Semantics of Datalog for the Evidential Tool Bus[1]

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

May 8, 2018

[1] Joint work with Simon Cruanes (Ecole Polytechnique), Stijn Heymans (SRI AIC), Ian Mason and Sam Owre (SRI CSL).

Motivation

- An assurance case is *"a documented body of evidence that provides a convincing and valid argument that a specified set of critical claims about a system's properties are adequately justified for a given application in a given environment."* [Adelard]
- From the FDA Draft Guidance document *Total Product Life Cycle: Infusion Pump - Premarket Notification [510(k)] Submissions*:

  An assurance case is a formal method for demonstrating the validity of a claim by providing a convincing argument together with supporting evidence. It is a way to structure arguments to help ensure that top-level claims are credible and supported. In an assurance case, many arguments, with their supporting evidence, may be grouped under one top- level claim. For a complex case, there may be a complex web of arguments and sub-claims.

Talk Outline

- Motivation and design of an Evidential Tool Bus (ETB) for building assurance arguments.
- Datalog as a metalanguage for defining workflows and building arguments.
- Peculiarities of ETB Datalog
- Semantics of ETB Datalog (omitted in the talk)
- Abstract machine for the tabled evaluation of ETB Datalog queries in a distributed setting
- Termination check for detecting fully evaluated goals
- Conclusions

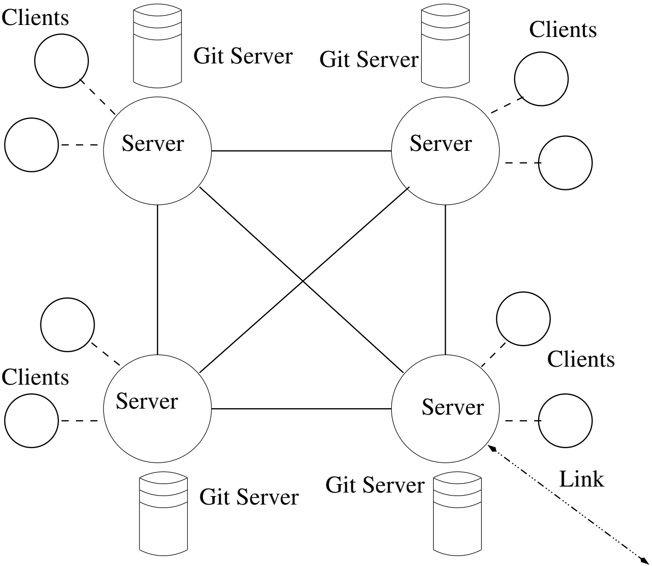
ETB Overview

- The Evidential Tool Bus (ETB) is a distributed tool integration framework for constructing and maintaining claims supported by arguments based on evidence.
- ETB provides the infrastructure for
  - Creating workflows that integrate multiple tools, e.g., static analyzers, dynamic analyzers, satisfiability solvers, model checkers, and theorem provers
  - Generating claims based on these workflows
  - Producing checkable evidence (e.g., files) supporting these claims
  - Maintaining the evidence against changes to the inputs
- ETB (`https://github.com/SRI-CSL/ETB`) is implemented in Python 2.7 (but still somewhat buggy!).
- This talk is preparation for a PVS formalization and code generation for a new implementation (integrating Cyberlogic, a logic of attestations).

ETB Desiderata

- ETB targets the production of claims supported by arguments in which some of the sub-claims can be established by external tools.
- The three key design requirements for ETB are
  - Extensibility:
    - New claim forms and rules of argumentation
    - New external tools (including human oracles)
    - New workflows that are defined by scripts
    - New clients and servers
  - Assurance:
    - Explication of tools, artifacts, rules, and assumptions on which the argument depends
    - Replay, revision, and rechecking of arguments
  - Semantic Neutrality: ETB makes no commitments to specific tools, languages, or models
- ETB is infrastructure for building and checking arguments, and can be used to implement specific assurance methodologies.

