Technische Universität Dresden

Alex Ivliev Markus Krötzsch Maximilian Marx
Knowledge-Based Systems Group · TU Dresden

Recovering Explanations from Transformed Rule-Based Ontologies

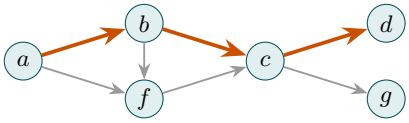
The 25th International Semantic Web Conference

October 28th 2026

Datalog and Proof Trees

Datalog is explainable: proof trees trace every fact back to the input.

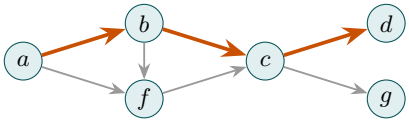
$$r(x, y) \leftarrow e(x, y) \quad \textbf{(B1)}$$
$$r(x, y) \leftarrow r(x, z),\ r(z, y) \quad \textbf{(B2)}$$

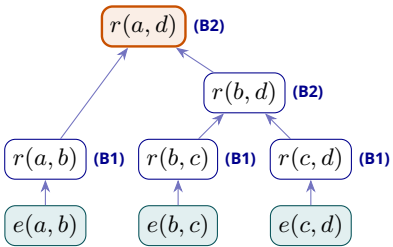


Datalog and Proof Trees

Datalog is explainable: proof trees trace every fact back to the input.

$$r(x,y) \leftarrow e(x,y) \quad \textbf{(B1)}$$
$$r(x,y) \leftarrow r(x,z),\, r(z,y) \quad \textbf{(B2)}$$



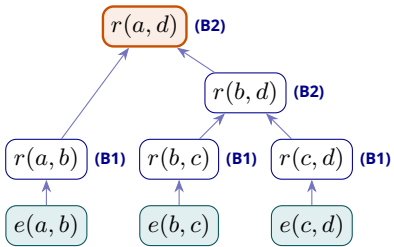


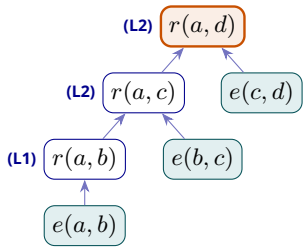
Datalog and Proof Trees

Datalog is explainable: proof trees trace every fact back to the input.

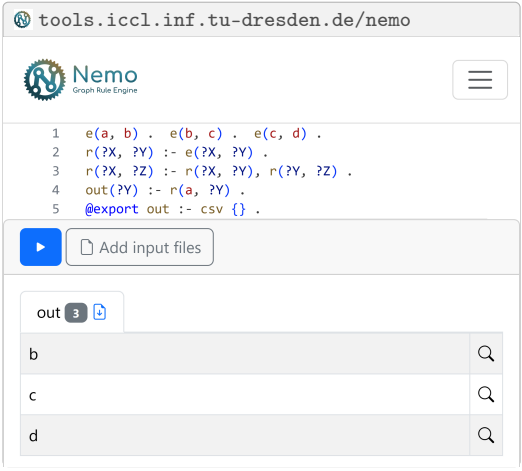
$$r(x,y) \leftarrow e(x,y) \quad \textbf{(B1)}$$
$$r(x,y) \leftarrow r(x,z),\, r(z,y) \quad \textbf{(B2)}$$

$$r(x,y) \leftarrow e(x,y) \quad \textbf{(L1)}$$
$$r(x,y) \leftarrow r(x,z),\, e(z,y) \quad \textbf{(L2)}$$

