Relation Algebra, Allegories, and Logic Programming

Emilio Jesús Gallego Arias
[joint work with J. Lipton, J. Mariño]

CRI-Mines ParisTech

Deducteam Seminar
17/10/2014 — Paris
Recall (Basic) Logic Programming

<table>
<thead>
<tr>
<th>Computation $\equiv$ Proof Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basics Program $(\Gamma)$, query $(\exists \vec{x}. \varphi)$, provability relation $(\vdash)$.</td>
</tr>
</tbody>
</table>

**Example**

```
add((0, X, X)).
```

```
add(X+1, Y, Z+1) <- add(X, Y, Z).
```

```
?- add(3, X, 4).  
{X = 1} ;
no more  
?- add(X, Y, Z).  
{X = o, Z = Y ?} ;
{X = s(o), Z = s(Y) ? }
```

Constraints: primitive class of formulas solved externally.
Recall (Basic) Logic Programming

<table>
<thead>
<tr>
<th>Computation ≡ Proof Search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basics</strong></td>
</tr>
<tr>
<td><strong>Execution</strong></td>
</tr>
</tbody>
</table>

Constraints: primitive class of formulas solved externally.
## Recall (Basic) Logic Programming

### Computation $\equiv$ Proof Search

<table>
<thead>
<tr>
<th>Basics</th>
<th>Program $(\Gamma)$, query $(\exists \vec{x}. \varphi)$, provability relation $(\vdash)$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution</td>
<td>Find a proof of $\Gamma \vdash \exists \vec{x}. \varphi$.</td>
</tr>
<tr>
<td>Output</td>
<td>Witnesses $\vec{a}$ for $\vec{x}$. <em>Possibly fresh variables in $\vec{a}$!</em></td>
</tr>
</tbody>
</table>

**Example**

```
add(0, X, X).
(\forall X) add(X+1, Y, Z+1) <- add(X, Y, Z).
(\forall XYZ) ?- add(3, X, 4).
{X = 1} ; no more
{X = o, Z = Y ?} ;
{X = s(o), Z = s(Y) ?} [more]
```

**Constraints:** primitive class of formulas solved externally.
Recall (Basic) Logic Programming

Computation $\equiv$ Proof Search

**Basics**  Program $(\Gamma)$, query $(\exists \vec{x}. \varphi)$, provability relation $(\vdash)$.

**Execution**  Find a proof of $\Gamma \vdash \exists \vec{x}. \varphi$.

**Output**  Witnesses $\tilde{a}$ for $\vec{x}$. *Possibly fresh variables in $\tilde{a}$!*

---

**Example**

\[
\begin{align*}
\text{add}(0, X, X). \\
\text{add}(X+1, Y, Z+1) & \leftarrow \text{add}(X, Y, Z). & (\forall X) \\
\text{add}(X+1, Y, Z+1) & \leftarrow \text{add}(X, Y, Z). & (\forall XYZ)
\end{align*}
\]
Recall (Basic) Logic Programming

**Computation ≡ Proof Search**

<table>
<thead>
<tr>
<th>Basics</th>
<th>Program ((\Gamma)), query ((\exists \vec{x}. \varphi)), provability relation ((\vdash)).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution</td>
<td>Find a proof of (\Gamma \vdash \exists \vec{x}. \varphi).</td>
</tr>
<tr>
<td>Output</td>
<td>Witnesses (\vec{a}) for (\vec{x}). <em>Possibly fresh variables in (\vec{a})</em>!</td>
</tr>
</tbody>
</table>

**Example**

\[
\text{add}(0, X, X).
\]
\[
\text{add}(X+1, Y, Z+1) \leftarrow \text{add}(X, Y, Z). \quad (\forall X)
\]
\[
\text{add}(X, Y, Z). \quad (\forall XYZ)
\]

\[- \text{add}(3, X, 4).\]
\[
\{X = 1\} \ ? ;
\]
\[
\text{no more}
\]
Recall (Basic) Logic Programming

### Computation ≡ Proof Search

- **Basics**: Program \((\Gamma)\), query \((\exists \vec{x}. \varphi)\), provability relation \((\vdash)\).
- **Execution**: Find a proof of \(\Gamma \vdash \exists \vec{x}. \varphi\).
- **Output**: Witnesses \(\vec{a}\) for \(\vec{x}\). Possibly fresh variables in \(\vec{a}\)!

### Example

\[
\begin{align*}
\text{add} & (0, X, X). \\
\text{add} & (X+1, Y, Z+1) \leftarrow \text{add} (X, Y, Z). \\
\end{align*}
\]

\[
\begin{align*}
?- \text{add} (3, X, 4). & \quad \text{?} - \text{add} (X, Y, Z). \\
\{X = 1\} \ ? \ ; & \quad \{X = o, Z = Y \ ?\} \ ; \\
\text{no more} & \quad \{X = s(o), Z = s(Y) \ ?\} \ [\text{more}] \\
\end{align*}
\]

Constraints: primitive class of formulas solved externally.
What is our goal?

To reason about logic programming equationally. ("Point-free style")
What is our goal?

To reason about logic programming equationally. ("Point-free style")

Proof search
What is our goal?

To reason about logic programming equationally. ("Point-free style")

Example

\[ \text{nat}(o) \leftarrow \top \quad \text{nat}(s(X)) \leftarrow \text{nat}(X) \]
Combinatorial Logic Programming

What is our goal?
To reason about logic programming equationally. ("Point-free style")

Example

\[
\text{nat}(o) \leftarrow \top \\
\forall X . \text{nat}(s(X)) \leftarrow \text{nat}(X)
\]
Combinatorial Logic Programming

What is our goal?
To reason about logic programming equationally. (“Point-free style”)

Example

\[
\text{nat}(o) \leftarrow \top \\
\forall X . \text{nat}(s(X)) \leftarrow \text{nat}(X)
\]

Point-free means no variables:

\[
\overline{\text{nat}} = \{o\} \cup s \cdot \overline{\text{nat}}
\]
Combinatorial Logic Programming

What is our goal?

To reason about logic programming equationally. (“Point-free style”)

Example

\[
\text{nat}(o) \leftarrow \top \\
\forall X. \text{nat}(s(X)) \leftarrow \text{nat}(X)
\]

Point-free means no variables:

\[
\overline{\text{nat}} = \{o, s(o)\} \cup s \cdot s \cdot \overline{\text{nat}}
\]

and equational program reasoning
Combinatoric Logic Programming

What is our goal?

To reason about logic programming equationally. ("Point-free style")

Example

\[
\text{nat}(o) \leftarrow \top \\
\forall X . \text{nat}(s(X)) \leftarrow \text{nat}(X)
\]

Point-free means no variables:

\[
\text{nat} = \{o, s(o), s(s(o))\} \cup s \cdot s \cdot s \cdot \overline{\text{nat}}
\]

and equational program reasoning
What is our goal?
To reason about logic programming equationally. (“Point-free style”)

Example

\[
\text{nat}(o) \leftarrow \top \\
\forall X . \text{nat}(s(X)) \leftarrow \text{nat}(X)
\]

Point-free means no variables:

\[
\overline{\text{nat}} = \{ o, s(o), s(s(o)) \} \cup s \cdot s \cdot s \cdot \overline{\text{nat}}
\]

and equational program reasoning and proof search:

\[
s(s(o)) \cap \overline{\text{nat}}
\]
An Existential Problem

Why this is hard?

- Point-free programming is well studied, what is the problem here?
### An Existential Problem

#### Why this is hard?

- Point-free programming is well studied, what is the problem here?
- **Existential variables** have *global scope* in LP.

```prolog
p(s(Y)).
```

Two proof witnesses for $p$ may not be equal, given it has the "power" to generate a fresh variable every time it has to be proved. Quantifiers capture this, but use variables! Also, operational reasoning involves tricky renaming apart, etc. . .