ETB Architecture



Datalog

- Datalog is a fragment of Horn-clause logic programming first introduced in the 1970s as a database query language.
- It was realized that first-order logic could not represent recursive queries like transitive closure:

```
ancestor(x, y) :- parent(x, y)
ancestor(x, y) :- parent(x, z), ancestor(z, y)
```

- Much of the research focused on evaluation strategies for such queries, e.g., semi-naïve, magic sets, tabled evaluation.
- In the last decade, Datalog has come to be seen as a versatile tool for many applications: data integration, provenance, declarative networking, synchronous programming, runtime monitors, program analysis, among others.

Datalog as a Metalanguage

- *Atoms* are of the form $p(a_1, \ldots, a_n)$, where p is a *predicate* and each a_i is either a *data object* or a variable, e.g.,
  - $models(Model, Formula)$
  - $cnf(Formula, CNFFormula)$
- Data objects can be JSON terms, file handles (with SHA-1 hash), tool handles (e.g., BDDs), session handles.
- A *predicate* p can either be
  - *Interpreted* by means of a tool invocation through wrappers, e.g., `yices`.
  - *Uninterpreted*, i.e., defined by a Datalog program that is evaluated locally, e.g., `allsat`.
- An uninterpreted predicate is defined by clauses of the form $p(a_1, \ldots, a_n) :- Q$, where Q is a list of atoms whose free variables contain those of $p(a_1, \ldots, a_n)$.
- A *query* is an atom (possibly) with free variables, e.g., $cnf(formula, CNFFormula)$.
- A *claim* is a *ground* atom that is supported by a proof.

An Example ETB Workflow: AllSAT

The defined predicates `sat` and `unsat` invoke the interpreted `yices` predicate on the given file `F`.

```
sat(F, M) :- yices(F, S, M),
             equal(S, 'sat').
unsat(F) :- yices(F, S, M),
            equal(S, 'unsat').

allsat(F, Answers) :- sat(F, M),
                      negateModel(F, M, NewF),
                      allsat(NewF, T),
                      cons(M, T, Answers).
allsat(F, Answers) :- unsat(F),
                      nil(Answers).
```

Though `allsat` calls `sat` and `unsat`, the `yices` wrapper is only executed once on the file `F` since the resulting claim is tabled.

A Variant: AllSAT with a Yices Session

```
allsat(F, Answers) :- yicesStart(Session0),
                      yicesIncludeFile(Session0, F, Session1),
                      allsat_enum(Session1, Answers).

allsat_enum(Session, Answers) :-
      yicesCheck(Session, Session1, Result),
      allsat_iter(Session1, Result, Answers).

allsat_iter(Session, Result, Answers) :-
      equal(Result, 'sat'),
      yicesModel(Session, Model),
      yicesAssertNegation(Session, Model, Session1),
      allsat_enum(Session1, Answers1),
      cons(Model, Answers1, Answers).

allsat_iter(Session, Result, Answers) :-
      equal(Result, 'unsat'),
      yicesClose(Session),
      nil(Answers).
```

ETB Datalog versus Datalog

- Datalog as a database query language has intensional and extensional predicates — ETB has uninterpreted and interpreted predicates.
- Interpreted predicates are similar to built-ins (which evaluate ground atoms), but more general.
- ETB only admits top-down left-to-right evaluation — no bottom-up evaluation.
- In ETB, Datalog is the metalanguage — sparse data, but elaborate workflows.
- In Datalog, the Herbrand universe is bounded, but in ETB Datalog, it is unbounded.
- ETB Datalog evaluation is distributed — uninterpreted predicates are evaluated locally, and interpreted predicates might be evaluated remotely.
- No (stratified) negation — we only establish positive claims.

What is an Interpreted Predicate?