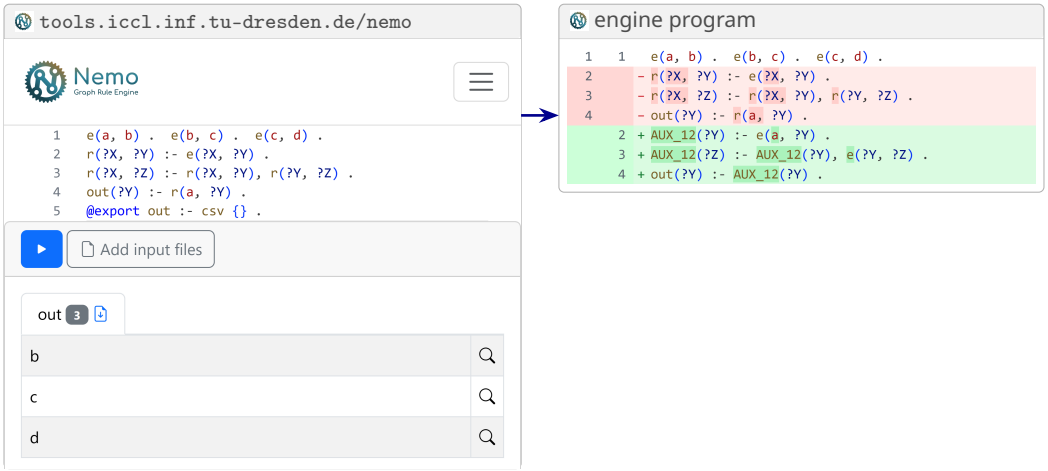




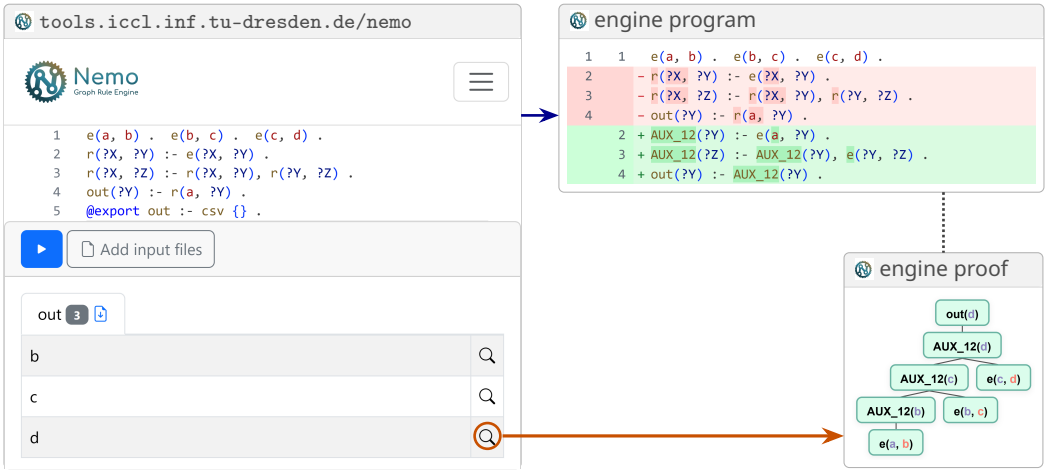
The Problem


tools.iccl.inf.tu-dresden.de/nemo
Nemo
Graph Rule Engine
1 e(a, b) . e(b, c) . e(c, d) .
2 r(?X, ?Y) :- e(?X, ?Y) .
3 r(?X, ?Z) :- r(?X, ?Y), r(?Y, ?Z) .
4 out(?Y) :- r(a, ?Y) .
5 @export out :- csv {} .
Add input files
out 3
b
c
d

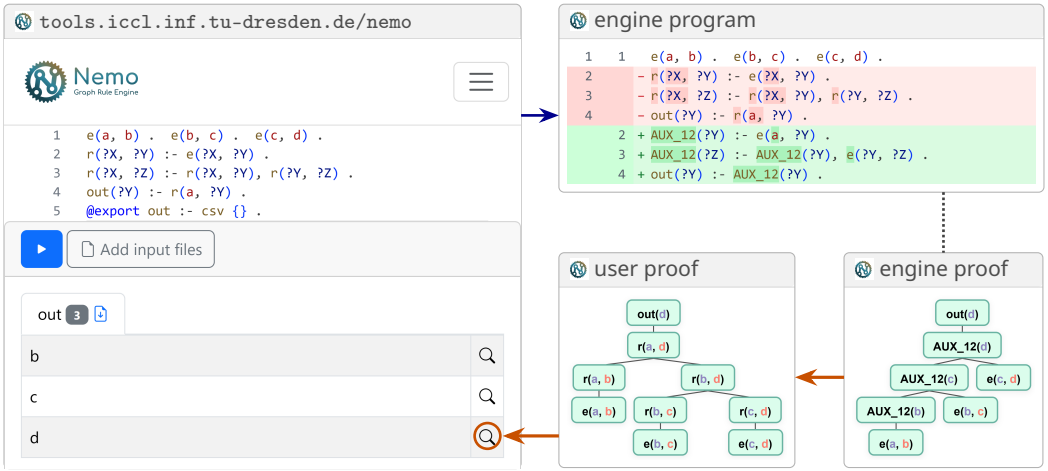
The Problem


tools.iccl.inf.tu-dresden.de/nemo
Nemo
Graph Rule Engine
1 e(a, b) . e(b, c) . e(c, d) .
2 r(?X, ?Y) :- e(?X, ?Y) .
3 r(?X, ?Z) :- r(?X, ?Y), r(?Y, ?Z) .
4 out(?Y) :- r(a, ?Y) .
5 @export out :- csv {} .
Add input files
out 3
b
c
d
engine program
1 1 e(a, b) . e(b, c) . e(c, d) .
2 - r(?X, ?Y) :- e(?X, ?Y) .
3 - r(?X, ?Z) :- r(?X, ?Y), r(?Y, ?Z) .
4 - out(?Y) :- r(a, ?Y) .
2 + AUX_12(?Y) :- e(a, ?Y) .
3 + AUX_12(?Z) :- AUX_12(?Y), e(?Y, ?Z) .
4 + out(?Y) :- AUX_12(?Y) .

The Problem


tools.iccl.inf.tu-dresden.de/nemo
Nemo
Graph Rule Engine
1 e(a, b) . e(b, c) . e(c, d) .
2 r(?X, ?Y) :- e(?X, ?Y) .
3 r(?X, ?Z) :- r(?X, ?Y), r(?Y, ?Z) .
4 out(?Y) :- r(a, ?Y) .
5 @export out :- csv {} .
Add input files
out 3
b
c
d
engine program
1 1 e(a, b) . e(b, c) . e(c, d) .
2 - r(?X, ?Y) :- e(?X, ?Y) .
3 - r(?X, ?Z) :- r(?X, ?Y), r(?Y, ?Z) .
4 - out(?Y) :- r(a, ?Y) .
2 + AUX_12(?Y) :- e(a, ?Y) .
3 + AUX_12(?Z) :- AUX_12(?Y), e(?Y, ?Z) .
4 + out(?Y) :- AUX_12(?Y) .
engine proof
out(d)
AUX_12(d)
AUX_12(c)
e(c, d)
AUX_12(b)
e(b, c)
e(a, b)

