By Reason and Authority: A System for Authorization of Proof-Carrying Code

Nathan Whitehead
Dept. Computer Science
UC Santa Cruz
nwhitehe@cs.ucsc.edu

Martín Abadi
Dept. Computer Science
UC Santa Cruz
abadi@cs.ucsc.edu

George Necula
Computer Science Division
UC Berkeley
necula@cs.berkeley.edu

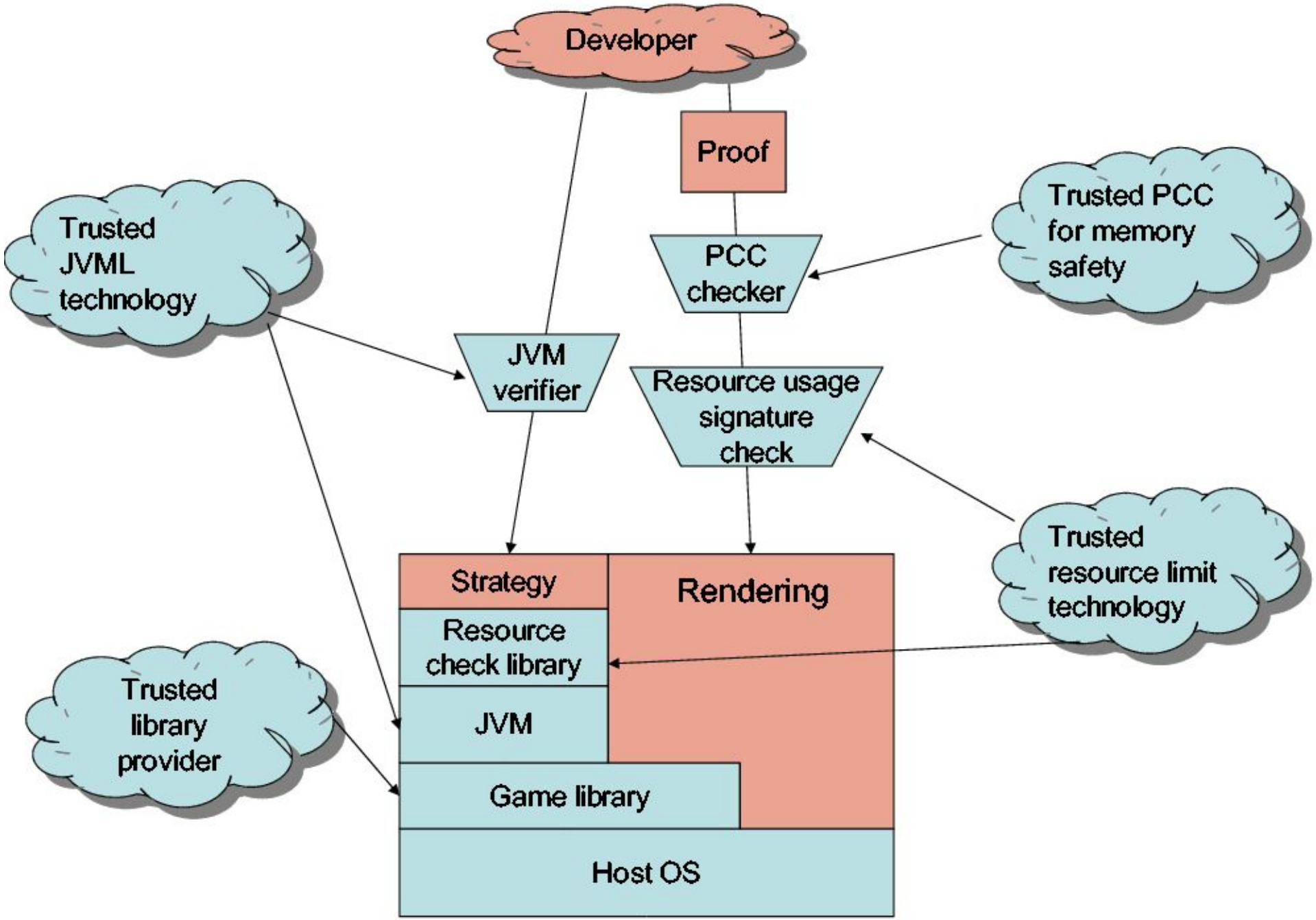
Motivation

- Extensible systems are common
  - web browser plug-ins
  - automatic software upgrades
  - .NET libraries
- Security of plug-ins and extensions is important
- Digital signatures solve some problems...
- Proof systems solve other problems...