- Datalog has been extended with built-in predicates, but these are usually evaluated when all arguments are grounded, e.g., $<(x, y)$.
- An interpreted predicate $p(a_1, \ldots, a_n)$ is evaluated by a wrapper.
- The evaluation of a predicate $p(a_1, \ldots, a_n)$ generates clauses (lemmata) of the form

$$\begin{array}{rcl} p(b_{11}, \ldots, b_{1n}) & :- & Q_1 \\ & \vdots & \\ p(b_{m1}, \ldots, b_{mn}) & :- & Q_m \end{array}$$

- For example, the evaluation of `veryComposite`$(8, 3)$, can generate

$$\begin{array}{rcl} veryComposite(8,3) & :- & composite(8), \\ & & composite(9), \\ & & composite(10). \end{array}$$

The ETB Datalog Abstract Inference System

- We can define a sound/complete inference system for query processing.
- A logical state consists of a pair $G; J$ with a set of (normalized) goals G and a set of normalized clauses J.
- Each goal in G is of the form $\neg A$ and each clause in J is of the form $[B :- Q]$.
- Initially, J is empty and G is the singleton $\{\neg A\}$.
- The inference system consists of inference rules that transform the logical state.

- **Backchain:**

$$\frac{G; \overbrace{[B :- A_1, \ldots, A_n], J'}^{J}}{G, \neg A_1; [B :- A_1, \ldots, A_n], J'}$$

- **Resolve:**

$$\frac{\overbrace{\neg A, G'}^{G}; J}{G; \sigma([B :- Q]), J} \; \sigma = mgu(A, B), [B :- Q] \in R$$

- **External:**

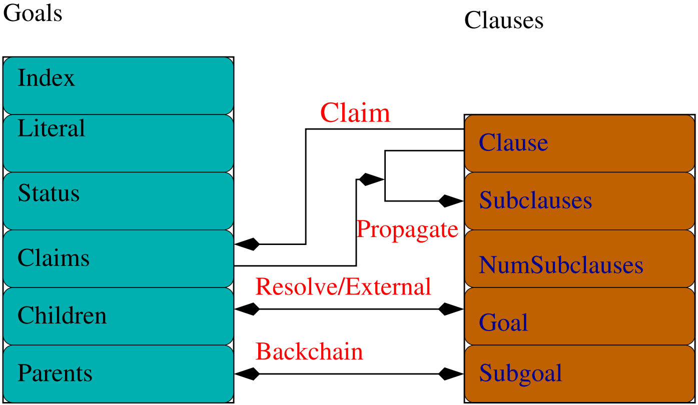
$$\frac{\neg A, G'; J}{G; J \cup E(A)} \; \text{external atom } A$$

- **Propagate:**

$$\frac{G; [B :- A, Q], A', J'}{G; \sigma([B :- Q]), J} \; \sigma(A) = A'$$

ETB Abstract Machine

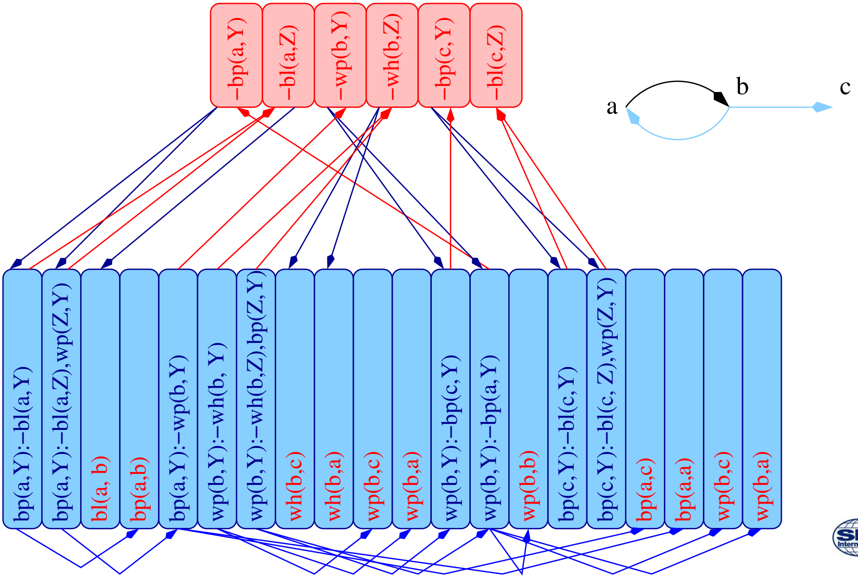
- The abstract inference system simplifies the correctness argument, but it can be inefficient without a *strategy* and *indexing*.
- We define an abstract machine for tabled evaluation whose state consists of Goals and Clauses.