The Problem


tools.iccl.inf.tu-dresden.de/nemo
Nemo
Graph Rule Engine
1 e(a, b) . e(b, c) . e(c, d) .
2 r(?X, ?Y) :- e(?X, ?Y) .
3 r(?X, ?Z) :- r(?X, ?Y), r(?Y, ?Z) .
4 out(?Y) :- r(a, ?Y) .
5 @export out :- csv {} .
Add input files
out 3
b
c
d
engine program
1 1 e(a, b) . e(b, c) . e(c, d) .
2 - r(?X, ?Y) :- e(?X, ?Y) .
3 - r(?X, ?Z) :- r(?X, ?Y), r(?Y, ?Z) .
4 - out(?Y) :- r(a, ?Y) .
2 + AUX_12(?Y) :- e(a, ?Y) .
3 + AUX_12(?Z) :- AUX_12(?Y), e(?Y, ?Z) .
4 + out(?Y) :- AUX_12(?Y) .
user proof
out(d)
r(a, d)
r(a, b)
r(b, d)
e(a, b)
r(b, c)
r(c, d)
e(b, c)
e(c, d)
engine proof
out(d)
AUX_12(d)
AUX_12(c)
e(c, d)
AUX_12(b)
e(b, c)
e(a, b)

Complexity of Proof Tree Recovery

TRANS$_{\text{enc}}(\Pi_1, \Pi_2)$ for programs $\Pi_1 \sqsubseteq \Pi_2$ and enc $\in \{\text{tree}, \text{dag}\}$

Input: a proof tree (enc $=$ tree) or proof DAG (enc $=$ dag) T_1 for f over Π_1

Output: a proof DAG T_2 for f over Π_2

Complexity of Proof Tree Recovery

$\textsc{Trans}_{\text{enc}}(\Pi_1, \Pi_2)$ for programs $\Pi_1 \sqsubseteq \Pi_2$ and $\text{enc} \in \{\text{tree}, \text{dag}\}$

Input: a proof tree (enc $=$ tree) or proof DAG (enc $=$ dag) T_1 for f over Π_1

Output: a proof DAG T_2 for f over Π_2

Upper bound. Recompute the answer:

1. collect the leaves of T_1 and re-evaluate Π_2 on them
   Datalog evaluation is in PTIME (data complexity)
2. record for every derived fact the rule producing it
3. trace back from the root fact

Complexity of Proof Tree Recovery

TRANS$_{\text{enc}}(\Pi_1, \Pi_2)$ for programs $\Pi_1 \sqsubseteq \Pi_2$ and enc $\in \{\text{tree}, \text{dag}\}$

Input: a proof tree (enc $=$ tree) or proof DAG (enc $=$ dag) T_1 for f over Π_1

Output: a proof DAG T_2 for f over Π_2

For suitable fixed pairs $\Pi_1 \sqsubseteq \Pi_2$:

DAG-encoded input

PTIME-hard

reduction from Circuit Value Problem

tree-encoded input

LOGCFL-hard

reduction from a CFL word problem

PTIME-Hardness for DAG Encoding

Reduce from Circuit Value Problem: input facts encode a circuit.

$\mathsf{and}(g, g_1, g_2)$, $\mathsf{not}(g, g_1)$, $\mathsf{t_{in}}(g)$, $\mathsf{f_{in}}(g)$, $\mathsf{out}(g)$

PTIME-Hardness for DAG Encoding

Reduce from Circuit Value Problem: input facts encode a circuit.

$\mathsf{and}(g, g_1, g_2)$, $\mathsf{not}(g, g_1)$, $\mathsf{t_{in}}(g)$, $\mathsf{f_{in}}(g)$, $\mathsf{out}(g)$

Π_1 checks the structure

Π_2 evaluates the circuit

PTIME-Hardness for DAG Encoding

Reduce from Circuit Value Problem: input facts encode a circuit.

$\mathsf{and}(g, g_1, g_2)$, $\mathsf{not}(g, g_1)$, $\mathsf{t_{in}}(g)$, $\mathsf{f_{in}}(g)$, $\mathsf{out}(g)$

Π_1 checks the structure

$$\mathsf{maybe}(g) \leftarrow \mathsf{t_{in}}(g)$$

$$\mathsf{maybe}(g) \leftarrow \mathsf{f_{in}}(g)$$

$$\mathsf{maybe}(g) \leftarrow \mathsf{and}(g, g_1, g_2), \mathsf{maybe}(g_1), \mathsf{maybe}(g_2)$$

$$\mathsf{maybe}(g) \leftarrow \mathsf{not}(g, g_1), \mathsf{maybe}(g_1)$$

$$\mathsf{goal} \leftarrow \mathsf{maybe}(g), \mathsf{out}(g)$$

Π_2 evaluates the circuit

PTIME-Hardness for DAG Encoding

Reduce from Circuit Value Problem: input facts encode a circuit.

$\mathsf{and}(g, g_1, g_2)$, $\mathsf{not}(g, g_1)$, $\mathsf{t_{in}}(g)$, $\mathsf{f_{in}}(g)$, $\mathsf{out}(g)$

Π_1 checks the structure

$\mathsf{maybe}(g) \leftarrow \mathsf{t_{in}}(g)$

$\mathsf{maybe}(g) \leftarrow \mathsf{f_{in}}(g)$

$\mathsf{maybe}(g) \leftarrow \mathsf{and}(g, g_1, g_2), \mathsf{maybe}(g_1), \mathsf{maybe}(g_2)$

$\mathsf{maybe}(g) \leftarrow \mathsf{not}(g, g_1), \mathsf{maybe}(g_1)$

$\mathsf{goal} \leftarrow \mathsf{maybe}(g), \mathsf{out}(g)$

Π_2 evaluates the circuit

$\mathsf{true}(g) \leftarrow \mathsf{t_{in}}(g)$

$\mathsf{true}(g) \leftarrow \mathsf{and}(g, g_1, g_2), \mathsf{true}(g_1), \mathsf{true}(g_2)$

$\mathsf{true}(g) \leftarrow \mathsf{not}(g, g_1), \mathsf{false}(g_1)$

+ dual rules deriving false

$\mathsf{goal} \leftarrow \mathsf{true}(g), \mathsf{out}(g)$

$\mathsf{goal} \leftarrow \mathsf{false}(g), \mathsf{out}(g)$

PTIME-Hardness for DAG Encoding

Reduce from Circuit Value Problem: input facts encode a circuit.

