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

Computation $\equiv$ Proof Search

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

Recall (Basic) Logic Programming

Computation $\equiv$ Proof Search

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

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

Recall (Basic) Logic Programming

Computation $\equiv$ Proof Search

Basics Program (Γ), 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* $\vec{a}$!

Recall (Basic) Logic Programming

Computation $\equiv$ Proof Search

Basics Program (Γ), 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

```
add(0, X, X).                         (∀X)
add(X+1, Y, Z+1) <- add(X, Y, Z).     (∀XYZ)
```

Recall (Basic) Logic Programming

Computation $\equiv$ Proof Search

Basics Program (Γ), 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

```
add(0, X, X).                        (∀X)
add(X+1, Y, Z+1) <- add(X, Y, Z).    (∀XYZ)
```

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

Recall (Basic) Logic Programming

Computation $\equiv$ Proof Search

Basics Program (Γ), 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

```
add(0, X, X).                            (∀X)
add(X+1, Y, Z+1) <- add(X, Y, Z).  (∀XYZ)
```

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

```
?- add(X, Y, Z).
{X = o, Z = Y ?} ;
{X = s(o), Z = s(Y) ? } [more]
```

Constraints: primitive class of formulas solved externally.

Combinatorial Logic Programming

What is our goal?

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

Combinatorial Logic Programming

proof search

What is our goal?

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

Combinatorial Logic Programming

proof search

What is our goal?

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

Example

$$nat(o) \leftarrow \top \qquad\qquad nat(s(X)) \leftarrow nat(X)$$

Combinatorial Logic Programming

proof search

What is our goal?

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

Example

$$nat(o) \leftarrow \top \qquad \forall X\, .nat(s(X)) \leftarrow nat(X)$$

Combinatorial Logic Programming

proof search

What is our goal?

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

Example

$$nat(o) \leftarrow \top \qquad \forall\, X\, .nat(s(X)) \leftarrow nat(X)$$

Point-free means no variables:

$$\overline{nat} = \{o\} \cup s \cdot \overline{nat}$$

Combinatorial Logic Programming

What is our goal?

proof search

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

Example

$$nat(o) \leftarrow \top \qquad\qquad \forall X\,.nat(s(X)) \leftarrow nat(X)$$

Point-free means no variables:

$$\overline{nat} = \{o, s(o)\} \cup s \cdot s \cdot \overline{nat}$$

and equational program reasoning

Combinatorial Logic Programming

proof search

What is our goal?

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

Example

$$nat(o) \leftarrow \top \qquad \forall X\, .nat(s(X)) \leftarrow nat(X)$$

Point-free means no variables:

$$\overline{nat} = \{o, s(o), s(s(o))\} \cup s \cdot s \cdot s \cdot \overline{nat}$$

and equational program reasoning

Combinatorial Logic Programming

proof search

What is our goal?

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

Example

$$nat(o) \leftarrow \top \qquad\qquad \forall\, X\, .nat(s(X)) \leftarrow nat(X)$$

Point-free means no variables:

$$\overline{nat} = \{o, s(o), s(s(o))\} \cup s \cdot s \cdot s \cdot \overline{nat}$$

and equational program reasoning
and proof search: $s(s(o)) \cap \overline{nat}$

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.