Use an algebraic theory of logic and quantification!
An Existential Problem

Why this is hard?

- Point-free programming is well studied, what is the problem here?
- Existential variables have *global scope* in LP.

\[
\begin{align*}
p(s(Y)). & \quad \text{and} \quad p(X) :- \exists Y, \ X = s(Y) \\
? \ p(X). & \end{align*}
\]
An Existential Problem

Why this is hard?

- Point-free programming is well studied, what is the problem here?
- **Existential variables** have *global scope* in LP.

\[
p(s(Y)). \quad \%
p(X) : - \ \exists \ Y, \ X = s(Y)
\]

? p(X).

\{
X = s(_X13)
\}
An Existential Problem

Why this is hard?

- Point-free programming is well studied, what is the problem here?
- **Existential variables** have *global scope* in LP.

\[
p(s(Y)). \quad \text{%% } p(X) :- \exists Y, \ X = s(Y)
\]

? p(X).

\{ X = s(_X13) \}

- Two proof witnesses for \( p \) may not be equal, given it has the "power" to generate a fresh variable every time it has to be proved.
An Existential Problem

Why this is hard?

- Point-free programming is well studied, what is the problem here?
- **Existential variables** have *global scope* in LP.

\[
p(s(Y)). \quad \%
\]
\[
? \quad p(X). \quad \%
\]
\[
\{ \quad X = s(_X13) \quad \}
\]

- Two proof witnesses for \( p \) may not be equal, given it has the “power” to generate a fresh variable every time it has to be proved.
- Quantifiers capture this, but use variables! Also, operational reasoning involves tricky renaming apart, etc. . .
An Existential Problem

Why this is hard?

- Point-free programming is well studied, what is the problem here?
- **Existential variables** have *global scope* in LP.

\[
p(s(Y)). \\
p(X) :- \exists Y, X = s(Y) \\
\{ X = s(_X13) \}
\]

- Two proof witnesses for \( p \) may not be equal, given it has the “power” to generate a fresh variable every time it has to be proved.
- Quantifiers capture this, but use variables! Also, operational reasoning involves tricky renaming apart, etc.
- **Use an algebraic theory of logic and quantification!**
## Distributive Relation Algebras

Operations $\cap, \cup, (\cdot)^\circ, \cdot; \cdot$ satisfying the intended laws of binary relations. Introduced by Peirce-Schröder in the 19th century, and further developed by Tarski and his students in mid 20th century.
**Relation Algebra to the Rescue!**

### Distributive Relation Algebras

Operations $\cap$, $\cup$, $(\cdot)^\circ$, $\cdot\cdot\cdot$ satisfying the intended laws of binary relations. Introduced by Peirce-Schröder in the 19\textsuperscript{th} century, and further developed by Tarski and his students in mid 20\textsuperscript{th} century.

### Relations and Logic

$\cap$ as $\wedge$ and $\cup$ as $\vee$; limited to formulas with at most 3 variables!
Relation Algebra to the Rescue!

**Distributive Relation Algebras**

Operations $\cap$, $\cup$, $(\cdot)^\circ$, $\cdot$; $\cdot$ satisfying the intended laws of binary relations. Introduced by Peirce-Schröder in the 19th century, and further developed by Tarski and his students in mid 20th century.

**Relations and Logic**

$\cap$ as $\land$ and $\cup$ as $\lor$; limited to formulas with at most 3 variables!

**Extended Relation Algebras**

QRA: add (quasi) projections $hd$, $tl$; no limit on variables. Freyd-Maddux-Tarski: QRA capture set theory (“equipollent”).
Relation Algebra to the Rescue!

Distributive Relation Algebras

Operations \(\cap, \cup, (\cdot)^\circ, \cdot; \cdot\) satisfying the intended laws of binary relations. Introduced by Peirce-Schröder in the 19\textsuperscript{th} century, and further developed by Tarski and his students in mid 20\textsuperscript{th} century.

Relations and Logic

\(\cap\) as \(\land\) and \(\cup\) as \(\lor\); limited to formulas with at most 3 variables!

Extended Relation Algebras


Allegories

Due to Freyd. We use it as a typed RA, better for our purposes.
The Plan

Compile, interpret and execute CLP to a QRA:

- Semantics (set of true instances):
  - $\text{QRA}_\Sigma$.
  - $\Sigma$-allegories.

- Translation, logical meta-aspects and variables formalized at the relational level.
  - Program to theory between ground terms.
  - Program to allegory. Sharing and memory is captured.

- Execution: Two notions of rewriting. Executable semantics. Many extensions and optimizations possible declaratively: partial evaluation, abstract interpretation, different search strategies...
The Plan

Compile, interpret and execute CLP to a QRA:

- Semantics (set of true instances):
  - $\text{QRA}_\Sigma$.
  - $\Sigma$-allegories.

- Translation, logical meta-aspects and variables formalized at the relational level.
  - Program to theory between ground terms.
  - Program to allegory. Sharing and memory is captured.

- Execution: Two notions of rewriting. Executable semantics. Many extensions and optimizations possible declaratively: partial evaluation, abstract interpretation, different search strategies...

---

Logic Without Variables + Logic Programming

<table>
<thead>
<tr>
<th>Formula</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof</td>
<td>Equational Derivation</td>
</tr>
</tbody>
</table>
The Plan

Compile, interpret and execute CLP to a QRA:

- Semantics (set of true instances):
  - $QRA_{\Sigma}$.
  - $\Sigma$-allegories.

- Translation, logical meta-aspects and variables formalized at the relational level.
  - Program to theory between \textit{ground} terms.
  - Program to allegory. Sharing and memory is captured.

- Execution: Two notions of rewriting. Executable semantics. Many extensions and optimizations possible \textit{declaratively}: partial evaluation, abstract interpretation, different search strategies…

Logic Without Variables + Logic Programming

<table>
<thead>
<tr>
<th>Program</th>
<th>Formula</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Proof Search</td>
<td>Equational Derivation</td>
</tr>
</tbody>
</table>
The Plan

Compile, interpret and execute CLP to a QRA:

- Semantics (set of true instances):
  - $QRA_\Sigma$.
  - $\Sigma$-allegories.

- Translation, logical meta-aspects and variables formalized at the relational level.
  - Program to theory between *ground* terms.
  - Program to allegory. Sharing and memory is captured.

- Execution: Two notions of rewriting. Executable semantics. Many extensions and optimizations possible *declaratively*: partial evaluation, abstract interpretation, different search strategies...
The Relational Theory

Assume a signature $\Sigma \equiv \{C, F, CP, P\}$ and Constr. Dom. $D$.
Assume a signature $\Sigma \equiv \{C, F, CP, P\}$ and Constr. Dom. $D$.

### Generate the relational language:

| $R_C$ | $\{ (a, a) | a \in \mathcal{C}_\Sigma \}$ |
|-------|--------------------------------|
| $R_F$ | $\{ R_f | f \in \mathcal{F}_\Sigma, \}$ |
| $R_{CP}$ | $\{ r | r \in \mathcal{CP}_\Sigma \}$ |
| $R_P$ | $\{ \bar{p} | p \in \mathcal{P}_\Sigma \}$ |

$R_{atom} ::= R_C \mid R_F \mid R_{CP} \mid R_P \mid id \mid di \mid 1 \mid 0 \mid hd \mid tl$

$R_\Sigma ::= R_{atom} \mid R_\Sigma^\circ \mid R_\Sigma \cup R_\Sigma \mid R_\Sigma \cap R_\Sigma \mid R_\Sigma R_\Sigma$
The Relational Theory

Assume a signature \( \Sigma \equiv \{ C, F, CP, P \} \) and Constr. Dom. \( D \).