$\mathsf{and}(g, g_1, g_2)$, $\mathsf{not}(g, g_1)$, $\mathsf{t_{in}}(g)$, $\mathsf{f_{in}}(g)$, $\mathsf{out}(g)$

Π_1 checks the structure

$\mathsf{maybe}(g) \leftarrow \mathsf{t_{in}}(g)$

$\mathsf{maybe}(g) \leftarrow \mathsf{f_{in}}(g)$

$\mathsf{maybe}(g) \leftarrow \mathsf{and}(g, g_1, g_2), \mathsf{maybe}(g_1), \mathsf{maybe}(g_2)$

$\mathsf{maybe}(g) \leftarrow \mathsf{not}(g, g_1), \mathsf{maybe}(g_1)$

$\mathsf{goal} \leftarrow \mathsf{maybe}(g), \mathsf{out}(g)$

Π_2 evaluates the circuit

$\mathsf{true}(g) \leftarrow \mathsf{t_{in}}(g)$

$\mathsf{true}(g) \leftarrow \mathsf{and}(g, g_1, g_2), \mathsf{true}(g_1), \mathsf{true}(g_2)$

$\mathsf{true}(g) \leftarrow \mathsf{not}(g, g_1), \mathsf{false}(g_1)$

+ dual rules deriving false

$\mathsf{goal} \leftarrow \mathsf{true}(g), \mathsf{out}(g)$

$\mathsf{goal} \leftarrow \mathsf{false}(g), \mathsf{out}(g)$

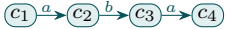
Exactly one goal-rule applies: the root rule gives the circuit's value

LogCFL-Hardness for Tree Encoding

Reduce from the word problem of a context-free language.

grammar in quadratic Greibach normal form: $A \to \sigma \mid \sigma A_1 \mid \sigma A_1 A_2$

first(c_1), last$(c_{\ell+1})$, next(c_i, σ_i, c_{i+1})

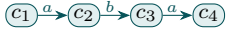


LogCFL-Hardness for Tree Encoding

Reduce from the word problem of a context-free language.

grammar in quadratic Greibach normal form: $A \to \sigma \mid \sigma A_1 \mid \sigma A_1 A_2$

$\mathsf{first}(c_1)$, $\mathsf{last}(c_{\ell+1})$, $\mathsf{next}(c_i, \sigma_i, c_{i+1})$



Π_1 checks the structure

$$\mathsf{r}(x, y) \leftarrow \mathsf{next}(x, \sigma, y)$$

$$\mathsf{r}(x, y) \leftarrow \mathsf{r}(x, z),\ \mathsf{r}(z, y)$$

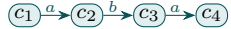
$$\mathsf{goal} \leftarrow \mathsf{r}(x, y), \mathsf{first}(x), \mathsf{last}(y)$$

LogCFL-Hardness for Tree Encoding

Reduce from the word problem of a context-free language.

grammar in quadratic Greibach normal form: $A \to \sigma \mid \sigma A_1 \mid \sigma A_1 A_2$

$\mathsf{first}(c_1)$, $\mathsf{last}(c_{\ell+1})$, $\mathsf{next}(c_i, \sigma_i, c_{i+1})$



Π_1 checks the structure

$$\mathsf{r}(x,y) \leftarrow \mathsf{next}(x,\sigma,y)$$

$$\mathsf{r}(x,y) \leftarrow \mathsf{r}(x,z),\ \mathsf{r}(z,y)$$

$$\mathsf{goal} \leftarrow \mathsf{r}(x,y), \mathsf{first}(x), \mathsf{last}(y)$$

Π_2 decides membership

$$\mathsf{A}(x,y) \leftarrow \mathsf{next}(x,\sigma,y)$$

$$\mathsf{A}(x,y) \leftarrow \mathsf{next}(x,\sigma,z), \mathsf{A}_1(z,y)$$

$$\mathsf{A}(x,y) \leftarrow \mathsf{next}(x,\sigma,z), \mathsf{A}_1(z,s),\ \mathsf{A}_2(s,y)$$

$$\bar{\mathsf{A}}(x,y) \leftarrow \text{`}A \text{ does not derive } w[x..y]\text{'}$$

$$\mathsf{goal} \leftarrow \mathsf{first}(x), \mathsf{last}(y), \mathsf{S}(x,y)$$

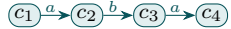
$$\mathsf{goal} \leftarrow \mathsf{first}(x), \mathsf{last}(y), \bar{\mathsf{S}}(x,y)$$

LogCFL-Hardness for Tree Encoding

Reduce from the word problem of a context-free language.

grammar in quadratic Greibach normal form: $A \to \sigma \mid \sigma A_1 \mid \sigma A_1 A_2$

$\mathsf{first}(c_1)$, $\mathsf{last}(c_{\ell+1})$, $\mathsf{next}(c_i, \sigma_i, c_{i+1})$



Π_1 checks the structure

$$\mathsf{r}(x,y) \leftarrow \mathsf{next}(x,\sigma,y)$$

$$\mathsf{r}(x,y) \leftarrow \mathsf{r}(x,z),\ \mathsf{r}(z,y)$$

$$\mathsf{goal} \leftarrow \mathsf{r}(x,y), \mathsf{first}(x), \mathsf{last}(y)$$

