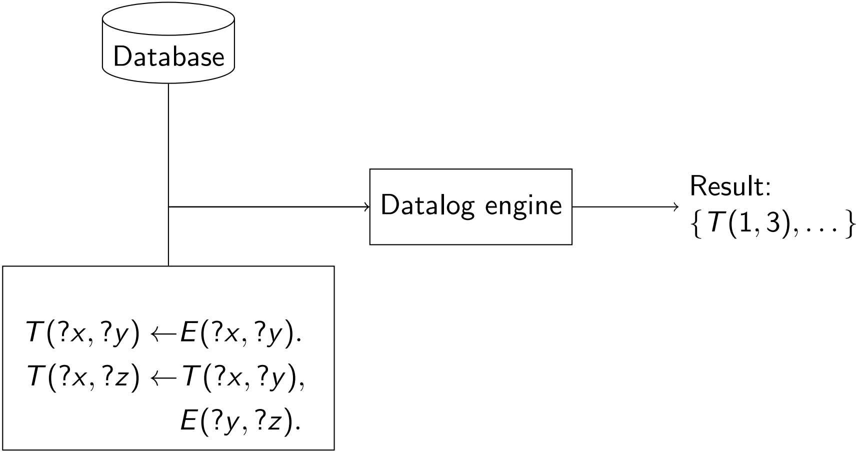
LEAN-Aided Certification of Datalog Reasoning Results

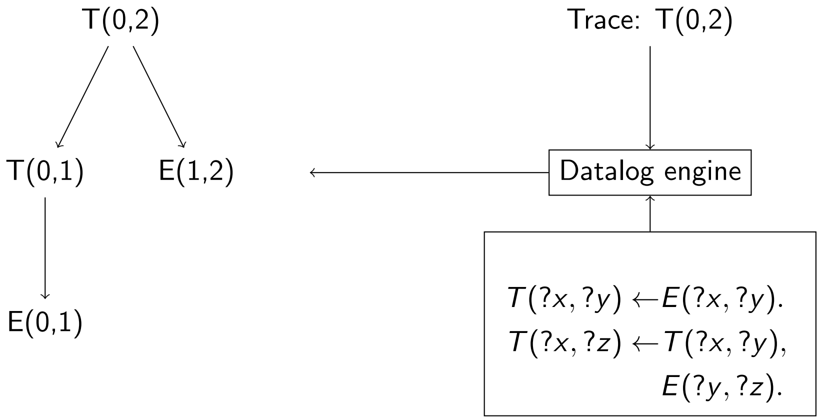
Johannes Tantow

05.06.2024

Introduction



Tracing


T(0,2)
T(0,1)
E(1,2)
E(0,1)
Trace: T(0,2)
Datalog engine
$T(?x, ?y) \leftarrow E(?x, ?y).$
$T(?x, ?z) \leftarrow T(?x, ?y),$
$E(?y, ?z).$

Lean

Playground.lean 2, U

Playground.lean > myNat.add_zero

```
  deriving DecidableEq, Inhabited

inductive evenNat: Type
| zero: evenNat
| evenSucc: evenNat → evenNat

def evenNatToMyNat (e: evenNat): myNat :=
  match e with
  | evenNat.zero => myNat.zero
  | evenNat.evenSucc e' => myNat.succ (myNat.succ (evenNatToMyNat e'))


instance coeEvenNatMyNat: Coe evenNat myNat := (evenNatToMyNat)

def myNat.add (n m: myNat): myNat :=
  match m with
  | zero => n
  | succ m' => succ (add n m')

theorem myNat.add_zero (n: myNat): myNat.add zero n = n := by
  induction n with
  | zero =>
    unfold add
    rfl
  | succ n' ih =>
    unfold add
    rw [ih]

def myNat.mod (n m: myNat): myNat := sorry



def myNat.Euclid (a b: myNat): myNat :=
  if b = zero
  then a
```

Lean Infoview

▼ Playground.lean:32:4

▼ Tactic state

1 goal

▼ case succ

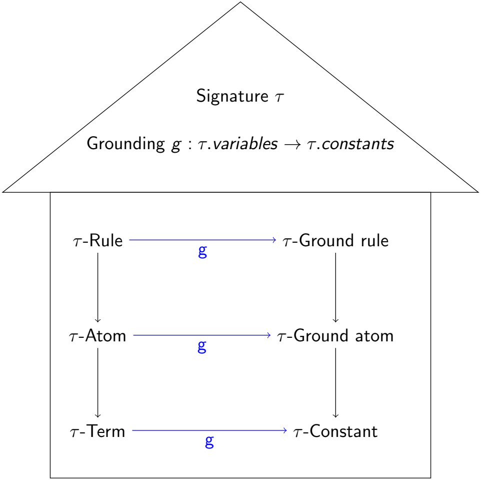
```
n' : myNat
ih : add zero n' = n'
⊢ succ (add zero n') = succ n'
```