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) :- ∃ Y, X = s(Y)
? p(X).
```

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) :- ∃ 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)).                     %% p(X) :- ∃ 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)).                %% p(X) :- ∃ 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.
- 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) :- ∃ 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.
- Quantifiers capture this, but use variables! Also, operational reasoning involves tricky renaming apart, etc...
- Use an algebraic theory of logic and quantification!

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.

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 $\wedge$ 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 $\wedge$ and $\cup$ as $\vee$; 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$^{\text{th}}$ century, and further developed by Tarski and his students in mid 20$^{\text{th}}$ century.

Relations and Logic

$\cap$ as $\wedge$ and $\cup$ as $\vee$; 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").

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):
  - $\mathbf{QRA}_\Sigma$.
  - Σ-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):
  - $\mathbf{QRA}_\Sigma$.
  - Σ-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

$$\begin{array}{lcl} \text{Formula} & = & \text{Relation} \\ \text{Proof} & = & \text{Equational Derivation} \end{array}$$

The Plan

Compile, interpret and execute CLP to a QRA:

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

Program	=	Formula	=	Relation
Computation	=	Proof Search	=	Equational Derivation

The Plan

Compile, interpret and execute CLP to a QRA:

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

Program	$=$	Relation
Computation	$=$	Equational Derivation

The Relational Theory

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

The Relational Theory

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

Generate the relational language:

$$\mathsf{R}_{\mathcal{C}} = \{(a,a) \mid a \in \mathcal{C}_\Sigma\} \quad \mathsf{R}_{\mathcal{F}} = \{\mathsf{R}_f \mid f \in \mathcal{F}_\Sigma, \}$$
$$\mathsf{R}_{\mathcal{CP}} = \{\mathsf{r} \mid r \in \mathcal{CP}_\Sigma\} \qquad \mathsf{R}_{\mathcal{P}} = \{\overline{p} \mid p \in \mathcal{P}_\Sigma\}$$
$$\mathsf{R}_{atom} ::= \mathsf{R}_{\mathcal{C}} \mid \mathsf{R}_{\mathcal{F}} \mid \mathsf{R}_{\mathcal{CP}} \mid \mathsf{R}_{\mathcal{P}} \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl$$
$$\mathsf{R}_\Sigma ::= \mathsf{R}_{atom} \mid \mathsf{R}_\Sigma{}^\circ \mid \mathsf{R}_\Sigma \cup \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \cap \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \mathsf{R}_\Sigma$$

The Relational Theory

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

Generate the relational language:

$$
\begin{array}{rclcrcl}
\mathsf{R}_{\mathcal{C}} & = & \{(a,a) \mid a \in \mathcal{C}_\Sigma\} & & \mathsf{R}_{\mathcal{F}} & = & \{\mathsf{R}_f \mid f \in \mathcal{F}_\Sigma,\} \\
\mathsf{R}_{\mathcal{CP}} & = & \{\mathsf{r} \mid r \in \mathcal{CP}_\Sigma\} & & \mathsf{R}_{\mathcal{P}} & = & \{\overline{p} \mid p \in \mathcal{P}_\Sigma\}
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{R}_{atom} & ::= & \mathsf{R}_{\mathcal{C}} \mid \mathsf{R}_{\mathcal{F}} \mid \mathsf{R}_{\mathcal{CP}} \mid \mathsf{R}_{\mathcal{P}} \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl \\
\mathsf{R}_\Sigma & ::= & \mathsf{R}_{atom} \mid \mathsf{R}_\Sigma{}^\circ \mid \mathsf{R}_\Sigma \cup \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \cap \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \mathsf{R}_\Sigma
\end{array}
$$

Interpretation

$[\![\cdot]\!] : \mathsf{R}_\Sigma \to \mathcal{P}(\mathcal{D}^\dagger \times \mathcal{D}^\dagger)$, where $\mathcal{D}^\dagger = \cup_n \mathcal{D}^n$.

The Relational Theory

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

Generate the relational language:

$$
\begin{array}{lcl}
