LBTrust: Declarative Reconfigurable Trust Management

Bill Marczak*, Dave Zook†, Wenchao Zhou*, Molham Aref†, Boon Thau Loo*

*University of Pennsylvania
†LogicBlox

What is “Trust Management”?

- Trust management is broadly defined as:
  - Assigning credentials (rights) to principals (users) to perform actions
  - Delegating among principals
  - Enforce access control policies in a multi-user environment
- Logic representation/reasoning:
  - Logical analysis of new security protocols
  - Declarative interface for implementing security policies
  - Several runtime systems based on distributed Datalog/Prolog
- Binder, a simple representative language:

At alice:

r1: access(P,O,read) ← good(P).
r2: access(P,O,read) ← bob says access(P,O,read).

“In alice's context, any principal P may access object O in read mode if
P is good (R1) or,
bob says P may do so (R2 - delegation)”

(Non-Exhaustive) Survey of Trust Management Languages

	Authentication	Delegation	Conditional Re-Delegation	Threshold Structures	Type System
Aura	Y	Y*	Y?	Y	Y
Binder	Y	Y*	N	N	N
Cassandra	Y	Y*	Y	Y	Y
D1LP	Y	Y	Y (depth/width)	Y	N
KeyNote	Y	Y	N	Y	N
SD3	Y	Y*	N	N	N
SeNDLoG	Y	Y*	N	Y	N
SPKI/SDSI	Y	Y*	Y (boolean)	Y	N

- Problem: too many languages, features, separate runtime systems, hard to compare and reuse
- Our goal: A unified declarative framework to enable all of these languages

Key Ideas of LBTrust

- Constraints: type safety, program correctness, security
- Meta-programmability
  - Meta-model: rules as data [VLDB 08]
  - Meta-rules (code generation)
  - Meta-constraints (constraint + reflection)
- Customizable partitioning, distribution, and communication
- Extensible predicates for cryptographic primitives

Constraints and Types

fail() ← access(P,O,M), !principal(P).

negation

"<u>let</u> fail() whenever access(P,O,M) and not principal(P)"

access(P,O,M) → principal(P).

"whenever access(P,O,M), <u>require</u> principal(P)"

access(P,O,M) → principal(P), object(O), mode(M).