▶ All Messages (2)

Outline

1. Formalize datalog
2. Soundness: Verify datalog proofs
3. Completeness: Can more be derived?
4. Evaluation

Syntax


Signature τ
Grounding $g : \tau.\mathit{variables} \to \tau.\mathit{constants}$
τ-Rule
g
τ-Ground rule
τ-Atom
g
τ-Ground atom
τ-Term
g
τ-Constant

Model-theoretic semantics

Definition (12.2.1, Alice Book)

Let P be a datalog program. [...] The semantics of P on input I, denoted $P(I)$, is the minimum model of P containing I, if it exists.

Model-theoretic semantics

Definition (12.2.1, Alice Book)

Let P be a datalog program. [...] The semantics of P on input I, denoted $P(I)$, is the minimum model of P containing I, if it exists.

$$M = \bigcap_{X \text{ is model of } P} X$$

Model-theoretic semantics

Definition (12.2.1, Alice Book)

Let P be a datalog program. [...] The semantics of P on input I, denoted $P(I)$, is the minimum model of P containing I, if it exists.

$$M = \bigcap_{X \text{ is model of } P} X$$

```
def iInter (s : ι → Set α) : Set α
```

Model-theoretic semantics

Definition (12.2.1, Alice Book)

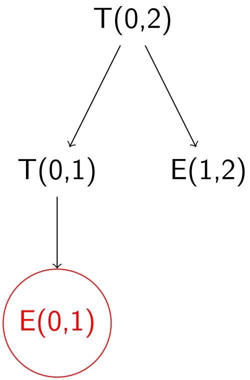
Let P be a datalog program. [...] The semantics of P on input I, denoted $P(I)$, is the minimum model of P containing I, if it exists.

$$M = \bigcap_{X \text{ is model of } P} X$$

```
def iInter (s : ι → Set α) : Set α
```

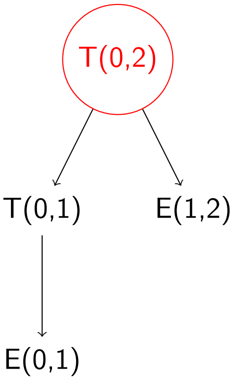
$$M = \{a \in \text{ groundAtom } \tau \mid \forall i, \text{isModel i P} \rightarrow a \in i\}$$

Proof-theoretic semantics



Database contains $E(0, 1)$.

Proof-theoretic semantics



$\exists r, g$:

1. $r \in P$
2. $T(0,2) \leftarrow T(0,1), E(1,2) = apply(r,g)$
3. All subtrees are valid

Unification

Signature τ

Grounding $g : \tau.\mathit{variables} \to \tau.\mathit{constants}$

$$\begin{tikzcd}
\tau\text{-Rule} \arrow[r, "g"'] \arrow[d] & \tau\text{-Ground rule} \arrow[d] \\
\text{List } (\tau\text{-Atom}) \arrow[r, "g"'] \arrow[d] & \text{List } (\tau\text{-Ground Atom}) \arrow[d] \\
\tau\text{-Atom} \arrow[r, "g"'] \arrow[d] & \tau\text{-Ground atom} \arrow[d] \\
\text{List } (\tau\text{-Term}) \arrow[r, "g"'] \arrow[d] & \text{List } (\tau\text{-Constant}) \arrow[d] \\
\tau\text{-Term} \arrow[r, "g"'] & \tau\text{-Constant}
\end{tikzcd}$$

Substitutions

Terms	$?x$	$?y$
Ground terms	a	b
Groundings	$?x \mapsto a, ?y \mapsto a$	Error

Substitutions

Terms	$?x$	$?y$
Ground terms	a	b
Groundings	$?x \mapsto a, ?y \mapsto a$	Error
Substitutions	$?x \mapsto a$	$?x \mapsto a, ?y \mapsto b$