Signatures and Proofs

- Digital signatures
  - Conceptually simple
  - Existing public-key algorithms work
  - Clear blame trail when something fails
  - Efficient to generate and check
- Proofs
  - Eliminate accidental and intentional errors
  - No relationship needed between code producer and trusted authority
  - Efficient to check, using proof-carrying code (PCC)

Game/Cell Phone Example



Authority and Proofs

- We have a system that can:
  - reason about properties by assertion from trusted authorities
  - AND check provided proofs
- It can mix both types of reasoning
  - prove a property based on a fact asserted by authority
  - use assertions to know which proof system to use
  - combine components proved secure in different systems

`sat` and `believe`

`sat(P)` means we have a proof of proposition `P`

`believe(P)` means we believe there is a proof of proposition `P`

- Distinguish *truth* from *assumption*
- Transmitted proofs will appear as `sat` formulas
- `sat` may not appear as the conclusion of a policy rule, `believe` may

Policy Excerpts

```
use R_TAL in
   forall P:talprg
   mayrun(P):-
      believe(safe P),
      believe(economical P)
end
```

```
use R_TAL in
   forall P:talprg
   forall Q:talprg
   forall R:talprg
   believe(economical R):-
      sat(link P Q R),
      resource_signer says enforcer(Q)
end
```

BLF

- Combines Binder and LF
  - Binder is a modal “says” logic based on Datalog
  - LF is a logical framework based on dependent types that describes proof systems
- Straightforward syntax and semantics
- Conservatively extends LF
  - nothing more or less is provable than in LF for any proof system
- Decidable (for well-behaved policies)
  - no proof search inside proof systems
  - no negation

Says Logic (English)

Alice
"Program P_0 is safe"

Bob
"Trust Alice about program safety"

User
"Run program P_0"

Reference Monitor
"Trust Bob about program safety"
"If P is safe and the user wants to run P, then run P"

Says Logic (Binder)