Generate the relational language:

\[
\begin{align*}
R_C &= \{ (a, a) \mid a \in C_\Sigma \} \\
R_F &= \{ R_f \mid f \in F_\Sigma \} \\
R_{CP} &= \{ r \mid r \in CP_\Sigma \} \\
R_P &= \{ \overline{p} \mid p \in P_\Sigma \} \\
R_{\text{atom}} &::= R_C \mid R_F \mid R_{CP} \mid R_P \mid id \mid di \mid 1 \mid 0 \mid \text{hd} \mid tl \\
R_\Sigma &::= R_{\text{atom}} \mid R_\Sigma^\circ \mid R_\Sigma \cup R_\Sigma \mid R_\Sigma \cap R_\Sigma \mid R_\Sigma R_\Sigma
\end{align*}
\]

Interpretation

\[
[\cdot] : R_\Sigma \rightarrow \mathcal{P}(D^+ \times D^+), \text{ where } D^+ = \bigcup_n D^n.
\]
The Relational Theory

Assume a signature \( \Sigma \equiv \{ C, F, CP, P \} \) and Constr. Dom. \( D \).

Generate the relational language:

\[
\begin{align*}
R_C &= \{(a, a) \mid a \in C_\Sigma\} & R_F &= \{R_f \mid f \in F_\Sigma, \}\bigg) \\
R_{CP} &= \{r \mid r \in CP_\Sigma\} & R_P &= \{p \mid p \in P_\Sigma\}
\end{align*}
\]
\[
R_{\text{atom}} ::= R_C \mid R_F \mid R_{CP} \mid R_P \mid id \mid di \mid 1 \mid 0 \mid hd \mid tl
\]
\[
R_\Sigma ::= R_{\text{atom}} \mid R_\Sigma^\circ \mid R_\Sigma \cup R_\Sigma \mid R_\Sigma \cap R_\Sigma \mid R_\Sigma R_\Sigma
\]

Interpretation

\( \llbracket \cdot \rrbracket : R_\Sigma \rightarrow \mathcal{P}(D^\dagger \times D^\dagger) \), where \( D^\dagger = \bigcup_n D^n \).

Example

\[
\llbracket \leq \rrbracket = \{(\langle m, n \rangle \bar{u}, \langle m, n \rangle \bar{u}') \mid m \leq n; m, n \in \mathbb{R}, \bar{u}, \bar{u}' \in D^\dagger\}
\]
The Relational Theory

Assume a signature $\Sigma \equiv \{C, F, CP, P\}$ and Constr. Dom. $D$.

Generate the relational language:

$R_C = \{(a, a) \mid a \in C_\Sigma\}$

$R_F = \{R_f \mid f \in F_\Sigma,\}$

$R_{CP} = \{r \mid r \in CP_\Sigma\}$

$R_P = \{\overline{p} \mid p \in P_\Sigma\}$

$R_{atom} ::= R_C \mid R_F \mid R_{CP} \mid R_P \mid id \mid di \mid 1 \mid 0 \mid hd \mid tl$

$R_\Sigma ::= R_{atom} \mid R_\Sigma^\circ \mid R_\Sigma \cup R_\Sigma \mid R_\Sigma \cap R_\Sigma \mid R_\Sigma R_\Sigma$

Interpretation

$[\cdot] : R_\Sigma \rightarrow P(D^+ \times D^+)$, where $D^+ = \bigcup_n D^n$.

Example

$[\leq] = \{(\langle m, n \rangle \bar{u}, \langle m, n \rangle \bar{u}') \mid m \leq n; m, n \in \mathbb{R}, \bar{u}, \bar{u}' \in D^+\}$

$[\text{add}] = \{(\langle m, n, o \rangle \bar{u}, \langle m, n, o \rangle \bar{u}') \mid m + n = o; m, n, o \in \mathbb{N}, \bar{u}, \bar{u}' \in D^+\}$
Projection and Permutations

$P_i$ is the relation projecting the $i$-th component of a vector; given a permutation $\pi$, $W_\pi$ is the associated relation.
Translation Overview: Helpers

Projections and Permutations

$P_i$ is the relation projecting the $i$-th component of a vector; given a permutation $\pi$, $W_\pi$ is the associated relation.

Partial Identity and Existential Quantification

The quasi-identity relation $Q_i$ is such that $(\vec{u}, \vec{v}) \in [Q_i]$ if the all but $i$-th component of $\vec{u}$ and $\vec{v}$ agree.
Projections and Permutations

$P_i$ is the relation projecting the $i$-th component of a vector; given a permutation $\pi$, $W_\pi$ is the associated relation.

Partial Identity and Existential Quantification

The quasi-identity relation $Q_i$ is such that $(\vec{u}, \vec{v}) \in [Q_i]$ if the all but $i$-th component of $\vec{u}$ and $\vec{v}$ agree.

Wrapping a relation $R$ in $Q_i R Q_i$ has the logical effect of existentially quantifying $i!$

\[
Q_i; \quad \underbrace{R \cdots S}_{i-\text{private for } R \cdots S}; \quad Q_i
\]
Translation Overview: Helpers

Projections and Permutations

\( P_i \) is the relation projecting the \( i \)-th component of a vector; given a permutation \( \pi \), \( W_\pi \) is the associated relation.

Partial Identity and Existential Quantification

The quasi-identity relation \( Q_i \) is such that \((\vec{u}, \vec{v}) \in [Q_i]\) if the all but \( i \)-th component of \( \vec{u} \) and \( \vec{v} \) agree.

Wrapping a relation \( R \) in \( Q_i RQ_i \) has the logical effect of existentially quantifying \( i \)!

\[
Q_i; \underbrace{R \cdots S}_{\text{i-private for } R \cdots S}; Q_i
\]

We’ll also use a variation of \( Q_i \), \( I_n \) that “hides” all the elements greater than \( n \).
Translation Overview [1/3]: Terms

Key Idea

A term $t[\bar{x}] \in \mathcal{T}_\Sigma(\mathcal{X})$ is translated to a relation between all its ground instances and instantiations for $\bar{x}$:
Translation Overview [1/3]: Terms

Key Idea

A term \( t[\vec{x}] \in \mathcal{T}_\Sigma(\mathcal{X}) \) is translated to a relation between all its \emph{ground} instances and instantiations for \( \vec{x} \):

\[
(b, \vec{a}\vec{u}) \in \llbracket K(t[\vec{x}]) \rrbracket^{D^+} \iff b = t^D[\vec{a}/\vec{x}]
\]
Translation Overview [1/3]: Terms

Key Idea
A term \( t[\vec{x}] \in T_\Sigma(\mathcal{X}) \) is translated to a relation between all its ground instances and instantiations for \( \vec{x} \):

\[
(b, \vec{a}\vec{u}) \in [K(t[\vec{x}])]^D^+ \iff b = t^D[\vec{a}/\vec{x}]
\]

Formally:

\[
K(t) : T_\Sigma(\mathcal{X}) \rightarrow R_\Sigma = \begin{cases} 
(c, c)1 & \text{if } t \equiv c \\
 P^\circ_i & \text{if } t \equiv x_i \\
 \bigcap_{i \leq n} f_i^r K(t_i) & \text{if } t \equiv f(t_1, \ldots, t_n)
\end{cases}
\]
Translation Overview [1/3]: Terms

Key Idea

A term $t[\bar{x}] \in T_\Sigma(\mathcal{X})$ is translated to a relation between all its **ground** instances and instantiations for $\bar{x}$:

$$(b, \bar{a}\bar{u}) \in \left[ K(t[\bar{x}]) \right]^{D^+} \iff b = t^D[\bar{a}/\bar{x}]$$

Formally:

$$K(t) : T_\Sigma(\mathcal{X}) \rightarrow R_\Sigma = \begin{cases} 
(c, c) \mathbf{1} & \text{if } t \equiv c \\
 P^o_i & \text{if } t \equiv x_i \\
 \bigcap_{i \leq n} f^r_i K(t_i) & \text{if } t \equiv f(t_1, \ldots, t_n) 
\end{cases}$$