Π_2 decides membership

$$\mathsf{A}(x,y) \leftarrow \mathsf{next}(x,\sigma,y)$$

$$\mathsf{A}(x,y) \leftarrow \mathsf{next}(x,\sigma,z), \mathsf{A}_1(z,y)$$

$$\mathsf{A}(x,y) \leftarrow \mathsf{next}(x,\sigma,z), \mathsf{A}_1(z,s),\ \mathsf{A}_2(s,y)$$

$$\bar{\mathsf{A}}(x,y) \leftarrow \text{`}A \text{ does not derive } w[x..y]\text{'}$$

$$\mathsf{goal} \leftarrow \mathsf{first}(x), \mathsf{last}(y), \mathsf{S}(x,y)$$

$$\mathsf{goal} \leftarrow \mathsf{first}(x), \mathsf{last}(y), \bar{\mathsf{S}}(x,y)$$

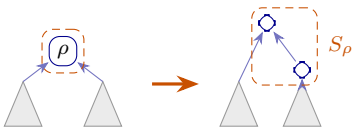
Exactly one goal-rule applies: the root rule decides whether $w \in L$

Proof Transformation Languages

Goal: Define proof transformation languages that are more efficient than recomputing the proof

Proof Tree Homomorphisms

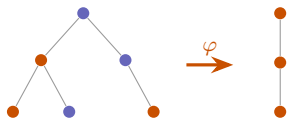
replace each input node
by a fixed template



MSO Transformations

define the output tree by
logical formulae:

$$\varphi \;=\; \exists y.\, \alpha(x, y) \,\wedge\, \ldots$$



Proof Tree Homomorphisms

Every rule $\rho \in \Pi_1$ is mapped to a proof tree template S_ρ:

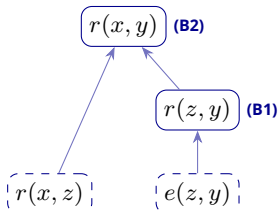
$\hookrightarrow$ a proof over Π_2 deriving the *head* of ρ from the *body atoms* of ρ

Proof Tree Homomorphisms

Every rule $\rho \in \Pi_1$ is mapped to a proof tree template S_ρ:

$\hookrightarrow$ a proof over Π_2 deriving the *head* of ρ from the *body atoms* of ρ

Template for rule **(L2)**: $r(x, y) \leftarrow r(x, z),\ e(z, y)$



Program Π_2

$r(x, y) \leftarrow e(x, y)$ **(B1)**

$r(x, y) \leftarrow r(x, z),\ r(z, y)$ **(B2)**

Proof Tree Homomorphisms

Proof tree homomorphism

Let $\{S_\rho\}_{\rho \in \Pi_1}$ contain a proof tree template over Π_2 for every rule $\rho \in \Pi_1$. The induced proof tree homomorphism τ maps proof trees over Π_1 to proof DAGs over Π_2, as follows:

1. if v is a leaf, then $\tau(T_{|v}) \coloneqq T_{|v}$;
2. if $\mathrm{rule}(v) = \rho$ with children $v_1, \ldots, v_n$, then $\tau(T_{|v}) \coloneqq S_\rho\big[\tau(T_{|v_1}), \ldots, \tau(T_{|v_n})\big]$

Proof Tree Homomorphisms

Proof tree homomorphism

Let $\{S_\rho\}_{\rho \in \Pi_1}$ contain a proof tree template over Π_2 for every rule $\rho \in \Pi_1$. The induced proof tree homomorphism τ maps proof trees over Π_1 to proof DAGs over Π_2, as follows:

1. if v is a leaf, then $\tau(T_{|v}) \coloneqq T_{|v}$;
2. if $\mathrm{rule}(v) = \rho$ with children $v_1, \ldots, v_n$, then $\tau(T_{|v}) \coloneqq S_\rho\big[\tau(T_{|v_1}), \ldots, \tau(T_{|v_n})\big]$

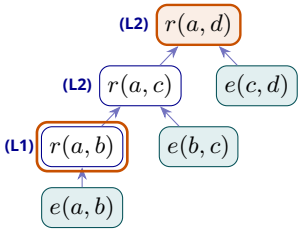


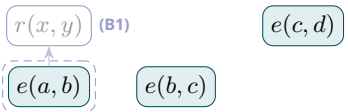
Proof Tree Homomorphisms

Proof tree homomorphism

Let $\{S_\rho\}_{\rho \in \Pi_1}$ contain a proof tree template over Π_2 for every rule $\rho \in \Pi_1$. The induced proof tree homomorphism τ maps proof trees over Π_1 to proof DAGs over Π_2, as follows:

1. if v is a leaf, then $\tau(T_{|v}) \coloneqq T_{|v}$;
2. if $\mathrm{rule}(v) = \rho$ with children $v_1, \ldots, v_n$, then $\tau(T_{|v}) \coloneqq S_\rho\big[\tau(T_{|v_1}), \ldots, \tau(T_{|v_n})\big]$



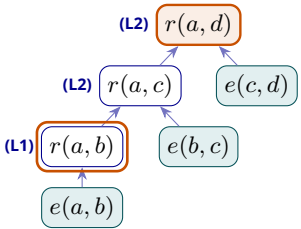


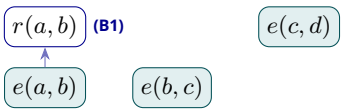
Proof Tree Homomorphisms

Proof tree homomorphism

Let $\{S_\rho\}_{\rho \in \Pi_1}$ contain a proof tree template over Π_2 for every rule $\rho \in \Pi_1$. The induced proof tree homomorphism τ maps proof trees over Π_1 to proof DAGs over Π_2, as follows:

1. if v is a leaf, then $\tau(T_{|v}) \coloneqq T_{|v}$;
2. if $\mathrm{rule}(v) = \rho$ with children $v_1, \ldots, v_n$, then $\tau(T_{|v}) \coloneqq S_\rho\big[\tau(T_{|v_1}), \ldots, \tau(T_{|v_n})\big]$