\mathsf{R}_{\mathcal{C}} = \{(a, a) \mid a \in \mathcal{C}_\Sigma\} & & \mathsf{R}_{\mathcal{F}} = \{\mathsf{R}_f \mid f \in \mathcal{F}_\Sigma, \} \\
\mathsf{R}_{\mathcal{CP}} = \{\mathsf{r} \mid r \in \mathcal{CP}_\Sigma\} & & \mathsf{R}_{\mathcal{P}} = \{\overline{p} \mid p \in \mathcal{P}_\Sigma\}
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{R}_{atom} & ::= & \mathsf{R}_{\mathcal{C}} \mid \mathsf{R}_{\mathcal{F}} \mid \mathsf{R}_{\mathcal{CP}} \mid \mathsf{R}_{\mathcal{P}} \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl \\
\mathsf{R}_\Sigma & ::= & \mathsf{R}_{atom} \mid \mathsf{R}_\Sigma{}^\circ \mid \mathsf{R}_\Sigma \cup \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \cap \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \mathsf{R}_\Sigma
\end{array}
$$

Interpretation

$[\![\cdot]\!] : \mathsf{R}_\Sigma \to \mathcal{P}(\mathcal{D}^\dagger \times \mathcal{D}^\dagger)$, where $\mathcal{D}^\dagger = \cup_n \mathcal{D}^n$.

Example

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

The Relational Theory

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

Generate the relational language:

$$
\begin{array}{rcl}
\mathsf{R}_{\mathcal{C}} = \{(a,a) \mid a \in \mathcal{C}_\Sigma\} & \mathsf{R}_{\mathcal{F}} & = \{\mathsf{R}_f \mid f \in \mathcal{F}_\Sigma, \} \\
\mathsf{R}_{\mathcal{CP}} \;=\; \{\mathsf{r} \mid r \in \mathcal{CP}_\Sigma\} & \mathsf{R}_{\mathcal{P}} & = \{\overline{p} \mid p \in \mathcal{P}_\Sigma\}
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{R}_{atom} & ::= & \mathsf{R}_{\mathcal{C}} \mid \mathsf{R}_{\mathcal{F}} \mid \mathsf{R}_{\mathcal{CP}} \mid \mathsf{R}_{\mathcal{P}} \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid \color{red}{hd} \mid \color{red}{tl} \\
\mathsf{R}_\Sigma & ::= & \mathsf{R}_{atom} \mid \mathsf{R}_\Sigma{}^\circ \mid \mathsf{R}_\Sigma \cup \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \cap \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \mathsf{R}_\Sigma
\end{array}
$$

Interpretation

$[\![\cdot]\!] : \mathsf{R}_\Sigma \to \mathcal{P}(\mathcal{D}^\dagger \times \mathcal{D}^\dagger)$, where $\mathcal{D}^\dagger = \cup_n \mathcal{D}^n$.

Example

$$
\begin{array}{l}
[\![\leq]\!] = \{(\langle m,n\rangle\vec{u}, \langle m,n\rangle\vec{u}') \mid m \leq n; m,n \in \mathbb{R}, \vec{u}, \vec{u}' \in \mathcal{D}^\dagger\} \\
[\![\overline{add}]\!] = \{(\langle m,n,o\rangle\vec{u}, \langle m,n,o\rangle\vec{u}') \mid m+n=o; m,n,o \in \mathbb{N}, \vec{u}, \vec{u}' \in \mathcal{D}^\dagger\}
\end{array}
$$

Translation Overview: Helpers

Projections and Permutations

P_i is the relation projecting the i-th component of a vector; given a permutation π, W_π 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 π, W_π 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.

Translation Overview: Helpers

Projections and Permutations

P_i is the relation projecting the i-th component of a vector; given a permutation π, W_π 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 π, W_π 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!

$$Q_i; \quad \underbrace{R \cdots S}_{i-\text{private for} R \cdots S} \quad ; Q_i$$

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

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 *ground* instances and instantiations for $\vec{x}$:

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

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 *ground* instances and instantiations for $\vec{x}$:

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

Formally:

$$K(t) : \mathcal{T}_\Sigma(\mathcal{X}) \to \mathrm{R}_\Sigma = \begin{cases} (c, c)\mathbf{1} & \text{if } t \equiv c \\ P_i^\circ & \text{if } t \equiv x_i \\ \bigcap_{i \le n} f_i^r K(t_i) & \text{if } t \equiv f(t_1, \ldots, t_n) \end{cases}$$

Translation Overview [1/3]: Terms

Key Idea

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

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

Formally:

$$K(t) : \mathcal{T}_\Sigma(\mathcal{X}) \to \mathrm{R}_\Sigma = \begin{cases} (c, c)\mathbf{1} & \text{if } t \equiv c \\ P_i^\circ & \text{if } t \equiv x_i \\ \bigcap_{i \le n} f_i^r K(t_i) & \text{if } t \equiv f(t_1, \ldots, t_n) \end{cases}$$

Example

$$K(f(x_1, g(x_2, a, h(x_1)))) = f_1^2; P_1^\circ \cap f_1^2; (g_1^3; P_2^\circ \cap g_1^2; (a, a); \mathbf{1} \cap g_3^3; h; P_1^\circ)$$

Translation Overview [2/3]: Constraints

Key Idea

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

Translation Overview [2/3]: Constraints

Key Idea

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

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

Translation Overview [2/3]: Constraints

Key Idea

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

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

Formally:

$$\dot{K}(t) : \mathcal{L}_{\mathcal{D}} \to \mathrm{R}_{\Sigma} = \begin{cases} K(\vec{t})^{\circ}; \mathrm{p}; K(\vec{t}) & \text{if } \varphi \equiv p(\vec{t}) \\ \dot{K}(\varphi) \cap \dot{K}(\theta) & \text{if } \varphi \equiv \varphi \wedge \theta \\ Q_i; \dot{K}(\varphi); Q_i & \text{if } \varphi \equiv \exists x_i.\ \varphi \end{cases}$$

Translation Overview [2/3]: Constraints

Key Idea

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

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

Formally:

$$\dot{K}(t) : \mathcal{L}_{\mathcal{D}} \to \mathrm{R}_{\Sigma} = \begin{cases} K(\vec{t})^{\circ}; \mathrm{p}; K(\vec{t}) & \text{if } \varphi \equiv p(\vec{t}) \\ \dot{K}(\varphi) \cap \dot{K}(\theta) & \text{if } \varphi \equiv \varphi \wedge \theta \\ Q_i; \dot{K}(\varphi); Q_i & \text{if } \varphi \equiv \exists x_i.\ \varphi \end{cases}$$

Example

$$\dot{K}(\exists x_1 x_2 . s(x_1) \leq x_2) = Q_1 Q_2; (P_1^{\circ}; \mathrm{s}^{\circ}; P_1 \cap P_2^{\circ}; P_2); \leq; (P_1; \mathrm{s}; P_1^{\circ} \cap P_2; P_2^{\circ}); Q_1 Q_2$$

Translation Overview [3/3]: Predicates

Key Idea

Defined predicates p are translated to equations $\overline{p} \mathrel{\mathring{=}} R$.

Translation Overview [3/3]: Predicates

Key Idea

Defined predicates p are translated to equations $\overline{p} \mathrel{\mathring{=}} R$.

Theorem (Adequacy)

$$(\vec{a}\vec{u}, \vec{a}\vec{u}') \in [\![\overline{p}]\!]^{\mathcal{D}^{\dagger}} \iff p(\vec{a}) \in T_P^{\omega}$$

Translation Overview [3/3]: Predicates

Key Idea

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

Theorem (Adequacy)

$$(\vec{a}\vec{u}, \vec{a}\vec{u}') \in [\![\overline{p}]\!]^{\mathcal{D}^\dagger} \iff p(\vec{a}) \in T_P^\omega$$

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 $\overline{p} \doteq R$.

Theorem (Adequacy)

$$(\vec{a}\vec{u}, \vec{a}\vec{u}') \in [\![\overline{p}]\!]^{\mathcal{D}^\dagger} \iff p(\vec{a}) \in T_P^\omega$$

The Procedure

1. Purify clause' heads, canonical renaming, Clark completion.
2. The Relational Step!

$$\begin{array}{lcl} add(x_1, x_2, x_3) & \leftarrow & x_1 = o, x_2 = x_3. \\ add(x_1, x_2, x_3) & \leftarrow & x_1 = s(x_4), x_3 = s(x_5), add(x_4, x_2, x_5). \end{array}$$

Translation Overview [3/3]: Predicates

Key Idea

Defined predicates p are translated to equations $\overline{p} \mathrel{\mathring{=}} R$.

Theorem (Adequacy)

$$(\vec{a}\vec{u}, \vec{a}\vec{u}') \in [\![\overline{p}]\!]^{\mathcal{D}^\dagger} \iff p(\vec{a}) \in T_P^\omega$$

The Procedure

1. Purify clause' heads, canonical renaming, Clark completion.
2. The Relational Step!

$$\begin{array}{lcl}
add(x_1, x_2, x_3) & \leftarrow & x_1 = o, x_2 = x_3. \\
add(x_1, x_2, x_3) & \leftarrow & x_1 = s(x_4), x_3 = s(x_5), add(x_4, x_2, x_5). \\
\overline{add} & = & \dot{K}(x_1 = o \wedge x_2 = x_3) \\
 & \cup & I_3; \dot{K}(x_1 = s(x_4) \wedge x_3 = s(x_5)); W; \overline{add}; W^\circ; I_3
\end{array}$$

Program Execution

Relations and Computation

$$r \wedge (s \vee t) \leftrightarrow r \wedge s \vee r \wedge t \qquad R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$$

Program Execution

Relations and Computation

$r \wedge (s \vee t) \leftrightarrow r \wedge s \vee r \wedge t$	$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$