Example

$$K(f(x_1, g(x_2, a, h(x_1)))) = f^2_1; P^o_1 \cap f^2_1; (g^3_1; P^o_2 \cap g^2_1; (a, a); \mathbf{1} \cap g^3_3; h; P^o_1)$$
Translation Overview [2/3]: Constraints

Key Idea

A constraint $\varphi[\vec{x}] \in \mathcal{L}_D$ is translated to the set of all its \textit{ground} solutions, encoded as a \textit{coreflexive} relation:
Translation Overview [2/3]: Constraints

Key Idea

A constraint $\phi[\vec{x}] \in \mathcal{L}_D$ is translated to the set of all its *ground* solutions, encoded as a *coreflexive* relation:

$$(\bar{a}\bar{u}, \bar{a}\bar{u}') \in \lbrack \dot{K}(\phi[\vec{x}])\rbrack^{D^\dagger} \iff D \models \phi[\bar{a}/\vec{x}]$$
Translation Overview [2/3]: Constraints

Key Idea

A constraint \( \varphi[\vec{x}] \in \mathcal{L}_D \) is translated to the set of all its *ground* solutions, encoded as a coreflexive relation:

\[
(\vec{a}u, \vec{a}u') \in [\dot{K}(\varphi[\vec{x}])]^{D^+} \iff D \models \varphi[\vec{a}/\vec{x}]
\]

Formally:

\[
\dot{K}(t) : \mathcal{L}_D \rightarrow R_\Sigma = \begin{cases} 
K(\vec{t}) \circ p; K(\vec{t}) & \text{if } \varphi \equiv p(\vec{t}) \\
\dot{K}(\varphi) \cap \dot{K}(\theta) & \text{if } \varphi \equiv \varphi \land \theta \\
Q_i; \dot{K}(\varphi); Q_i & \text{if } \varphi \equiv \exists x_i. \varphi
\end{cases}
\]
Translation Overview [2/3]: Constraints

Key Idea

A constraint \( \varphi[\vec{x}] \in \mathcal{L}_D \) is translated to the set of all its ground solutions, encoded as a coreflexive relation:

\[
(\vec{a}u, \vec{a}u') \in \left[ \dot{\mathcal{K}}(\varphi[\vec{x}]) \right]^{D^+} \iff D \models \varphi[\vec{a}/\vec{x}]
\]

Formally:

\[
\dot{\mathcal{K}}(t) : \mathcal{L}_D \rightarrow R_\Sigma = \begin{cases} 
K(\vec{t})^\circ; p; K(\vec{t}) & \text{if } \varphi \equiv p(\vec{t}) \\
\dot{\mathcal{K}}(\varphi) \cap \dot{\mathcal{K}}(\theta) & \text{if } \varphi \equiv \varphi \land \theta \\
Q_i; \dot{\mathcal{K}}(\varphi); Q_i & \text{if } \varphi \equiv \exists x_i. \varphi
\end{cases}
\]

Example

\[
\dot{\mathcal{K}}(\exists x_1 x_2.s(x_1) \leq x_2) = Q_1 Q_2; (P_1^\circ; s^\circ; P_1 \cap P_2^\circ; P_2) ; \leq; (P_1; s; P_1^\circ \cap P_2; P_2^\circ); Q_1 Q_2
\]
Key Idea

Defined predicates $p$ are translated to equations $\overline{p} \models R$. 
### Key Idea

Defined predicates $p$ are translated to equations $\bar{p} \models R$.

### Theorem (Adequacy)

\[(\bar{\alpha}_{u}, \bar{\alpha}_{u'}) \in [\bar{p}]^{D^+} \iff p(\bar{a}) \in T^\omega_p\]
## Translation Overview [3/3]: Predicates

### Key Idea
Defined predicates $p$ are translated to equations $\overline{p} \circledR R$.

### Theorem (Adequacy)

$$(\overline{\overline{a}u}, \overline{\overline{a}u'}) \in [\overline{p}]^{D^+} \iff p(\overline{a}) \in T^\omega_P$$

### The Procedure

1. Purify clause’ heads, canonical renaming, Clark completion.
2. The Relational Step!
Translation Overview [3/3]: Predicates

Key Idea
Defined predicates \( p \) are translated to equations \( \overline{p} \models R \).

Theorem (Adequacy)
\[
(\overline{a\bar{u}}, \overline{a\bar{u}'} \in [\overline{p}]^{D^+} \iff p(\overline{a}) \in T_{\omega}^p
\]

The Procedure
1. Purify clause’ heads, canonical renaming, Clark completion.
2. The Relational Step!
   \[
   \text{add}(x_1, x_2, x_3) \iff x_1 = o, x_2 = x_3.
   \]
   \[
   \text{add}(x_1, x_2, x_3) \iff x_1 = s(x_4), x_3 = s(x_5), \text{add}(x_4, x_2, x_5).
   \]
Translation Overview [3/3]: Predicates

Key Idea

Defined predicates $p$ are translated to equations $\overline{p} \Rightarrow R$.

Theorem (Adequacy)

$$(\vec{a}\vec{u}, \vec{a}\vec{u}') \in [\overline{p}]^D \iff p(\vec{a}) \in T^\omega_p$$

The Procedure

1. Purify clause’ heads, canonical renaming, Clark completion.

2. The Relational Step!

$$add(x_1, x_2, x_3) \leftarrow x_1 = o, x_2 = x_3.$$  
$$add(x_1, x_2, x_3) \leftarrow x_1 = s(x_4), x_3 = s(x_5), add(x_4, x_2, x_5).$$  
$\underbrace{add}_K(x_1 = o \land x_2 = x_3)$  
$\cup l_3; \hat{K}(x_1 = s(x_4) \land x_3 = s(x_5)); W; \overline{add}; W^o; l_3$
Program Execution

Relations and Computation

\[ r \land (s \lor t) \iff r \land s \lor r \land t \quad R \cap (S \cup T) = (R \cap S) \cup (R \cap T) \]
### Relations and Computation

<table>
<thead>
<tr>
<th>Relation</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r \land (s \lor t) \iff r \land s \lor r \land t$</td>
<td>$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$</td>
</tr>
<tr>
<td>Computation rule</td>
<td>$R \cap (S \cup T) \mapsto (R \cap S) \cup (R \cap T)$</td>
</tr>
</tbody>
</table>
### Relations and Computation

<table>
<thead>
<tr>
<th>Computation rule</th>
<th>Cut rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r \land (s \lor t) \iff r \land s \lor r \land t )</td>
<td>( R \cap (S \cup T) = (R \cap S) \cup (R \cap T) )</td>
</tr>
<tr>
<td>( R \cap (S \cup T) \iff (R \cap S) \cup (R \cap T) )</td>
<td>( S \cup T \supseteq S )</td>
</tr>
<tr>
<td>( S \cup T \iff S )</td>
<td>( S \cup T \iff S )</td>
</tr>
</tbody>
</table>
Program Execution

Relations and Computation

\[ r \land (s \lor t) \iff r \land s \lor r \land t \]

Computation rule

\[ R \cap (S \cup T) = (R \cap S) \cup (R \cap T) \]

Cut rule

\[ R \cap (S \cup T) \Rightarrow (R \cap S) \cup (R \cap T) \]

\[ S \cup T \supseteq S \]

\[ S \cup T \Rightarrow S \]

Queries

For executing a query, we just intersect and rewrite, for instance for \( \text{add}(o, o, o) \):

\[ \dot{K}(x_1 = x_2 = x_3 = o) \cap \overline{\text{add}} \]
Rewriting and the Modular Law

Some Sample Rules

\[ 0 \cup R \rightarrow_P R \]
\[ 0 \cap R \rightarrow_P 0 \]
\[ (R \cup S) \cap T \rightarrow_P (R \cap T) \cup (S \cap T) \]

Meta Rules

Calls to the constraint solver are modeled by meta-rewriting rules:

\[ \check{K}(\psi_1) \cap \check{K}(\psi_2) \rightarrow \check{K}(\psi_1 \land \psi_2) \]

