI:(!A:i.!phi:i B.!x:i.in x A -> phi x -> in x (dsetconstr A (\y:i.phi y)))
axiom dsetconstrEL:(!A:i.!phi:i B.!x:i.in x (dsetconstr A (\y:i.phi y)) -> in x A)
axiom dsetconstrER:(!A:i.!phi:i B.!x:i.in x (dsetconstr A (\y:i.phi y)) -> phi x)
lemma exuE1:(!phi:i B.exu (\x:i.phi x) -> (?x:i.phi x & (!y:i.phi y -> x = y)))
const prop2set:B i
lemma prop2setE:(!phi:B.!x:i.in x (prop2set phi) -> phi)
lemma emptysetE:(!x:i.in x emptyset -> (!phi:B.phi))
lemma emptysetimpfalse:(!x:i.in x emptyset -> false)
lemma notinemptyset:(!x:i.~ (in x emptyset))
lemma exuE3e:(!phi:i B.exu (\x:i.phi x) -> (?x:i.phi x))
lemma setext:(!A:i.!B:i.(!x:i.in x A -> in x B) -> (!x:i.in x B -> in x A) -> A = B)
lemma emptyI:(!A:i.(!x:i.~ (in x A)) -> A = emptyset)
lemma noeltsimpempty:(!A:i.(!x:i.~ (in x A)) -> A = emptyset)
lemma setbeta:(!A:i.!phi:i B.!x:i.in x A -> (in x (dsetconstr A (\y:i.phi y)) <-> phi x))
term nonempty = (\x:i.~ (x = emptyset))
lemma nonemptyE1:(!A:i.nonempty A -> (?x:i.in x A))
lemma nonemptyI:(!A:i.!phi:i B.!x:i.in x A -> phi x -> nonempty (dsetconstr A (\y:i.phi y)))
lemma nonemptyI1:(!A:i.(?x:i.in x A) -> nonempty A)
lemma setadjoinIL:(!x:i.!y:i.in x (setadjoin x y))
lemma emptyinunitempty:in emptyset (setadjoin emptyset emptyset)
lemma setadjoinIR:(!x:i.!A:i.!y:i.in y A -> in y (setadjoin x A))
lemma setadjoinE:(!x:i.!A:i.!y:i.in y (setadjoin x A) -> (!phi:B.(y = x -> phi) -> (in y A -> phi) -> phi))
lemma setadjoinOr:(!x:i.!A:i.!y:i.in y (setadjoin x A) -> y = x | in y A)
lemma setoftrueEq:(!A:i.dsetconstr A (\x:i.true) = A)
lemma powersetI:(!A:i.!B:i.(!x:i.in x B -> in x A) -> in B (powerset A))
lemma emptyinPowerset:(!A:i.in emptyset (powerset A))
lemma emptyInPowerset:(!A:i.in emptyset (powerset A))
lemma powersetE:(!A:i.!B:i.!x:i.in B (powerset A) -> in x B -> in x A)
lemma setunionI:(!A:i.!x:i.!B:i.in x B -> in B A -> in x (setunion A))
lemma setunionE:(!A:i.!x:i.in x (setunion A) -> (!phi:B.(!B:i.in x B -> in B A -> phi) -> phi))
lemma subPowSU:(!A:i.!x:i.in x A -> in x (powerset (setunion A)))
lemma exuE2:(!phi:i B.exu (\x:i.phi x) -> (?x:i.!y:i.phi y <-> y = x))
lemma nonemptyImpWitness:(!A:i.nonempty A -> (?x:i.in x A & true))
lemma uniqinunit:(!x:i.!y:i.in x (setadjoin y emptyset) -> x = y)
lemma notinsingleton:(!x:i.!y:i.~ (x = y) -> ~ (in y (setadjoin x emptyset)))
lemma eqinunit:(!x:i.!y:i.x = y -> in x (setadjoin y emptyset))..........................
```

Anyone wants to morph it?

What can we expect?

- Theoretical foundations for storing, retrieving, combining, morphing (...) higher order "theories" (= presentations) and problems
- A "state-of-the-art-web-2.0" implementation of these theoretical foundations
  - A database for storing theories/problems
  - An interface (a webpage ;-)) for searching the database, retrieving, combining, morphing (...) theories/problems
  - A possibility to present results in different "formats", e.g. JITPRO syntax, THF0 (TPTP syntax)...
- A direct connection from JITPRO to that system

Questions?

Thank you for your attention!

References

- [1] G. Smolka, C.E. Brown: "Introduction to Computational Logic. Lecture Notes SS 2008" (2008)

- See slide 14...