Computation rule	$R \cap (S \cup T) \mapsto (R \cap S) \cup (R \cap T)$

Program Execution

Relations and Computation

$r \wedge (s \vee t) \leftrightarrow r \wedge s \vee r \wedge t$	$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$
Computation rule	$R \cap (S \cup T) \mapsto (R \cap S) \cup (R \cap T)$
Cut rule	$S \cup T \supseteq S \qquad S \cup T \mapsto S$

Program Execution

Relations and Computation

$r \wedge (s \vee t) \leftrightarrow r \wedge s \vee r \wedge t$	$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$
Computation rule	$R \cap (S \cup T) \mapsto (R \cap S) \cup (R \cap T)$
Cut rule	$S \cup T \supseteq S \qquad S \cup T \mapsto S$

Queries

For executing a query, we just intersect and rewrite, for instance for $add(o, o, o)$:

$$\dot{K}(x_1 = x_2 = x_3 = o) \cap \overline{add}$$

Rewriting and the Modular Law

Some Sample Rules

$$\mathbf{0} \cup R \xrightarrow{P} R \qquad 0 \cap R \xrightarrow{P} \mathbf{0} \qquad (R \cup S) \cap T \xrightarrow{P} (R \cap T) \cup (S \cap T)$$

Meta Rules

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

$$\dot{K}(\psi_1) \cap \dot{K}(\psi_2) \rightarrow \dot{K}(\psi_1 \wedge \psi_2)$$

Procedure Call: The Modular Law

$$\dot{K}(\psi) \cap I_m(R) \xrightarrow{P} I_m(I_m(\dot{K}(\psi)) \cap R) \cap \dot{K}(\psi)$$

We hide variables in ψ that may be in conflict with variables in R, but we need to “unhide” them later.

Example Execution

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

1 $\quad I_3 \dot{K}(o, s(o), x_3) I_3 \cap \overline{add} \qquad\qquad \rightarrow$

Example Execution

Query: **add**`(o,s(o),X)`

$$
\begin{array}{lll}
1 & I_3\dot{K}(o,s(o),x_3)I_3 \cap \overline{add} & \rightarrow \\
2 & I_3\dot{K}(o,s(o),x_3)I_3 \cap (\dot{K}(o,x_2,x_2) \cup & \\
 & (I_3[\dot{K}(s(x_4),x_2,s(x_5),x_4,x_5) \;\cap\; W\,\overline{add}\,W^{\circ}]I_3)) & \rightarrow
\end{array}
$$

Example Execution

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

$$
\begin{array}{lll}
1 & I_3 \dot{K}(o, s(o), x_3) I_3 \cap \overline{add} & \rightarrow \\
2 & I_3 \dot{K}(o, s(o), x_3) I_3 \cap (\dot{K}(o, x_2, x_2) \cup & \\
 & (I_3 [\dot{K}(s(x_4), x_2, s(x_5), x_4, x_5) \ \cap \ W \,\overline{add}\, W^\circ] I_3)) & \rightarrow \\
3 & I_3 [\dot{K}(o, s(o), x_3) \cap \dot{K}(o, x_2, x_2)] I_3 \cup & \\
 & I_3 (\dot{K}(o, s(o), x_3) \cap \dot{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap & \\
 & W \,\overline{add}\, W^\circ) I_3 & \rightarrow
\end{array}
$$

Example Execution

Query: **add**`(o,s(o),X)`

$$
\begin{array}{lll}
1 & I_3 \dot{K}(o, s(o), x_3) I_3 \cap \overline{add} & \rightarrow \\
2 & I_3 \dot{K}(o, s(o), x_3) I_3 \cap (\dot{K}(o, x_2, x_2) \cup & \\
 & (I_3[\dot{K}(s(x_4), x_2, s(x_5), x_4, x_5) \ \cap \ W \,\overline{add}\, W^\circ] I_3)) & \rightarrow \\
3 & I_3[\dot{K}(o, s(o), x_3) \cap \dot{K}(o, x_2, x_2)] I_3 \cup & \\
 & I_3(\dot{K}(o, s(o), x_3) \cap \dot{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap & \\
 & W \,\overline{add}\, W^\circ) I_3 & \rightarrow \\
4 & \dot{K}(o, s(o), s(o)) \cup \color{red}{I_3[\mathbf{0} \cap W \,\overline{add}\, W^\circ] I_3} & \rightarrow
\end{array}
$$

Example Execution

Query: **add**`(o,s(o),X)`

$$
\begin{array}{lll}
1 & I_3 \dot{K}(o, s(o), x_3) I_3 \cap \overline{add} & \rightarrow \\
2 & I_3 \dot{K}(o, s(o), x_3) I_3 \cap (\dot{K}(o, x_2, x_2) \cup & \\
 & (I_3[\dot{K}(s(x_4), x_2, s(x_5), x_4, x_5) \ \cap \ W \, \overline{add} \, W^\circ] I_3)) & \rightarrow \\
3 & I_3[\dot{K}(o, s(o), x_3) \cap \dot{K}(o, x_2, x_2)] I_3 \cup & \\
 & I_3(\dot{K}(o, s(o), x_3) \cap \dot{K}(s(x_4), x_2, s(x_5), x_4, x_5) \cap & \\
 & W \, \overline{add} \, W^\circ) I_3 & \rightarrow \\
4 & \dot{K}(o, s(o), s(o)) \cup I_3[\mathbf{0} \cap W \, \overline{add} \, W^\circ] I_3 & \rightarrow \\
5 & \dot{K}(o, s(o), s(o)) &
\end{array}
$$

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. $A^{\dagger} = \mathcal{T}_{\Sigma} \cup \mathcal{T}_{\Sigma}^{*} \cup (\mathcal{T}_{\Sigma}^{*})^{*} \cup \ldots$ 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.

Allegories versus RA