Procedure Call: The Modular Law

\[ \check{K}(\psi) \cap \imath_m(R) \rightarrow_P \imath_m(\check{K}(\psi)) \cap R \cap \check{K}(\psi) \]

We hide variables in \( \psi \) that may be in conflict with variables in \( R \), but we need to “unhide” them later.
Example Execution

Query: \( \text{add}(o, s(o), X) \)

\[
1 \quad l_3 K(o, s(o), x_3) \cap \text{add} \quad \rightarrow
\]
Example Execution

Query: \texttt{add}(o, s(o), X)

\begin{align*}
1 \quad & l_3 \dot{K}(o, s(o), x_3) l_3 \cap \overline{\text{add}} \quad \rightarrow \\
2 \quad & l_3 \dot{K}(o, s(o), x_3) l_3 \cap (\dot{K}(o, x_2, x_2) \cup \\
& (l_3 [\dot{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap W \overline{\text{add}} W^\circ] l_3)) \quad \rightarrow 
\end{align*}
Example Execution

Query: \( add(o, s(o), X) \)

1. \( l_3 \tilde{K}(o, s(o), x_3) l_3 \cap add \rightarrow \)
2. \( l_3 \tilde{K}(o, s(o), x_3) l_3 \cap (\tilde{K}(o, x_2, x_2) \cup (l_3[\tilde{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap W \ add \ W^\circ] l_3)) \rightarrow \)
3. \( l_3[\tilde{K}(o, s(o), x_3) \cap \tilde{K}(o, x_2, x_2)] l_3 \cup l_3(\tilde{K}(o, s(o), x_3) \cap \tilde{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap W \ add \ W^\circ) l_3 \rightarrow \)
Example Execution

Query: \( \text{add}(o, s(o), X) \)

1. \( l_3 \hat{K}(o, s(o), x_3) l_3 \cap \overline{\text{add}} \) →
2. \( l_3 \hat{K}(o, s(o), x_3) l_3 \cap (\hat{K}(o, x_2, x_2) \cup (l_3 [\hat{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap W \overline{\text{add}} W^\circ] l_3)) \) →
3. \( l_3 [\hat{K}(o, s(o), x_3) \cap \hat{K}(o, x_2, x_2)] l_3 \cup l_3 (\hat{K}(o, s(o), x_3) \cap \hat{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap W \overline{\text{add}} W^\circ) l_3 \) →
4. \( \hat{K}(o, s(o), s(o)) \cup l_3 [0 \cap W \overline{\text{add}} W^\circ] l_3 \) →
Example Execution

Query: \texttt{add}(o, s(o), X)

1. $l_3 \dot{K}(o, s(o), x_3) l_3 \cap \overline{\text{add}}$
2. $l_3 \dot{K}(o, s(o), x_3) l_3 \cap (\dot{K}(o, x_2, x_2) \cup$
   \hspace{1cm} $(l_3 [\dot{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap W \overline{\text{add } W^\circ}) l_3]))) \rightarrow$
3. $l_3 [\dot{K}(o, s(o), x_3) \cap \dot{K}(o, x_2, x_2)] l_3 \cup$
   \hspace{1cm} $l_3 (\dot{K}(o, s(o), x_3) \cap \dot{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap$
   \hspace{1cm} $W \overline{\text{add } W^\circ}) l_3 \rightarrow$
4. $\dot{K}(o, s(o), s(o)) \cup l_3 [\emptyset \cap W \overline{\text{add } W^\circ}] l_3 \rightarrow$
5. $\dot{K}(o, s(o), s(o))$
Why move to Categories?

An old thought (2004)

“We need types to run fast and allocate memory for the relations.”
Just a implementor’s intuition.
Why move to Categories?

An old thought (2004)

“We need types to run fast and allocate memory for the relations.”
Just a implementor’s intuition.

Problems of the Pure Relational Approach

- Terms and substitution are complex. Duplicity of relational terms everywhere. 6 months of research just for unification.
- Difficult to implement. $A^+ = \mathcal{T}_\Sigma \cup \mathcal{T}_\Sigma^* \cup (\mathcal{T}_\Sigma^*)^* \cup \ldots$ is a hell of a data type.
- Renaming apart is difficult to model and understand. Crucial information is missing the number of variables currently in use. Combinatorial approach: bad for performance.
- Efficiency is difficult due to duplicity.
<table>
<thead>
<tr>
<th>Question</th>
<th>Allegory</th>
<th>RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the domain for the relations?</td>
<td>QRA</td>
<td>Single domain</td>
</tr>
<tr>
<td></td>
<td>Allegory</td>
<td>Σ ∪ Σ ∗ ∪ (Σ ∗ ∪ Σ ∗ ∪ . . . )</td>
</tr>
<tr>
<td>Signature?</td>
<td>QRA</td>
<td>New relations for every constant and term former.</td>
</tr>
<tr>
<td></td>
<td>Allegory</td>
<td>Freely adjoined arrows.</td>
</tr>
<tr>
<td>How are variables represented?</td>
<td>QRA</td>
<td>Untyped projections.</td>
</tr>
<tr>
<td></td>
<td>Allegory</td>
<td>We use categorical projections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In essence, we replace typed projections πₙ: N → A † ↔ A †. A small change with big implications.</td>
</tr>
</tbody>
</table>
Allegories versus RA

What is the domain for the relations?

QRA  A single domain $\mathcal{T}_\Sigma \cup \mathcal{T}_\Sigma^* \cup (\mathcal{T}_\Sigma^*)^* \cup \ldots$

Allegory  Types represent fixed-length sequences of terms.

Signature?

How are variables represented?

QRA  Untyped projections.

Allegory  We use categorical projections. In essence, we replace typed projections $\pi^N_i : N \rightarrow 1$ for untyped quasiprojections $P_i : A^\dagger \leftrightarrow A^\dagger$. A small change with big implications.
Allegories versus RA

What is the domain for the relations?

<table>
<thead>
<tr>
<th>QRA</th>
<th>A single domain $T_\Sigma \cup T_\Sigma^* \cup (T_\Sigma^<em>)^</em> \cup \ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allegory</td>
<td>Types represent fixed-length sequences of terms.</td>
</tr>
</tbody>
</table>

Signature?

<table>
<thead>
<tr>
<th>QRA</th>
<th>New relations for every constant and term former.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allegory</td>
<td>Freely adjoined arrows.</td>
</tr>
</tbody>
</table>

How are variables represented?

<table>
<thead>
<tr>
<th>QRA</th>
<th>Untyped projections.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allegory</td>
<td>Categorical projections. In essence, we replace typed projections $\pi^N_i : N \to 1$ for untyped quasiprojections $P^i : A^\dagger \leftrightarrow A^\dagger$. A small change with big implications.</td>
</tr>
</tbody>
</table>
### Allegories versus RA

#### What is the domain for the relations?

<table>
<thead>
<tr>
<th>QRA</th>
<th>A single domain $\mathcal{T}<em>\Sigma \cup \mathcal{T}</em>\Sigma^* \cup (\mathcal{T}_\Sigma^<em>)^</em> \cup \ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allegory</td>
<td>Types represent fixed-length sequences of terms.</td>
</tr>
</tbody>
</table>

#### Signature?

<table>
<thead>
<tr>
<th>QRA</th>
<th>New relations for every constant and term former.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allegory</td>
<td>Freely adjoined arrows.</td>
</tr>
</tbody>
</table>

#### How are variables represented?

<table>
<thead>
<tr>
<th>QRA</th>
<th>Untyped projections.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allegory</td>
<td>We use categorical projections.</td>
</tr>
</tbody>
</table>

In essence, we replace typed projections $\pi_i^N : N \rightarrow 1$ for untyped quasiprojections $P_i : A^\uparrow \leftrightarrow A^\uparrow$. A small change with big implications.
Categories of Syntax (Lawvere Categories)

Defining the Category

For a signature $\Sigma$, we define a Lawvere Category $C$:

- Objects are the natural numbers. Terminal object 0, the rest of the objects are powers of 1, $1 \times 1 \times 1 = 2 \times 1 = 1 \times 2 = 3$.
- For every constant $a \in T_\Sigma$, we freely adjoin an arrow $a : 0 \to 1$.
- For every function symbol $f \in T_\Sigma$ with arity $\alpha(f) = N$, we freely adjoin an arrow $f : N \to 1$.