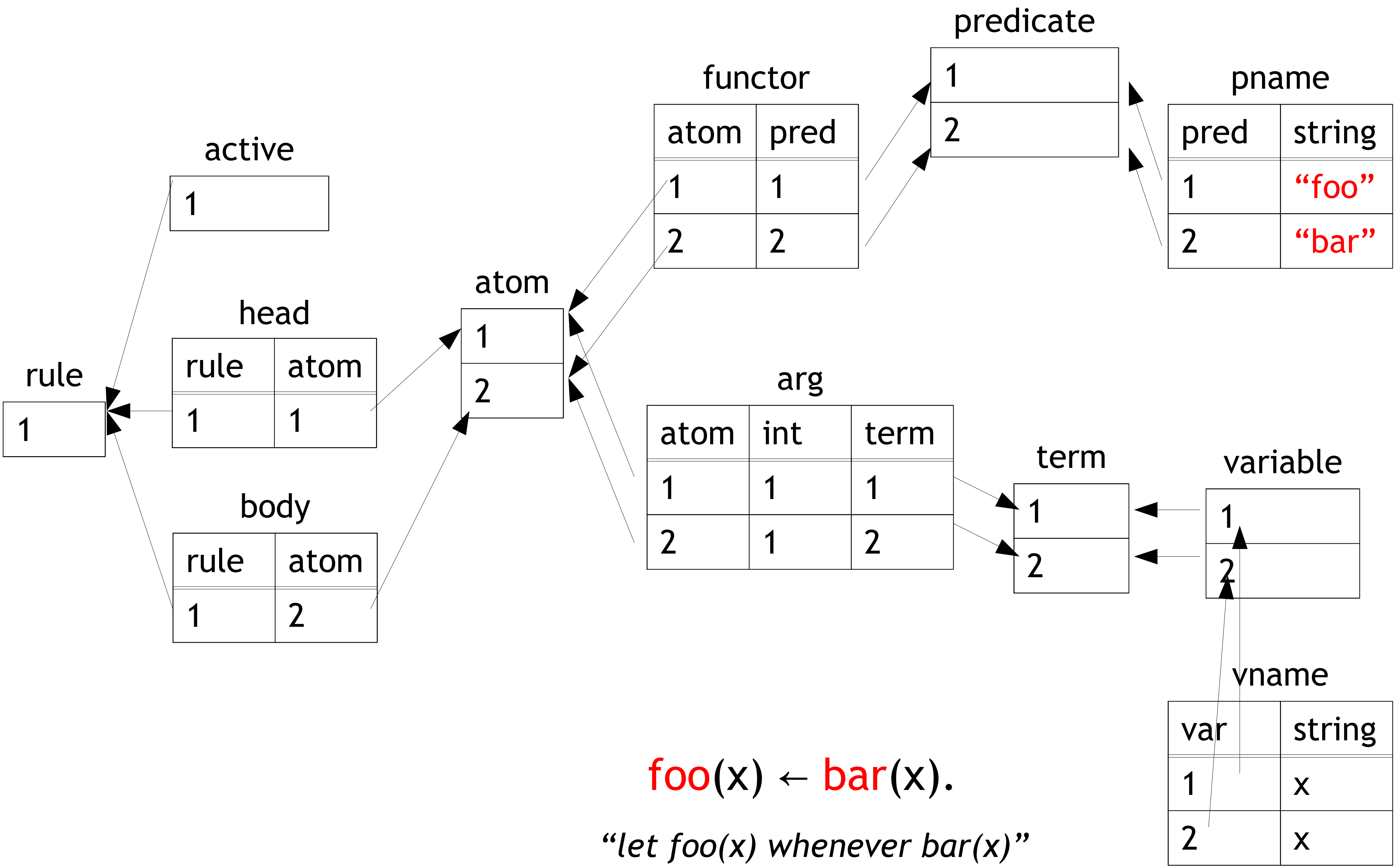
type constraint

Meta-Model Schema

```
rule(R) → .
active(R) → rule(R).
head(R,A) → rule(R), atom(A).
body(R,A) → rule(R), atom(A).

atom(A) → .
functor(A,P) → atom(A), predicate(P).
arg(A,I,T) → atom(A), int(I), term(T).
negated(A) → atom(A).

term(T) → .
variable(X) → term(X).
vname(X.N) → variable(X), string(N).
constant(C) → term(C).
value(C,V) → constant(C), string(V).

predicate(P) → .
pname(P,N) → predicate(P), string(N).
```

ensures rules are well-structured

Rules as Data



foo(x) ← bar(x).

"let foo(x) whenever bar(x)"

Meta Rules for Security

- Meta
  - Code generation (insert new rules that must be evaluated)
  - Reflection (query for program structure)
- Meta-Syntax
  - Embedded rule/bounded constants

active([| active(R) ← says(~P2,~P1,R). |]) ← delegates(P1,P2).

"activate a rule active(R) ← says(P2,P1,R). for every delegates(P1,P2)."

Meta-Constraints

- Meta
  - Code generation (insert new rules that must be evaluated)
  - Reflection (query for program structure)

owner(U, [| A <- P(T*), A*. |]) → access(U,P,read).

"whenever user U owns a rule, require that U has read access to every predicate P in the rule body"

fail() ← owner(U, [| A <- ~P(T*), A*. |]), !access(U,P,read).

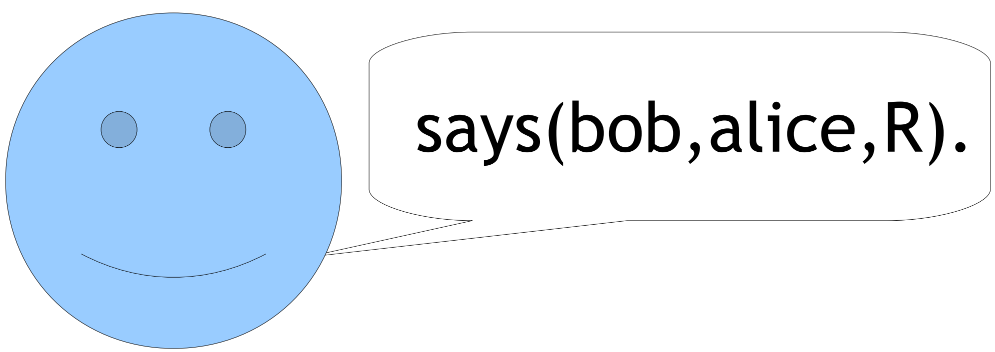
meta variables

Kleene star

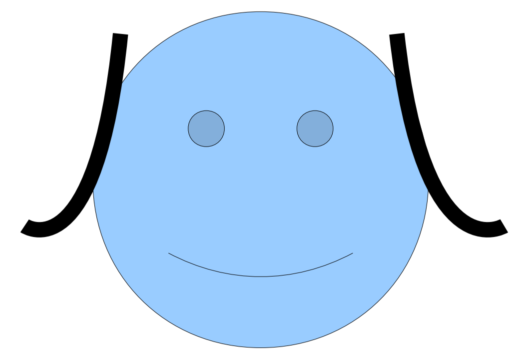
A Concrete Example: The "Says" Authentication Construct

says(P1,P2,R) → prin(P1), prin(P2), rule(R).
rulesig(R,S) → rule(R), string(S).
rsapubkey(P,K) → prin(P), string(K).
rsaprivkey(P,K) → prin(P), string(K).

schema / type constraints



bob



alice

r1: rulesig(R,S) ←
says(P1,_,R),
rsaprivkey(P1,K),
rsasign(R,S,K).