What is the domain for the relations?

Signature?

How are variables represented?

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?

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?

QRA New relations for every constant and term former.

Allegory Freely adjoined arrows.

How are variables represented?

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?

QRA New relations for every constant and term former.

Allegory Freely adjoined arrows.

How are variables represented?

QRA Untyped projections.

Allegory We use categorical projections.

In essence, we replace typed projections $\pi_i^N : N \to 1$ for untyped quasiprojections $P_i : A^\dagger \leftrightarrow A^\dagger$. A small change with big implications.

Categories of Syntax (Lawvere Categories)

Defining the Category

For a signature Σ, we define a Lawvere Category $\mathcal{C}$:

- Objects are the natural numbers. Terminal object 0, the rest of the objects are powers of 1, $1x1x1 = 2x1 = 1x2 = 3$.
  - For every constant $a \in \mathcal{T}_\Sigma$, we freely adjoin an arrow $a : 0 \to 1$.
  - For every function symbol $f \in \mathcal{T}_\Sigma$ with arity $\alpha(f) = N$, we freely adjoin an arrow $f : N \to 1$.

Example

For instance, for $\Sigma = (\{o\}, \{s/1, +/2\})$, $\mathcal{C}$ 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 $\mathcal{C} \to \mathit{Set}$ which preserves finite products and pullbacks. It maps the object 1 to $\mathcal{T}_\Sigma$.

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

$$\begin{tikzcd} & C \arrow[dl, "f"'] \arrow[dr, "g"] & \\ A & & B \end{tikzcd}$$

Allegories

Allegories and Distributive Allegories

An (distributive) allegory is a category with added structure, such that if $f, g : A \to B$ are arrows, $(f \cup g)$ $f \cap g : A \to B$ 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 $R = f^\circ ; g$. We say that (f, g) tabulate R. Regular categories are categories of maps for tabular allegories, thus they generate them. Diagrammatically:

$$
\begin{tikzcd}
 & C \arrow[dr, "g"] & \\
A \arrow[ur, "f^\circ"] \arrow[rr, leftrightarrow, "R"] & & B
\end{tikzcd}
\qquad
\begin{tikzcd}
 & C \arrow[dl, "f"'] \arrow[dr, "g"] & \\
A \arrow[rr, leftrightarrow, "R"] & & B
\end{tikzcd}
$$

Regular Lawvere Categories

Pullbacks in Syntax Categories

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

Regular Lawvere Categories

Pullbacks in Syntax Categories

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

Regular Completion of $\mathcal{C}$

- Adjoin an initial object $\perp$
- Freely adjoin the corresponding initial arrows $?_A : \perp \to A$ 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, $\perp; a = \perp; b$.

Regular Lawvere Categories: Examples

Renaming Apart

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

$$
\begin{tikzcd}
2 \arrow[d, "\pi_1"'] & 2 \arrow[l, "id"'] \arrow[d, "id"] \\
1 & 2 \arrow[l, "\pi_1"]
\end{tikzcd}
\qquad
\begin{tikzcd}
2 \arrow[d, "\pi_1"'] & 3 \arrow[l, "\langle \pi_1, \pi_2 \rangle"'] \arrow[d, "\langle \pi_1, \pi_3 \rangle"] \arrow[dl, phantom, "\lrcorner", very near start] \\
1 & 2 \arrow[l, "\pi_1"]
\end{tikzcd}
$$

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

Σ-Allegories

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:

$$\begin{tikzcd} & {|\vec{y}|} \arrow[dl, "K_{\vec{y}}(\vec{t})"'] \arrow[dr, "K_{\vec{y}}(\vec{t})"] & \\ {|\vec{x}|} \arrow[rr, leftrightarrow, "K(\vec{t}[\vec{y}])"] & & {|\vec{x}|} \end{tikzcd}$$

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:

Arrows (tabulations, f, g)	Arrays of terms
Projections (π_i)	Pointers
Domain of tabulations	Free (heap) variables (Yi)
Target of tabulations	Registers (Xi)
Composition of tabulations ($f; g$)	Substitution
Intersection	Term Unification

Example (Term Storage in Registers)

$$X1 = f(Y1, Y3)$$
$$f_1 = \langle \pi_1, \pi_3 \rangle; f$$
$$X2 = g(Y2, a)$$
$$f_2 = \langle \pi_2, !_2; a \rangle; g$$

$$\begin{tikzcd}
 & 3 \arrow[dl, "\langle f_1, f_2 \rangle"'] \arrow[dr, "\langle f_1, f_2 \rangle"] & \\