Example

For instance, for $\Sigma = (\{o\}, \{s/1, +/2\})$, $C$ has all the terminal and product arrows plus $o : 0 \to 1$, $s : 1 \to 1$ and $+ : 2 \to 1$.

Initial Model

The initial model is a functor $C \to \text{Set}$ which preserves finite products and pullbacks. It maps the object 1 to $T_\Sigma$. 
Regular Categories, Tabular Relations

Regular Category
Category with products, pullbacks and certain exactness conditions.

Categories of Relations

\( f : C \to A \) and \( g : C \to B \) is a monic pair iff \( \langle f, g \rangle : C \to A \times B \) is monic, informally, a subset of \( A \times B \), thus, \( (f, g) \) represent a relation from \( A \) to \( B \):

\[
\begin{array}{cc}
\ & C \\
/ & ^f \ \\
A & \ & \ & \ & \ & \ & B \\
/ & _g \ \\
\ & B
\end{array}
\]
**Allegories and Distributive Allegories**

An (distributive) allegory is a category with added structure, such that if \( f, g : A \to B \) are arrows, \((f \cup g) \cap g : A \to B\) and \(f^\circ : B \to A\) are arrows and obey the appropriate relational laws. Typed version of relational algebras.

**Tabular Allegories**

An allegory is tabular if for each morphism \( R \) there is a pair of maps \( f, g \), such that \( R = f^\circ \circ g \). We say that \((f, g)\) tabulate \( R \). Regular categories are categories of maps for tabular allegories, thus they generate them. Diagrammatically:

```
       C
      / \ 
     f^\circ /  \ g
    /    \    
A   R   C   B
      \   / 
    \  /  
     \B
```
Regular Lawvere Categories

Pullbacks in Syntax Categories

- In \( \mathcal{C} \), arrows are freely added.
- No way of equalizing constants \( a, b : 0 \rightarrow 1 \).
- \( \mathcal{C} \) is not a regular category, it lacks pullbacks.
Regular Lawvere Categories

Pullbacks in Syntax Categories

- In $C$, arrows are freely added.
- No way of equalizing constants $a, b : 0 \to 1$.
- $C$ is not a regular category, it lacks pullbacks.

Regular Completion of $C$

- Adjoin an initial object $\bot$
- Freely adjoin the corresponding initial arrows $?_A : \bot \to A$ for every object $A$. Apply the quotient $?_A; f = ?_B$ for any arrow $f : A \to B$.
- Now every arrow can be made equal to another, $\bot; a = \bot; b$. 
Renaming Apart

In the case of a pullback, every term feeds from a different set of variables, so unification module renaming apart is guaranteed.

So each clause will be translated to feed from the same set of variables.
Σ-Allegories

Σ-Allegories are distributive allegories generated from a Regular Lawvere Category for the signature Σ and thus partially tabular. They are the target of our translation.
Term Translation

For a sequence of terms \( \vec{x} = \vec{t}[\vec{y}] \), \( K(\vec{t}[\vec{y}]) \) is the coreflexive relation:

\[
\begin{align*}
K(\vec{t}[\vec{y}]) & \quad \text{if } \vec{y} = \vec{y} \\
K(\vec{t}[\vec{y}]) & \quad \text{if } \vec{t} \neq \vec{y}
\end{align*}
\]

Registers

If we look at a completed clause:

\[
p(\vec{x}') \leftarrow \vec{x} = \vec{t}[\vec{y}], p_1(w_1(\vec{x})), \ldots, p_n(w_n(\vec{x})).
\]

it is clear that \( \vec{x} = x_1, \ldots, x_n \) plays the role of parameter registers. Names are eliminated by using \( \langle t_1, \ldots, t_n \rangle \).
Encoding terms and registers

Correspondence of concepts:

<table>
<thead>
<tr>
<th>Arrows (tabulations, $f$, $g$)</th>
<th>Arrays of terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projections ($\pi_i$)</td>
<td>Pointers</td>
</tr>
<tr>
<td>Domain of tabulations</td>
<td>Free (heap) variables ($Y_i$)</td>
</tr>
<tr>
<td>Target of tabulations</td>
<td>Registers ($X_i$)</td>
</tr>
<tr>
<td>Composition of tabulations ($f; g$)</td>
<td>Substitution</td>
</tr>
<tr>
<td>Intersection</td>
<td>Term Unification</td>
</tr>
</tbody>
</table>

Example (Term Storage in Registers)

\[ X1 = f(Y1, Y3) \]
\[ f_1 = \langle \pi_1, \pi_3 \rangle; f \]
\[ X2 = g(Y2, a) \]
\[ f_2 = \langle \pi_2, !_2; a \rangle; g \]
The Relational Step

Step 1: Local Storage

We previously translated the clause:

\[ p(\bar{x}') \leftarrow \bar{x} = \bar{t}[\bar{y}], p_1(w_1(\bar{x})), \ldots, p_n(w_n(\bar{x})). \]

to

\[ \bar{p} = K(\bar{t}) \cap W_1; \bar{p}_1; W_1^\circ \cap \cdots \cap W_n; \bar{p}_n; W_n^\circ \]

but now the number of arguments of \(|\bar{x}|, |\bar{x}'|\) may not be the same.
The Relational Step

Step 1: Local Storage

We previously translated the clause:

\[ p(\vec{x}') \leftarrow \vec{x} = \vec{t}[\vec{y}], p_1(w_1(\vec{x})), \ldots, p_n(w_n(\vec{x})). \]

to

\[ \overline{p} = K(\vec{t}) \cap W_1; \overline{p_1}; W_1^\circ \cap \cdots \cap W_n; \overline{p_n}; W_n \]

but now the number of arguments of \(|\vec{x}|, |\vec{x}'|\) may not be the same.

Step 2: Environments

We introduce environment creation (and its reciprocal, destruction) relations \( I_{MN} = \langle \pi_1, \ldots, \pi_m \rangle^\circ : M \rightarrow N \). Now the clause is translated to:

\[ I_{MN}; (K(\vec{t}) \cap W_1; \overline{p_1}; W_1^\circ \cap \cdots \cap W_n; \overline{p_n}; W_n^\circ); I_{MN}^\circ \]

\( M \) are parameters, \( N - M \) is the number of local variables.
Environment Management

Environments: \( I_{MN} \)

We introduce an environment creation (and its reciprocal, destruction) relation

\[
I_{MN} = \langle \pi_1, \ldots, \pi_m \rangle^\circ : M \to N.
\]

\((\pi_1^2)^\circ : 1 \to 2\) is the canonical “new” variable creation relation.

Example (Compile time optimization)

Let two registers in

\[
R = \langle \pi_1; f, \pi_1; g \rangle : 1 \to 2.
\]

Compute \( R^\circ; R; l_{23} \).
Intersection and Procedure Call

Coreflexive Arrows

If $R, S$ are coreflexive then $R \cap S = R; S$. This is highly convenient, and we may eliminate $\cap$ and simplify our machine.