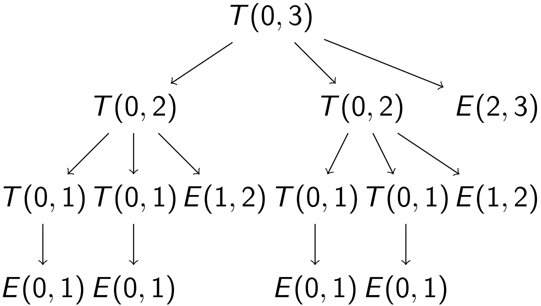
```
def substitution (τ: signature):= τ.vars → Option
    (τ.constants)
```

Proof graphs

$$T(?x, ?y) \leftarrow E(?x, ?y).$$
$$T(?x, ?z) \leftarrow T(?x, ?y), T(?x, ?y), E(?y, ?z).$$

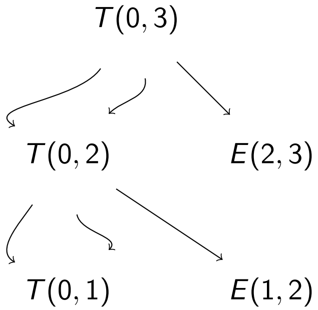
Proof graphs

$$T(?x,?y) \leftarrow E(?x,?y).$$
$$T(?x,?z) \leftarrow T(?x,?y), T(?x,?y), E(?y,?z).$$



Proof graphs

$$T(?x, ?y) \leftarrow E(?x, ?y).$$
$$T(?x, ?z) \leftarrow T(?x, ?y), T(?x, ?y), E(?y, ?z).$$



Graphs

$$T(0,3)$$

$$T(0,2) \qquad E(2,3)$$

$$T(0,1) \qquad E(1,2)$$

```
structure SimpleGraph (V : Type u) where
  Adj : V → V → Prop
  symm : Symmetric Adj
```

Graphs

$$T(0,3)$$

$$T(0,2) \qquad E(2,3)$$

$$T(0,1) \qquad E(1,2)$$

```
structure Graph (V : Type u) where
  vertices : List V
  successors: V → List V
```

Graphs

$$\begin{tikzcd}
 & & T(0,3) \arrow[dl] \arrow[dr] & \\
 & T(0,2) \arrow[dl] \arrow[dr] & & E(2,3) \\
T(0,1) & & E(1,2) &
\end{tikzcd}$$

```
def Graph (V: Type) := Std.HashMap V (List V)
```

Acyclity criteria

Dfs G = true $\Leftrightarrow$ G is acyclic
DfsStep G a = true $\Leftrightarrow$ ...?

$$
\begin{tikzcd}
D \arrow[r] & C \arrow[r] & B \arrow[r] & A \arrow[ll, bend right] \\
E \arrow[r] & F & &
\end{tikzcd}
$$

Acyclity criteria

Dfs G = true $\Leftrightarrow$ G is acyclic
DfsStep G a = true $\Leftrightarrow$ . . . ?

$$\begin{tikzcd} D \arrow[r] & C \arrow[r] & B \arrow[r] & A \arrow[ll, bend right] \end{tikzcd}$$

D is not in a cycle.
A,B,C are all in a cycle.

Acyclity criteria

Dfs G = true $\Leftrightarrow$ G is acyclic

DfsStep G a = true $\Leftrightarrow$ ...?

$$\begin{tikzcd} D \arrow[r] & C \arrow[r] & B \arrow[r] & A \arrow[ll, bend right] \end{tikzcd}$$

D is not in a cycle.
D can reach a cycle.

A,B,C are all in a cycle.
A,B,C all can reach a cycle.

Completeness

V: atoms occurring in the proof graph
M: solution

Proof graph valid: $V \subseteq M$

Completeness

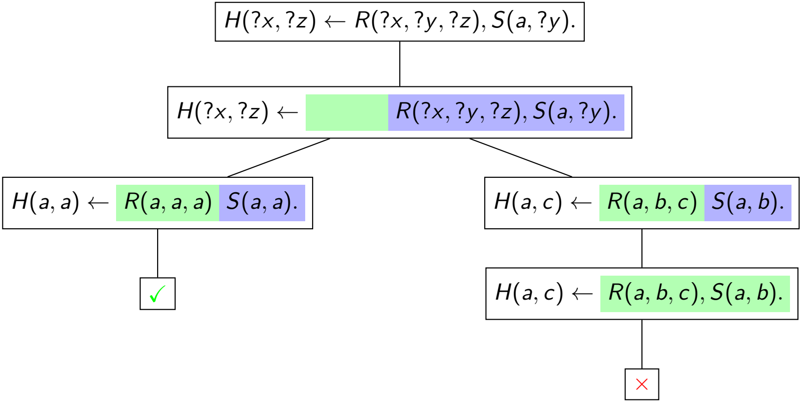
V: atoms occurring in the proof graph
M: solution