signature derivation

signature check constraint

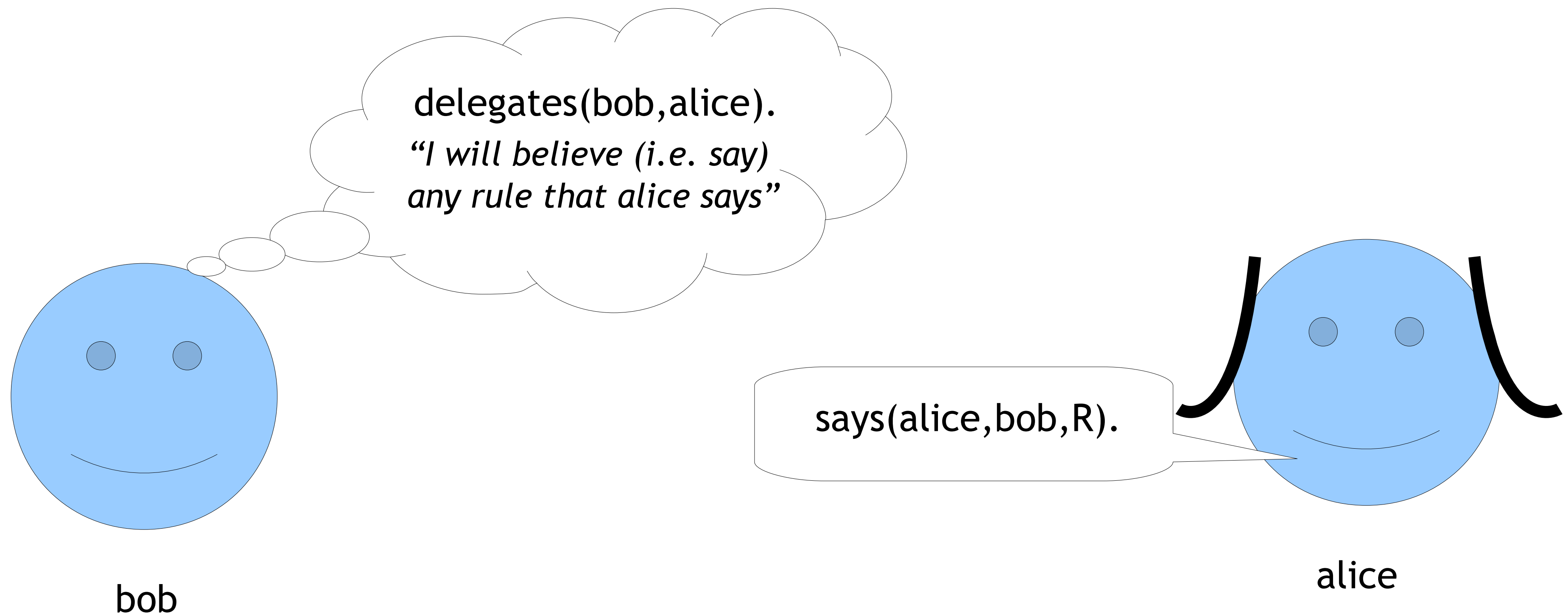
r2: says(P1,_,R),
rsapubkey(P1,K),
rulesig(R,S) →
rsaverify(R,S,K).

Delegation (Basic)

alice "*speaks-for*" bob == "if alice says something, bob says it too."

speaks-for is a special form of delegation:

- delegates(P1,P2) → prin(P1), prin(p2).



r1: active([| active(R) ← says(P2,P1,R). |]) ← delegates(P1,P2).

r2: active(R) ← says(alice,bob,R).

Other cool features (see paper for details)

- Conditional Delegations:
  - Constraint by width, depth, or predicates
  - Detecting delegation violations (use of provenance)
- Customizable distribution/partitioning policies
  - Partition data and rules by principals
  - Distribute principals across machines
  - Same security policy rules can run in local/distributed environment
- Customizable authentication and encryption (RSA vs HMAC)
- Use meta-rules to rewrite top-down access control to execute in a bottom-up evaluation engine
- Example languages:
  - Binder
  - Delegation logic, D1LP
  - Secure Network Datalog [ICDE 09]
    - Authenticated routing protocols

LogicBlox - a commercial Datalog Engine

- Startup company based in Atlanta (50 employees + 65 academic collaborators)
- Decision Automation Applications:
  - Retail supply-chain management (Predictix) – e.g: Best Buy, Sainsbury,
  - Insurance risk management (Verabridge) – e.g. RenRe
  - Context Sensitive Program Analysis (Semmle) - TBD
- LBTrust is developed using LogicBlox:
  - Classic datalog with well behaved constructors or E variables in head
  - Constraints
  - Meta-programmability: model, rules, constraints
  - Higher-Order: gets us aggs, state + ECA, default values, etc.