A Datalog Example

C_1	$black(a, b)$		
C_2	$white(b, c)$		
C_3	$white(b, a)$		
C_4	$blackpath(X, Y)$	$:-$	$black(X, Y)$
C_5	$blackpath(X, Y)$	$:-$	$black(X, Z), whitepath(Z, Y)$
C_6	$whitepath(X, Y)$	$:-$	$white(X, Y)$
C_7	$whitepath(X, Y)$	$:-$	$white(X, Z), blackpath(Z, Y)$

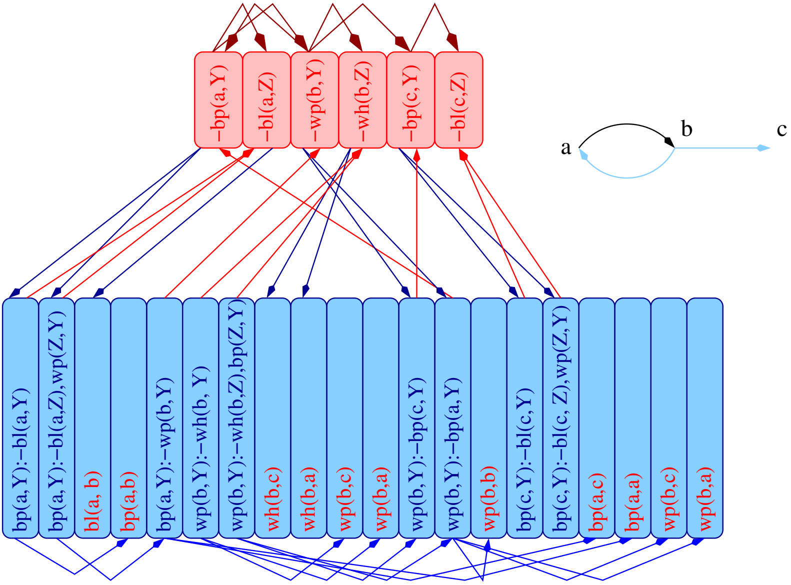
Example Derivation



Terminating Evaluation

- How do we know when a goal or subgoal has been fully evaluated?
- Evaluation state is shared by a number of queries.
- We add inference rules for termination checking that can be interleaved with normal evaluation.
- These rules track the dependencies between goals counting claims propagated from subgoals to goals.
- The algorithm exploits the order between goals to close off goals g in which all transitive subgoals h are such that $h < g$ or $h = g$ and all claims have been propagated, or h is closed.
- When a node is closed and has no prior unclosed subgoals, it is marked as *Complete*.
- The *Complete* marking is propagated to any closed subgoals.

Termination in our Example



Conclusions

- ETB is a distributed framework for tools, tool chains, workflows, and evidence.
- It is based on a simple architecture with
  - Datalog as a metalanguage
  - A well-defined denotational and operational semantics
  - Interpreted predicates for tool invocation, and uninterpreted predicates for other claims
  - Data in JSON format
  - Datalog inference trees as proofs
  - Git as a medium for file identity and version control
- The semantics and abstract machine has guided the implementation
- ETB can also be used for non-formal applications in distributed computing.

2018 Summer School on Formal Techniques

- The Eighth Summer School on Formal Techniques will take place during May 19 - May 25, 2018, at Menlo College, Atherton, CA. See `http://fm.csl.sri.com/SSFT18` for details.
- The lecturers at the school include:
  1. Emina Torlak (U. Washington): Solver-Aided Programming
  2. Nikhil Swami & Jonathan Protzenko (MSR): Programming and Proving in F* and Low*
  3. Andreas Abel (U. Göteborg): Introduction to Dependent Types and Agda
  4. Dirk Beyer (LMU Munich): Configurable Software Model Checking — A Unifying View
  5. Mooly Sagiv (Tel Aviv): Modularity for Decidability: Implementing and Semi-Automatically Verifying Distributed Systems
- Invited speakers include Gordon Plotkin (Edinburgh), Nina Narodytska (VMWare Research), Edward A. Lee (Berkeley).