Proof graph valid: $V \subseteq M$
V is a model: $M \subseteq V$

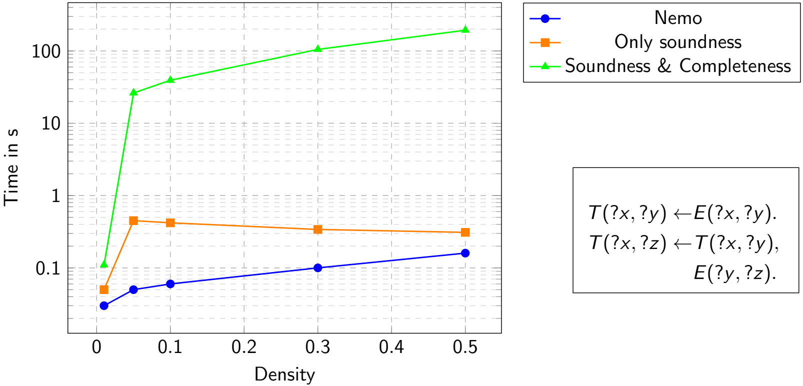
Hence: $V = M$

Partial ground rules

$$I = \{R(a,a,a), R(a,b,c), S(a,b)\}$$

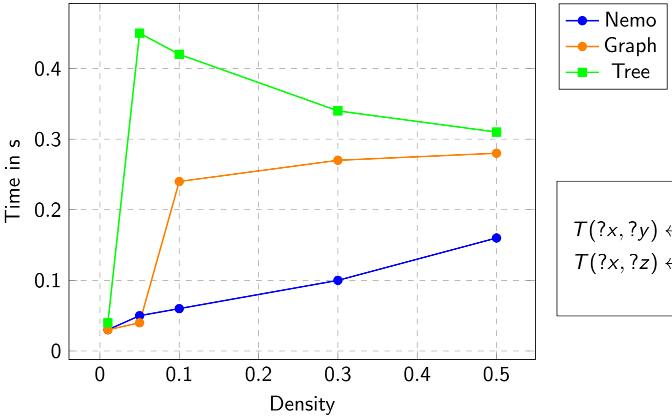


Soundness & Completeness



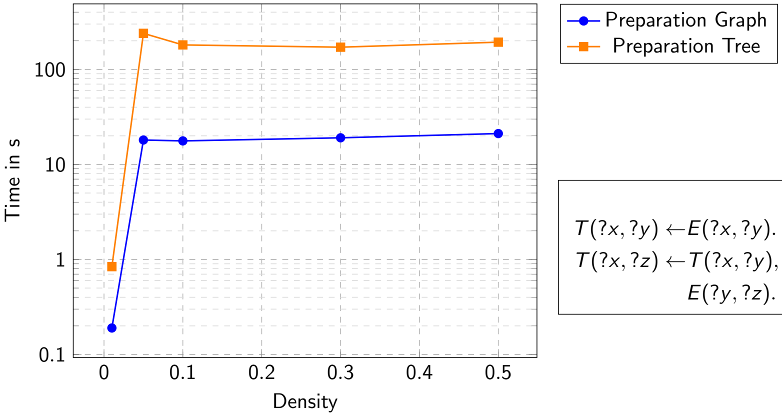
$$
\begin{aligned}
T(?x,?y) &\leftarrow E(?x,?y). \\
T(?x,?z) &\leftarrow T(?x,?y), \\
&\qquad E(?y,?z).
\end{aligned}
$$

Graphs vs. trees



$$T(?x, ?y) \leftarrow E(?x, ?y).$$
$$T(?x, ?z) \leftarrow T(?x, ?y), E(?y, ?z).$$

Graphs vs. trees



$$T(?x, ?y) \leftarrow E(?x, ?y).$$
$$T(?x, ?z) \leftarrow T(?x, ?y), E(?y, ?z).$$

Results

```
lemma checkValidnessIffAllTreesAreValid :
checkValidnessMockDatabase problem = Except.ok () ↔
∀ (t: proofTree τ), t ∈ problem.trees → isValid
    program db t
```

In total: 6k lines of code

Required to verify: 50 lines of code for definitions & axioms

Lessons learned

1. Proving showed bugs during the development
2. Proof assistants useful for changing definitions
3. More automation in Lean would have helped
4. Definitions sometimes have to be redeveloped for Lean