Procedure Call

The previous translation is semantically correct translation, but $W_i; I_{NK}; \overline{p_i}; l^\circ_{NK}; W_i^\circ : N \to N$ is not in general a coreflexive relation, so we cannot apply $\cap$ elimination. We fix this using a correflexive version: $(id_{M-\alpha(p_i)} \times \overline{p_i}) N \to N$. Then, if $A_i = N - \alpha(p_i)$ the final translation is:

$$\overline{p} = I_{MN}; (K(\overline{t}); W_1; (id_{A_1} \times \overline{p_1}); W_1^\circ; \ldots; W_n; (id_{A_n} \times \overline{p_n}); W_n^\circ); l^\circ_{MN}$$
Example: Partial Evaluation

Translation of add

\[
\begin{align*}
\text{add}(x_1, x_2, x_3) & \leftarrow x_1 = o, x_2 = y_1, x_3 = y_1. \\
\text{add}(x_1, x_2, x_3) & \leftarrow x_1 = s(y_1), x_2 = y_2, x_3 = s(y_3), x_4 = y_1, x_5 = y_3, \\
& \text{add}(x_4, x_2, x_5).
\end{align*}
\]
Example: Partial Evaluation

<table>
<thead>
<tr>
<th>Translation of add</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \texttt{add}(x_1, x_2, x_3) ) &amp; ( \leftarrow x_1 = o, x_2 = y_1, x_3 = y_1. )</td>
</tr>
<tr>
<td>( \texttt{add}(x_1, x_2, x_3) ) &amp; ( \leftarrow x_1 = s(y_1), x_2 = y_2, x_3 = s(y_3), x_4 = y_1, x_5 = y_3, )</td>
</tr>
<tr>
<td>( \texttt{add} ) &amp; ( \leftarrow \texttt{add}(x_4, x_2, x_5). )</td>
</tr>
<tr>
<td>( \texttt{add} ) &amp; ( = \langle o, \pi_1, \pi_1 \rangle^\circ; \langle o, \pi_1, \pi_1 \rangle )</td>
</tr>
<tr>
<td>&amp; ( \cup \ l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \texttt{add}); W^\circ; l_{35}^\circ )</td>
</tr>
</tbody>
</table>
Example: Partial Evaluation

Translation of add

\[
\text{add}(x_1, x_2, x_3) \leftarrow x_1 = o, x_2 = y_1, x_3 = y_1.
\]
\[
\text{add}(x_1, x_2, x_3) \leftarrow x_1 = s(y_1), x_2 = y_2, x_3 = s(y_3), x_4 = y_1, x_5 = y_3,
\]
\[
\text{add}(x_4, x_2, x_5).
\]
\[
\text{add} = \langle o, \pi_1, \pi_1 \rangle^\circ; \langle o, \pi_1, \pi_1 \rangle
\]
\[
\cup l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}); W^\circ; l_{35}^\circ
\]

Unfold

Execute \text{add} without a query!

\[
\text{add} = \langle o, \pi_1, \pi_1 \rangle^\circ; \langle o, \pi_1, \pi_1 \rangle
\]
\[
\cup \langle os, \pi_1, \pi_1 s \rangle^\circ; \langle os, \pi_1, \pi_1 s \rangle
\]
\[
\cup l_{35}; \langle \pi_1 ss, \pi_2, \pi_3 ss, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}); W^\circ; l_{35}^\circ
\]

etc...
Composition of tabular relations: the core
Composition of tabular relations: the core

Composition captures unification, parameter passing, renaming apart, variable allocation and (a form of) garbage collection.
Composition of tabular relations: the core

Composition captures unification, parameter passing, renaming apart, variable allocation and (a form of) garbage collection.
Composition of tabular relations: the core

Composition captures unification, parameter passing, renaming apart, variable allocation and (a form of) garbage collection.
Composition of tabular relations: the core

Composition captures unification, parameter passing, renaming apart, variable allocation and (a form of) garbage collection.
The Pullback Algorithm

**Definition (Arrow Normalization)**

We write $\rightarrow^!_R$ for the associated normalizing relation based on $\rightarrow_R$:

\[
\begin{align*}
  h; \left< f, g \right> &\rightarrow_R \left< h; f, h; g \right> \\
  \left< f, g \right>; \pi_1 &\rightarrow_R f \\
  \left< f, g \right>; \pi_2 &\rightarrow_R g \\
  f; !_N &\rightarrow_R !_M \\
  f : M &\rightarrow N
\end{align*}
\]

**Definition (Starting Diagram)**

For a pullback problem, build the pre-starting diagram $\mathcal{P}$:

\[
\begin{array}{ccc}
  N \times N' & \xrightarrow{\pi_1; f} & M \\
  \downarrow & \downarrow & \downarrow \\
  \pi_2; g & \downarrow & \downarrow \\
  & M & \end{array}
\]
The Pullback Algorithm

The starting diagram is:

\[ N + N' \xrightarrow{id = \langle \pi_1, \ldots, \pi_{N+N'} \rangle} N + N' \xrightarrow{f'} \quad \xrightarrow{g'} \quad M \]

\( f' = \langle f_1, \ldots, f_M \rangle, \quad g' = \langle g_1, \ldots, g_M \rangle, \quad S = \{ f_1 \approx g_1, \ldots, f_M \approx g_M \} \). Initial state \(( S \mid \langle \pi_1, \ldots, \pi_{N+N'} \rangle )\). Proceed iteratively:

\[
\begin{align*}
!_M; a & \approx !_M; b \Rightarrow \text{Fail} \\
!_M; a & \approx h; f \Rightarrow \text{Fail} \\
g; f & \approx g'; f' \Rightarrow \text{Fail} \\
\pi_i & \approx \pi_j \Rightarrow ( S' \mid S(j, \pi_i, h) ) \\
\pi_i & \approx g; f \Rightarrow ( S' \mid S(i, g; f, h) ) \\
!_M; a & \approx \pi_i \Rightarrow ( S' \mid S(i, !_M; a, h) ) \\
!_M; a & \approx !_M; a \Rightarrow ( S' \mid h ) \\
g; f & \approx g'; f \Rightarrow ( \{ g_1 \approx g'_1 \} \cup \ldots \cup \{ g_n \approx g'_n \} \cup S' \mid h )
\end{align*}
\]
Specification of the machine

Diagram Rewriting

Basic diagrams: \((f \mid g), R_1 \cup \cdots \cup R_n\) and \((f \mid \langle g, [R]\rangle)\).

\[
\begin{align*}
(f \mid g); (f' \mid g') & \quad \xrightarrow{(h,h')} \quad (h; f \mid h'; g') \\
(f \mid \langle g_K, g_N \rangle); (id_K \times \overline{p_N}) & \quad \Rightarrow \quad (f \mid \langle g_K, [g_N; p_1]\rangle) \cup \\
& \quad \vdots \\
& \quad \cup \\
& \quad (f \mid \langle g_K, [g_N; p_n]\rangle) \\
(f \mid \langle g, [(g' \mid g')]\rangle) & \quad \Rightarrow \quad (f \mid \langle g, g\rangle) \\
(f \mid \langle g, [E]\rangle) & \quad \Rightarrow \quad (h; f \mid \langle h; g, [E']\rangle) \quad \text{iff } E \Rightarrow E' \\
R \cup S & \quad \Rightarrow \quad R' \cup S \quad \text{iff } R \Rightarrow R' \\
0 \cup S & \quad \Rightarrow \quad S
\end{align*}
\]
The Machine: An Example

A query $\text{add}(s(X), Y, Z)$ is translated to $\langle \pi_1 s, \pi_2, \pi_3 \rangle; \text{add}$:

$\langle \pi_1 s, \pi_2, \pi_3 \rangle; \text{add} \Rightarrow$
The Machine: An Example

A query $\text{add}(s(X), Y, Z)$ is translated to $\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{\text{add}}$:

$$
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{\text{add}}
$$

$$(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots$$

$\Rightarrow$

$\Rightarrow$
The Machine: An Example

A query \(\text{add}(s(X), Y, Z)\) is translated to \(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \text{add}\):

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \text{add} \\
\left( \langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle \text{o}, \pi_1, \pi_1 \rangle \right) \cup \ldots \\
0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}); W^\circ; l_{35}^\circ
\]
The Machine: An Example