2 \arrow[rr, dash, "\langle f_1, f_2 \rangle^\circ ; \langle f_1, f_2 \rangle"] & & 2
\end{tikzcd}$$

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

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

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

The Relational Step

Step 1: Local Storage

We previously translated the clause:

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

to

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

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 $I_{MN} = \langle \pi_1, \dots, \pi_m \rangle^\circ : M \to N$. 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.

$$
\begin{tikzcd}
 & 3 \arrow[dl, "\langle \pi_1, \pi_2 \rangle"'] \arrow[dr, mapsto, "id_3"] & \\
2 \arrow[rr, leftrightarrow, "I_{23}"] & & 3
\end{tikzcd}
$$

Example (Compile time optimization)

Let two registers in $R = \langle \pi_1; f, \pi_1; g \rangle : 1 \to 2$.
Compute $R^\circ; R; I_{23}$.

$$
\begin{tikzcd}
 & 2 \arrow[dl, "\langle \pi_1; f, \pi_1; g \rangle"'] \arrow[dr, mapsto, "\langle \pi_1; f, \pi_1; g, \pi_2 \rangle"] & \\
2 \arrow[rr, leftrightarrow, "R; R^\circ; I_{23}"] & & 3
\end{tikzcd}
$$

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 $W_i; I_{NK}; \overline{p_i}; I^\circ_{NK}; W^\circ_i : N \to N$ 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:

$$\overline{p} = I_{MN}; (K(\vec{t}); W_1; (id_{A_1} \times \overline{p_1}); W^\circ_1; \ldots; W_n; (id_{A_n} \times \overline{p_n}); W^\circ_n); I^\circ_{MN}$$

Example: Partial Evaluation

Translation of add

$$
\begin{array}{lcl}
add(x_1, x_2, x_3) & \leftarrow & x_1 = o, x_2 = y_1, x_3 = y_1. \\
add(x_1, x_2, x_3) & \leftarrow & x_1 = s(y_1), x_2 = y_2, x_3 = s(y_3), x_4 = y_1, x_5 = y_3, \\
 & & add(x_4, x_2, x_5).
\end{array}
$$

Example: Partial Evaluation

Translation of add

$$
\begin{array}{lcl}
add(x_1, x_2, x_3) & \leftarrow & x_1 = o, x_2 = y_1, x_3 = y_1. \\
add(x_1, x_2, x_3) & \leftarrow & x_1 = s(y_1), x_2 = y_2, x_3 = s(y_3), x_4 = y_1, x_5 = y_3, \\
 & & add(x_4, x_2, x_5). \\
\overline{add} & = & \langle o, \pi_1, \pi_1 \rangle^\circ ; \langle o, \pi_1, \pi_1 \rangle \\
 & \cup & I_{35} ; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle ; W ; (id_2 \times \overline{add}) ; W^\circ ; I_{35}^\circ
\end{array}
$$

Example: Partial Evaluation

Translation of add

$$
\begin{array}{lcl}
add(x_1, x_2, x_3) & \leftarrow & x_1 = o, x_2 = y_1, x_3 = y_1. \\
add(x_1, x_2, x_3) & \leftarrow & x_1 = s(y_1), x_2 = y_2, x_3 = s(y_3), x_4 = y_1, x_5 = y_3, \\
 & & add(x_4, x_2, x_5). \\
\overline{add} & = & \langle o, \pi_1, \pi_1 \rangle^\circ; \langle o, \pi_1, \pi_1 \rangle \\
 & \cup & I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ
\end{array}
$$

Unfold

Execute $\overline{add}$ without a query!

$$
\begin{array}{lcl}
\overline{add} & = & \langle o, \pi_1, \pi_1 \rangle^\circ; \langle o, \pi_1, \pi_1 \rangle \\
 & \cup & \langle os, \pi_1, \pi_1 s \rangle^\circ; \langle os, \pi_1, \pi_1 s \rangle \\
 & \cup & I_{35}; \langle \pi_1 ss, \pi_2, \pi_3 ss, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ
\end{array}
$$

etc...

Composition of tabular relations: the core

$$\Rightarrow$$

$$\begin{tikzcd}
 & M \arrow[dl, "f"'] \arrow[dr, "g"] & & N \arrow[dl, "h"'] \arrow[dr, "i"] & \\
A & & B & & C
\end{tikzcd}$$

Composition of tabular relations: the core

$$
\begin{tikzcd}
 & & P \arrow[dl] \arrow[dr] \arrow[d, phantom, "\lrcorner" rotate=-45, very near start] & & \\
 & M \arrow[dl, "f"'] \arrow[dr, "g"] & & N \arrow[dl, "h"'] \arrow[dr, "i"] & \\
A & & B & & C
\end{tikzcd}
\qquad \Rightarrow
$$

Composition of tabular relations: the core

$$
\begin{tikzcd}
& & D \arrow[ddl, "u"'] \arrow[ddr, "v"] & & \\
& & P \arrow[u, tail] \arrow[dl] \arrow[dr] \arrow[dd, phantom, very near start, "\lrcorner", rotate=45] & & \\
& M \arrow[dl, "f"'] \arrow[dr, "g"] & & N \arrow[dl, "h"'] \arrow[dr, "i"] & \\
A & & B & & C
\end{tikzcd}
\qquad \Rightarrow
$$

Composition of tabular relations: the core

$$
\begin{tikzcd}
 & & D \arrow[ddl, "u"'] \arrow[ddr, "v"] & & \\
 & & P \arrow[u, tail] \arrow[dl] \arrow[dr] \arrow[d, phantom, "\vee"] & & \\