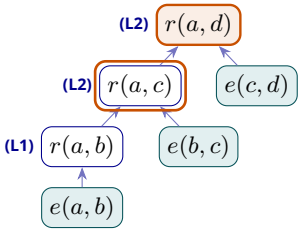


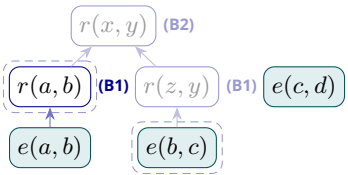
Proof Tree Homomorphisms

Proof tree homomorphism

Let $\{S_\rho\}_{\rho \in \Pi_1}$ contain a proof tree template over Π_2 for every rule $\rho \in \Pi_1$. The induced proof tree homomorphism τ maps proof trees over Π_1 to proof DAGs over Π_2, as follows:

1. if v is a leaf, then $\tau(T_{|v}) \coloneqq T_{|v}$;
2. if $\mathrm{rule}(v) = \rho$ with children $v_1, \ldots, v_n$, then $\tau(T_{|v}) \coloneqq S_\rho\big[\tau(T_{|v_1}), \ldots, \tau(T_{|v_n})\big]$



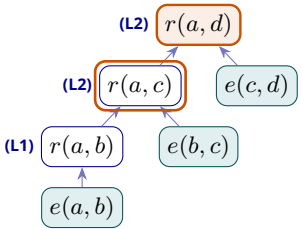


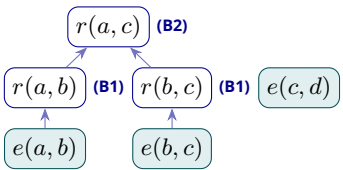
Proof Tree Homomorphisms

Proof tree homomorphism

Let $\{S_\rho\}_{\rho \in \Pi_1}$ contain a proof tree template over Π_2 for every rule $\rho \in \Pi_1$. The induced proof tree homomorphism τ maps proof trees over Π_1 to proof DAGs over Π_2, as follows:

1. if v is a leaf, then $\tau(T_{|v}) \coloneqq T_{|v}$;
2. if $\mathrm{rule}(v) = \rho$ with children $v_1, \ldots, v_n$, then $\tau(T_{|v}) \coloneqq S_\rho\big[\tau(T_{|v_1}), \ldots, \tau(T_{|v_n})\big]$



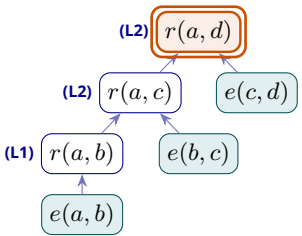


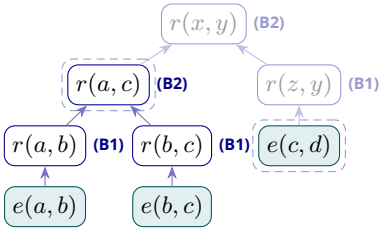
Proof Tree Homomorphisms

Proof tree homomorphism

Let $\{S_\rho\}_{\rho \in \Pi_1}$ contain a proof tree template over Π_2 for every rule $\rho \in \Pi_1$. The induced proof tree homomorphism τ maps proof trees over Π_1 to proof DAGs over Π_2, as follows:

1. if v is a leaf, then $\tau(T_{|v}) \coloneqq T_{|v}$;
2. if $\mathrm{rule}(v) = \rho$ with children $v_1, \ldots, v_n$, then $\tau(T_{|v}) \coloneqq S_\rho\big[\tau(T_{|v_1}), \ldots, \tau(T_{|v_n})\big]$



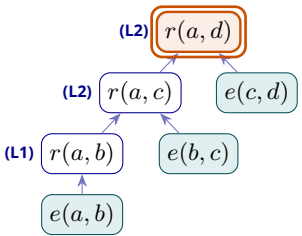


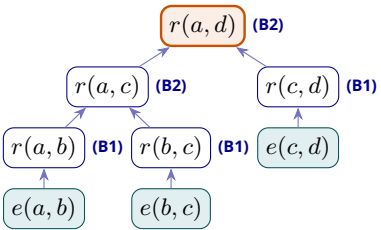
Proof Tree Homomorphisms

Proof tree homomorphism

Let $\{S_\rho\}_{\rho \in \Pi_1}$ contain a proof tree template over Π_2 for every rule $\rho \in \Pi_1$. The induced proof tree homomorphism τ maps proof trees over Π_1 to proof DAGs over Π_2, as follows:

1. if v is a leaf, then $\tau(T_{|v}) \coloneqq T_{|v}$;
2. if $\mathrm{rule}(v) = \rho$ with children $v_1, \ldots, v_n$, then $\tau(T_{|v}) \coloneqq S_\rho\big[\tau(T_{|v_1}), \ldots, \tau(T_{|v_n})\big]$





Uniform Containment

$\Pi_1 \sqsubseteq_u \Pi_2$ is a decidable alternative to containment [Sagiv 1988]:

$\iff$ $\Pi_1, \mathcal{D} \vdash f$ implies $\Pi_2, \mathcal{D} \vdash f$ for *every* database $\mathcal{D}$ and all facts f

Uniform Containment

$\Pi_1 \sqsubseteq_u \Pi_2$ is a decidable alternative to containment [Sagiv 1988]:

$\iff$ $\Pi_1, \mathcal{D} \vdash f$ implies $\Pi_2, \mathcal{D} \vdash f$ for *every* database $\mathcal{D}$ and all facts f

$\iff$ head of every rule $\rho \in \Pi_1$ can be derived from its body over Π_2 $(\dagger)$