Alice

```
safe(P0)
```

Bob

```
safe(P):-
   Alice says safe(P)
```

User

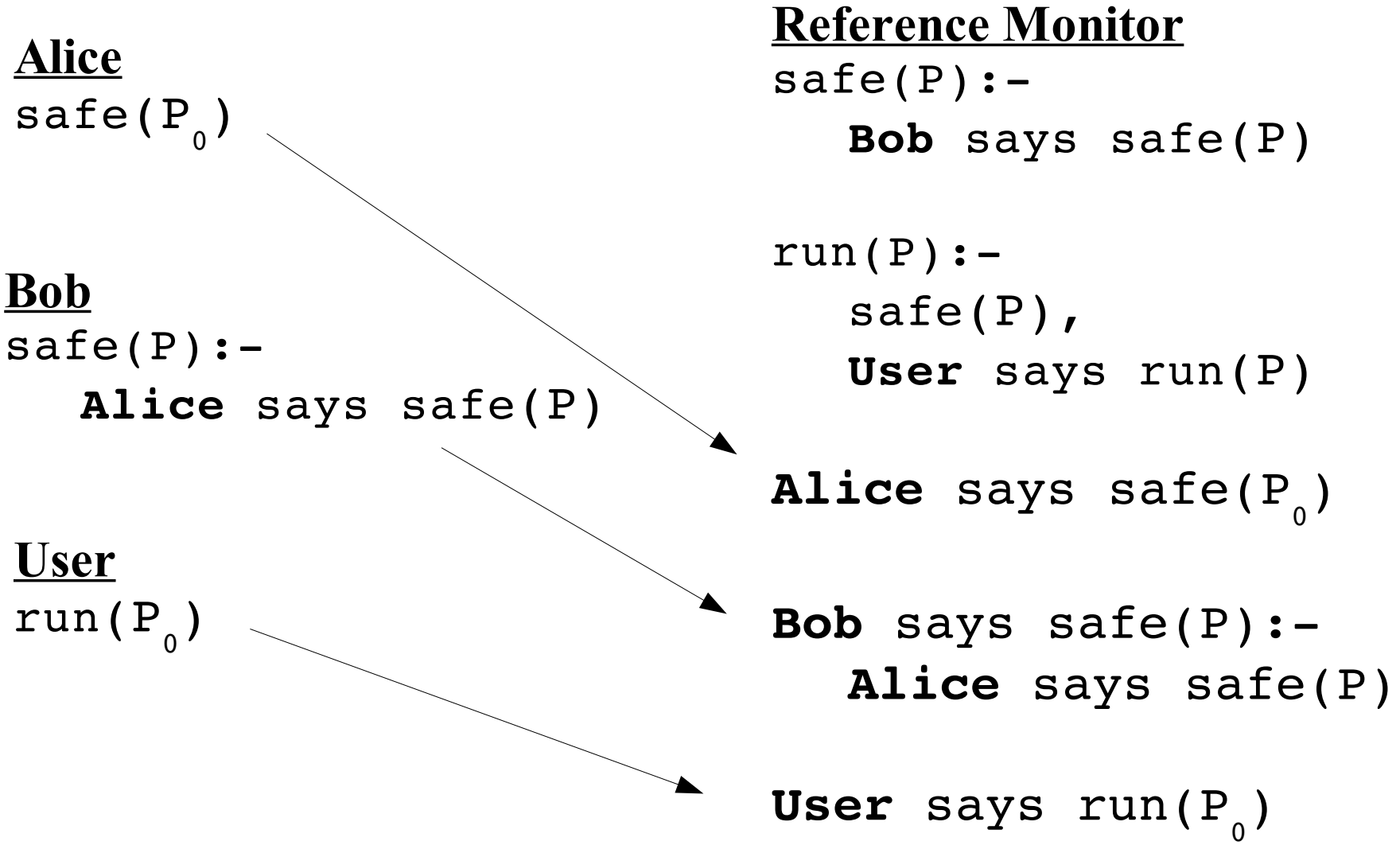
```
run(P0)
```

Reference Monitor

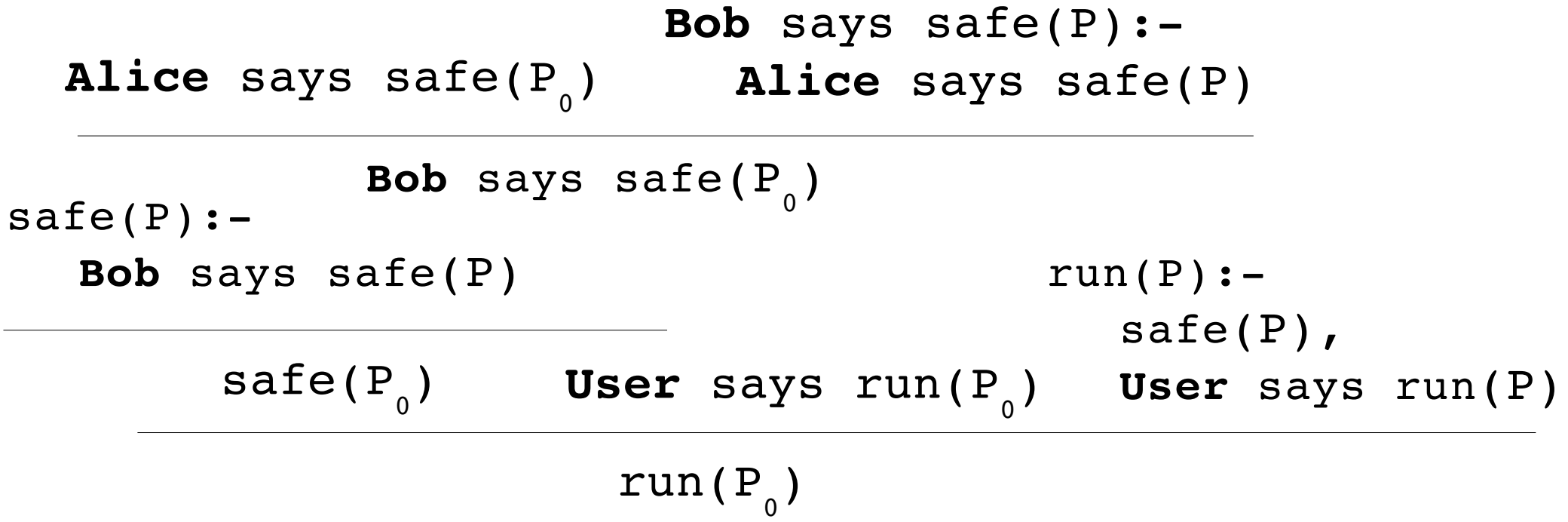
```
safe(P):-
   Bob says safe(P)

run(P):-
   safe(P),
   User says run(P)
```

Says Logic (Import)



Says Logic (Deduction)