A query \( add(s(X), Y, Z) \) is translated to \( \langle \pi_1 s, \pi_2, \pi_3 \rangle; add: \)

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; add
\]
\[
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots
\]
\[
0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times add); W^o; I_{35}
\]
\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times add); W^o; I_{35}
\]
A query $\text{add}(s(X), Y, Z)$ is translated to $\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{\text{add}}$:

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{\text{add}} \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots \\
0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^\circ; l_{35}^\circ \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^\circ; l_{35}^\circ \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^\circ; l_{35}^\circ \\
\Rightarrow \\
\Rightarrow \\
\Rightarrow \\
\Rightarrow \\
\Rightarrow \\
\Rightarrow \\
\Rightarrow
The Machine: An Example

A query \( add(s(X), Y, Z) \) is translated to \( \langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} : \)

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots \\
0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^o; l_{35} \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^o; l_{35} \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^o; l_{35} \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{add}); W^o; l_{35}
\]
The Machine: An Example

A query \( \text{add}(s(X), Y, Z) \) is translated to \( \langle \pi_1 s, \pi_2, \pi_3 \rangle; \text{add} \):

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \text{add} \quad \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots \\
0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}) ; W^\circ; l_{35}^\circ \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}) ; W^\circ; l_{35}^\circ \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle | \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}) ; W^\circ; l_{35}^\circ \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle | \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \text{add}) ; W^\circ; l_{35}^\circ \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle | \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle); (id_2 \times \text{add}) ; W^\circ; l_{35}^\circ
\]
The Machine: An Example

A query $add(s(X), Y, Z)$ is translated to $\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add}$:

$$
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots \\
0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^o; l_{35}^o \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^o; l_{35}^o \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle | \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^o; l_{35}^o \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle | \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{add}); W^o; l_{35}^o \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle | \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle); (id_2 \times \overline{add}); W^o; l_{35}^o \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle | \langle \pi_1 s, \pi_3 s, [\langle \pi_1, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle] \rangle; W^o; l_{35}^o \cup \ldots
$$
The Machine: An Example

A query $\text{add}(s(X), Y, Z)$ is translated to $\langle \pi_1 s, \pi_2, \pi_3 \rangle; \text{add}$:

$$
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \text{add}
\quad \Rightarrow
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle \cup \ldots
\quad \Rightarrow
0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35} \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}); W^o; l_{35}^o
\langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35} \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}); W^o; l_{35}^o
\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle \rangle \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}); W^o; l_{35}^o
\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \text{add}); W^o; l_{35}^o
\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle; (id_2 \times \text{add}); W^o; l_{35}^o
\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, \langle \pi_1, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle \rangle; W^o; l_{35}^o \cup \ldots
\langle Os, \pi_1, \pi_1 s \rangle \mid \langle Os, \pi_1 s, \langle o, \pi_1, \pi_1 \rangle \rangle; W^o; l_{35}^o \cup \ldots
$$
The Machine: An Example

A query $\text{add}(s(X), Y, Z)$ is translated to $\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{\text{add}}$:

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{\text{add}} \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots \\
0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle); (id_2 \times \overline{\text{add}}); W^o; l_{35}^o \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, [\langle \pi_1, \pi_2, \pi_3 \rangle; o, \pi_1, \pi_1 \rangle]; W^o; l_{35}^o \cup \ldots \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1 s, [\langle o, \pi_1, \pi_1 \rangle] \rangle); W^o; l_{35}^o \cup \ldots \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1 s, o, \pi_1, \pi_1 \rangle); W^o; l_{35}^o \cup \ldots 
\]
A query $add(s(X), Y, Z)$ is translated to $\langle \pi_1 s, \pi_2, \pi_3 \rangle$; $\overline{add}$:

$$\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add}$$

$$(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots$$

$0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}$; $\langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle$; $W; (id_2 \times \overline{add})$; $W^o; I_{35}$

$\langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}$; $\langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle$; $W; (id_2 \times \overline{add})$; $W^o; I_{35}$

$$(\langle \pi_1 s, \pi_2, \pi_3 \rangle | \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle$; $W; (id_2 \times \overline{add})$; $W^o; I_{35}$$

$$(\langle \pi_1 s, \pi_2, \pi_3 s \rangle | \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{add})$; $W^o; I_{35}$$

$$(\langle \pi_1 s, \pi_2, \pi_3 s \rangle | \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle); (id_2 \times \overline{add})$; $W^o; I_{35}$$

$$(\langle \pi_1 s, \pi_2, \pi_3 s \rangle | \langle \pi_1 s, \pi_3 s, \pi_1 \rangle); W; (id_2 \times \overline{add})$; $W^o; I_{35}$$

$$(\langle \pi_1 s, \pi_1, \pi_1 s \rangle | \langle \pi_1 s, \pi_1 s, \pi_1 \rangle); W^o; I_{35} \cup \ldots$$

$$(\langle os, \pi_1, \pi_1 s \rangle | \langle os, \pi_1 s, \pi_1 \rangle); W^o; I_{35} \cup \ldots$$

$$(\langle os, \pi_1, \pi_1 s \rangle | \langle os, \pi_1 s, o, \pi_1 \rangle); W^o; I_{35} \cup \ldots$$

$$(\langle os, \pi_1, \pi_1 s \rangle | \langle os, \pi_1, \pi_1 s, o, \pi_1 \rangle); I_{35} \cup \ldots$$

then $\langle os, \pi_1, \pi_1 s \rangle$ is translated back to the answer $X = o$, $Z = s(Y)$. 

EJGA (CRI-Mines)
The Machine: An Example

A query \(\text{add}(s(X), Y, Z)\) is translated to \(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{\text{add}}:\)

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{\text{add}}
\]

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle \cup \ldots
\]

\[
0 \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o
\]

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o
\]

\[
\langle \pi_1 s, \pi_2, \pi_3 \rangle; l_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o
\]

\[
\langle \pi_1 s, \pi_2, \pi_3 s \rangle; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o
\]

\[
\langle \pi_1 s, \pi_2, \pi_3 s \rangle; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o
\]

\[
\langle \pi_1 s, \pi_2, \pi_3 s \rangle; \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o
\]

\[
\langle \pi_1 s, \pi_2, \pi_3 s \rangle; \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle; W; (id_2 \times \overline{\text{add}}); W^o; l_{35}^o
\]

\[
\langle \pi_1 s, \pi_2, \pi_3 s \rangle; \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle; W^o; l_{35}^o \cup \ldots
\]

\[
\langle \text{os}, \pi_1, \pi_1 s \rangle; \langle \text{os}, \pi_1 s, [\langle o, \pi_1, \pi_1 \rangle] \rangle; W^o; l_{35}^o \cup \ldots
\]

\[
\langle \text{os}, \pi_1, \pi_1 s \rangle; \langle \text{os}, \pi_1 s, o, \pi_1, \pi_1 \rangle; W^o; l_{35}^o \cup \ldots
\]

\[
\langle \text{os}, \pi_1, \pi_1 s \rangle; \langle \text{os}, \pi_1 s, o, \pi_1, \pi_1 \rangle; l_{35}^o \cup \ldots
\]

\[
\langle \text{os}, \pi_1, \pi_1 s \rangle \cup \ldots
\]

then \(\langle \text{os}, \pi_1, \pi_1 s \rangle\) is translated back to the answer \(X = o, Z = s(Y)\).
Future work

Not in this talk:

- Extensions: Monads, types, functions.
- Diagrams.
- Relational Unification.

Future Work:

- Beyond logic logic programming? Other applications?
- Higher-order types.
- Coalgebraic derivations [Komendantskaya-Power2011]
- Full formalization down to the instruction level.
- Research algebraic optimization. \([R; (S \cup T) = R; S \cup R; T]\)
- New Coq formalization and compiler: at 50%.
Merci pour votre attention.

Questions?