Uniform Containment

$\Pi_1 \sqsubseteq_u \Pi_2$ is a decidable alternative to containment [Sagiv 1988]:

$\iff$ $\Pi_1, \mathcal{D} \vdash f$ implies $\Pi_2, \mathcal{D} \vdash f$ for *every* database $\mathcal{D}$ and all facts f

$\iff$ head of every rule $\rho \in \Pi_1$ can be derived from its body over Π_2 $(\dagger)$

Theorem

There is a proof tree homomorphism from Π_1 to Π_2 if and only if $\Pi_1 \sqsubseteq_u \Pi_2$.

proof of $(\dagger)$ corresponds to proof tree templates

Uniform Containment

$\Pi_1 \sqsubseteq_u \Pi_2$ is a decidable alternative to containment [Sagiv 1988]:

$\iff$ $\Pi_1, \mathcal{D} \vdash f$ implies $\Pi_2, \mathcal{D} \vdash f$ for *every* database $\mathcal{D}$ and all facts f

$\iff$ head of every rule $\rho \in \Pi_1$ can be derived from its body over Π_2 $(\dagger)$

Theorem

There is a proof tree homomorphism from Π_1 to Π_2 if and only if $\Pi_1 \sqsubseteq_u \Pi_2$.

proof of $(\dagger)$ corresponds to proof tree templates

But: no homomorphism turns *bushy* into *linear* proof trees

$\hookrightarrow$ this direction needs a global formalism: MSO transformations

MSO Transformations

Encode the proof tree T as a relational structure $\mathcal{I}(T)$

$\mathsf{child}_i(x, y)$	x is the i-th child of y
$\mathsf{rule}_\rho(x)$	rule ρ is applied at x
$\mathsf{arg}_j(x, d)$	the j-th argument of the fact at x is d

MSO Transformations

Encode the proof tree T as a relational structure $\mathcal{I}(T)$

$\mathsf{child}_i(x, y)$	x is the i-th child of y
$\mathsf{rule}_\rho(x)$	rule ρ is applied at x
$\mathsf{arg}_j(x, d)$	the j-th argument of the fact at x is d

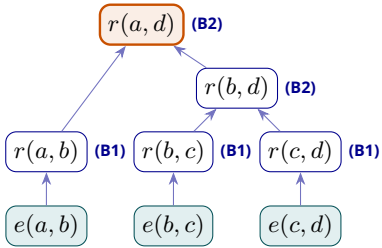
k-dimensional MSO transformation

Defines the output proof DAG over Π_2 by MSO formulas over $\mathcal{I}(T_1)$, whose nodes are k-tuples of input nodes:

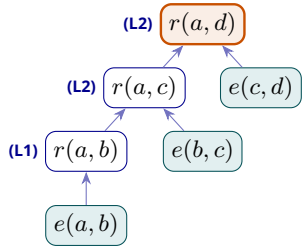
$\varphi^{\alpha}_{\mathsf{dom}}(\vec{x})$	… the k-tuples $\vec{x}$ of input nodes that become output nodes
$\varphi^{\alpha}_{\rho}(\vec{x})$	… which rule $\rho \in \Pi_2$ is applied at $\vec{x}$
$\varphi^{\alpha,\beta,i}_{\mathsf{child}}(\vec{x}, \vec{y})$	… whether $\vec{x}$ is the i-th child of $\vec{y}$
$\varphi^{\alpha,j}_{\mathsf{arg}}(\vec{x}, d)$	… the j-th argument of the fact at $\vec{x}$

Transforming Bushy into Linear Proof Trees

input: proof tree T

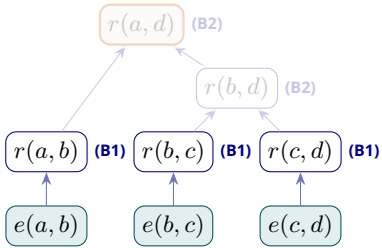


target: transformed tree $\tau(T)$

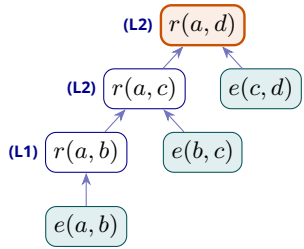


Transforming Bushy into Linear Proof Trees

input: proof tree T



target: transformed tree $\tau(T)$

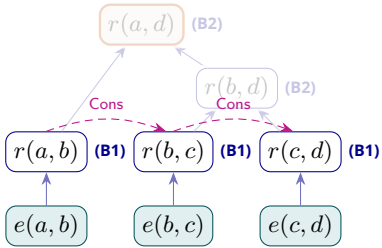


1. select the output nodes

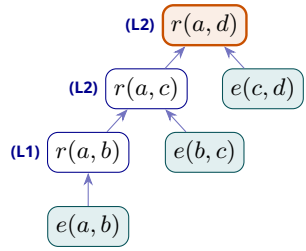
$$\varphi_{\mathsf{dom}}(x) := \neg\, \mathsf{rule}_{\textbf{(B2)}}(x)$$

Transforming Bushy into Linear Proof Trees

input: proof tree T



target: transformed tree $\tau(T)$



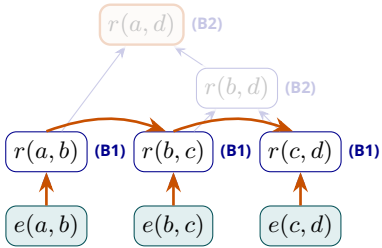
2. order the (B1)-nodes from left to right

$$u <_\ell v \coloneqq \mathsf{rule}_{\text{(B1)}}(u) \wedge \mathsf{rule}_{\text{(B1)}}(v) \wedge \exists p, a, b.\ \mathsf{child}_1(a,p) \wedge \mathsf{child}_2(b,p) \wedge u \preceq a \wedge v \preceq b$$