LF Proof Systems

```
nat : type.
0 : nat.
s : nat → nat.
even : nat → type.
ax0 : even 0.
axadd2 : even X → even (s (s X)).
```

Propositions are *types*

`(even (s (s 0)))`
is the proposition that two is even

Proofs are *objects*

`(axadd2 ax0)`
is a proof of the proposition

Proving Program Properties in LF

- For high-level languages
  - encode “correctness” derivations into LF directly
  - encode VCgen as a predicate and encode logic in LF
- Another approach is to use TAL
  - encode assembly type rules into LF axioms
- Another way: foundational PCC
  - encode high-level proof rules directly (i.e. type rules)
  - also encode proof that high-level proofs imply low-level safety (i.e. proof of soundness of type rules)

Policy Excerpts (Revisited)

```
use R_TAL in
   forall P:talprg
   mayrun(P):-
      believe(safe P),
      believe(economical P)
end
```

```
use R_TAL in
   forall P:talprg
   forall Q:talprg
   forall R:talprg
   believe(economical R):-
      sat(link P Q R),
      resource_signer says enforcer(Q)
end
```

Other Pieces of Our Work

- More examples
  - endorsing rulesets
  - trust annotations on assertions
  - trust in the λ-calculus
- Precise definition of syntax and proof rules
- Proof of conservativity
- Decision algorithm
- Rudimentary implementation

Related Work

- Several variations of distributed logics for security, various PCC efforts
- Reason and authority are often combined in security, usually implicitly
  - X.509 certificates are from authorities; reasoning about transitivity verifies chains of trust
  - JVM does static analysis before running code; JVM is provided by trusted authority
- (See paper for more discussion)

Conclusions

- BLF combines Binder and LF
  - allows security decisions based on authority and reason
  - many types of interactions between proofs and assertions possible
- Makes PCC more practical
  - don't need to prove everything; can “cheat” with digital signatures
  - can combine results from multiple proof systems
  - makes trust of proof rules explicit

Future Work

- Integrate BLF into a PCC framework such as the Open Verifier project
  - present a complete story of a real application within the framework
  - modify logic as necessary
- Explore extensions such as function symbols
  - makes logic undecidable
  - allows principals to make more interesting statements
  - opens up new application domains

The End

Trusting Rulesets Example

```
forallrules R
   use R in
      forall P:prg
      mayrun(P):-
         sat(safe P),
         useruleset(R)
   end

useruleset(R):-
   pcc-provider says useruleset(R)
```

Trust in the λ-Calculus

- Encode type system of Ørbæk and Palsberg
- Typed λ-calculus with trust annotations
  - example types: α^{tr}, $(\alpha^{tr} \rightarrow \beta^{tr})^{dis}$
  - "trust" construct forces output to be trusted
- Require a digital signature from auditor for every occurrence of "trust"
  - Provider: (audit E_1)→(audit E_2)→(typecheck E)
- Also works for information flow and declassification

Java Policy Excerpt

```
use R_TAL in
   forall P:talprg
   believe(safe P):-
      use R_Java in
         exists Q:javaprg
            sat(typechecks Q),
            sat(compiles Q P)
      end
end
```

Deciding Queries

- Use bottom-up Datalog evaluation
- While the set of facts is still growing:
  - Add all direct inferences
  - Combine existing LF terms in any legal way to generate new LF terms
  - Update `sat` and `believe` atoms for LF terms
- Answer query using fixed point database of facts
- No proof search within LF signatures; instead find all ways to combine existing facts

Policy Constraints

- For termination, LF terms *mentioned in policies* must be well-behaved
  - `(even X → good X)` is OK
  - `(even X → even (s (s X)))` is not
  - no restrictions on LF signatures themselves
- Policies may not include `sat` as conclusion of rule
- Import of statements is partial function
  - `says` is never nested, as in Binder

Intuitionistic Logic in LF

```
form : type.
pf : form → type.
true : form.
false : form.
and : form → form → form.
imp : form → form → form.
true_i : pf true.
and_i : pf X → pf Y → pf (and X Y).
and_el : pf (and X Y) → pf X.
and_er : pf (and X Y) → pf Y.
imp_i : (pf X → pf Y) → pf (imp X Y).
imp_e : pf (imp X Y) → pf X → pf Y.
```

```
imp_i ([A:pf (and P (imp P Q))]
  imp_e (and_er A) (and_el A))
: pf (imp (and P (imp P Q)) Q)
```

proof

proposition
$P \wedge (P \rightarrow Q) \rightarrow Q$