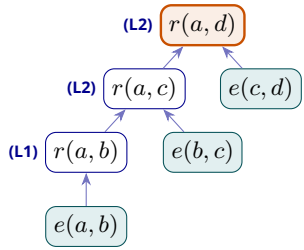
$$\mathsf{Cons}(u,v) \coloneqq u <_\ell v \wedge \neg\exists w.\ u <_\ell w \wedge w <_\ell v \qquad \mathsf{First}(v) \coloneqq \mathsf{rule}_{\text{(B1)}}(v) \wedge \neg\exists u.\ \mathsf{Cons}(u,v)$$

Transforming Bushy into Linear Proof Trees

input: proof tree T



target: transformed tree $\tau(T)$



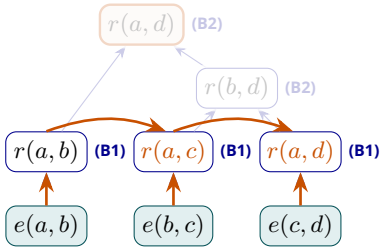
3. connect the children

$$\varphi^1_{\mathsf{child}}(x,y) \coloneqq \big(\mathsf{child}_1(x,y) \wedge \mathsf{First}(y)\big) \vee \mathsf{Cons}(x,y)$$

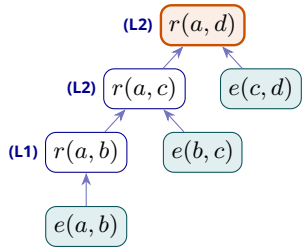
$$\varphi^2_{\mathsf{child}}(x,y) \coloneqq \neg\mathsf{First}(y) \wedge \mathsf{rule}_{\textbf{(B1)}}(y) \wedge \mathsf{child}_2(x,y)$$

Transforming Bushy into Linear Proof Trees

input: proof tree T



target: transformed tree $\tau(T)$



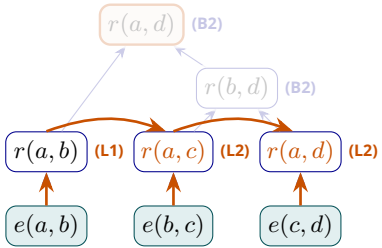
4. relabel the facts

$$\varphi^1_{\mathsf{arg}}(x,d) \coloneqq \big(\mathsf{Leaf}(x) \wedge \mathsf{arg}_1(x,d)\big) \vee \big(\mathsf{rule}_{\textbf{(B1)}}(x) \wedge \exists r.\, \mathsf{Root}(r) \wedge \mathsf{arg}_1(r,d)\big)$$

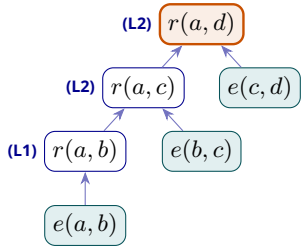
$$\varphi^2_{\mathsf{arg}}(x,d) \coloneqq \mathsf{arg}_2(x,d)$$

Transforming Bushy into Linear Proof Trees

input: proof tree T



target: transformed tree $\tau(T)$



5. assign the rules

$$\varphi_{\text{(L1)}}(x) := \mathsf{First}(x) \qquad \varphi_{\text{(L2)}}(x) := \exists y.\ \mathsf{Cons}(y, x)$$

Complexity of MSO Transformations

Theorem. MSO transformations are computable by LogSpace transducers.

Complexity of MSO Transformations

> **Theorem.** MSO transformations are computable by LogSpace transducers.

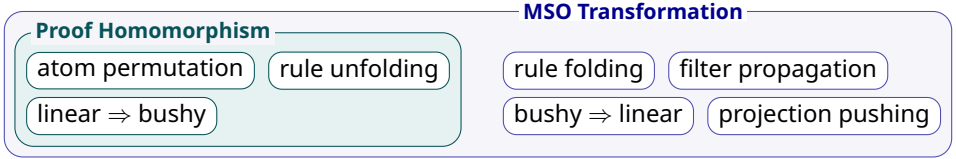
Recall: recovery is LogCFL-hard, so MSO likely cannot reverse *everything*

Complexity of MSO Transformations

> **Theorem.** MSO transformations are computable by LogSpace transducers.

Recall: recovery is LogCFL-hard, so MSO likely cannot reverse *everything*

Yet: common program transformations are all MSO-reversible

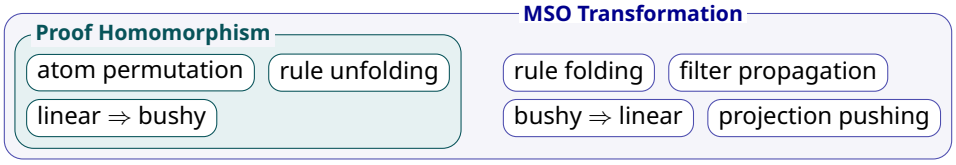


Complexity of MSO Transformations

Theorem. MSO transformations are computable by LogSpace transducers.

Recall: recovery is LogCFL-hard, so MSO likely cannot reverse *everything*

Yet: common program transformations are all MSO-reversible



Next step. Which program transformations are MSO-reversible?

Summary

We show how to recover explanations from program transformations.

Complexity

DAG-encoded input	PTIME-complete
tree-encoded input	LOGCFL-hard

Specification languages

local	proof homomorphisms
global	MSO proof transformations

Future work

- prove PTIME-completeness of the transformation problem with tree-encoded input
- characterize which program transformations are MSO-reversible
- implement proof-tree recovery in Nemo (fully)