 & M \arrow[dl, "f"'] \arrow[dr, "g"] & {} & N \arrow[dl, "h"'] \arrow[dr, "i"] & \\
A & & B & & C
\end{tikzcd}
\quad \Rightarrow \quad
\begin{tikzcd}
 & D \arrow[dl, "u;f"'] \arrow[dr, "v;i"] & \\
A & & C
\end{tikzcd}
$$

Composition of tabular relations: the core

$$
\begin{tikzcd}
 & & D \arrow[dl, "u"'] \arrow[dr, "v"] & & \\
 & & P \arrow[u, tail] \arrow[dl] \arrow[dr] \arrow[d, phantom, "\vee"] & & \\
 & M \arrow[dl, "f"'] \arrow[dr, "g"] & {} & N \arrow[dl, "h"'] \arrow[dr, "i"] & \\
A & & B & & C
\end{tikzcd}
\quad \Rightarrow \quad
\begin{tikzcd}
 & D \arrow[dl, "u;f"'] \arrow[dr, "v;i"] & \\
A & & C
\end{tikzcd}
$$

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 $\rightarrow_R$:

$$
\begin{array}{lll}
h; \langle f, g\rangle & \rightarrow_R & \langle h; f, h; g\rangle \\
\langle f, g\rangle; \pi_1 & \rightarrow_R & f \\
\langle f, g\rangle; \pi_2 & \rightarrow_R & g \\
f; !_N & \rightarrow_R & !_M \qquad f : M \rightarrow N
\end{array}
$$

Definition (Starting Diagram)

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

$$
\begin{tikzcd}
N \times N' \arrow[r, shift left, "\pi_1 ; f"] \arrow[r, shift right, "\pi_2 ; g"'] & M
\end{tikzcd}
$$

The Pullback Algorithm

The starting diagram is:

$$
\begin{tikzcd}
N + N' \arrow[r, "id = \langle \pi_1, \ldots, \pi_{N+N'} \rangle"] & N + N' \arrow[r, shift left=2, "f'"] \arrow[r, shift right=2, "g'"'] & M
\end{tikzcd}
$$

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

$$
\begin{array}{lcllll}
!_M; a & \approx & !_M; b & \Rightarrow & \mathit{Fail} \\
!_M; a & \approx & h; f & \Rightarrow & \mathit{Fail} \\
g; f & \approx & g'; f' & \Rightarrow & \mathit{Fail} \\
\pi_i & \approx & \pi_j & \Rightarrow & (S' \mid S(j, \pi_i, h)) \\
\pi_i & \approx & g; f & \Rightarrow & (S' \mid S(i, g; f, h)) \\
!_M; a & \approx & \pi_i & \Rightarrow & (S' \mid S(i, !_M; a, h)) \\
!_M; a & \approx & !_M; a & \Rightarrow & (S' \mid h) \\
g; f & \approx & g'; f & \Rightarrow & (\{g_1 \approx g'_1\} \cup \cdots \cup \{g_n \approx g'_n\} \cup S' \mid h)
\end{array}
$$

Specification of the machine

Diagram Rewriting

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

$$
\begin{array}{lcll}
(f \mid g); (f' \mid g') & \xRightarrow{(h,h')} & (h; f \mid h'; g') & \\
(f \mid \langle g_K, g_N \rangle); (id_K \times \overline{p_N}) & \Rightarrow & (f \mid \langle g_K, [g_N; p_1] \rangle) \cup & \\
 & & \vdots \qquad\qquad\qquad \cup & \\
 & & (f \mid \langle g_K, [g_N; p_n \rangle]) & \\
(f \mid \langle g, [(g' \mid g')] \rangle) & \Rightarrow & (f \mid \langle g, g \rangle) & \\
(f \mid \langle g, [E] \rangle) & \Rightarrow & (h; f \mid \langle h; g, [E'] \rangle) & \text{iff } E \Rightarrow E' \\
R \cup S & \Rightarrow & R' \cup S & \text{iff } R \Rightarrow R' \\
\mathbf{0} \cup S & \Rightarrow & S &
\end{array}
$$

The Machine: An Example

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

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

The Machine: An Example

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

$$\begin{array}{lr} \langle \pi_1 s, \pi_2, \pi_3 \rangle ; \overline{add} & \Rightarrow \\ (\langle \pi_1 s, \pi_2, \pi_3 \rangle ; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \end{array}$$

The Machine: An Example

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

$$
\begin{array}{ll}
\langle \pi_1 s, \pi_2, \pi_3 \rangle ; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle ; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle ; I_{35} ; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle ; W ; (id_2 \times \overline{add}) ; W^\circ ; I_{35}^\circ & \Rightarrow
\end{array}
$$

The Machine: An Example

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

$$
\begin{array}{ll}
\langle \pi_1 s, \pi_2, \pi_3\rangle; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3\rangle; \langle o, \pi_1, \pi_1\rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3\rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3\rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
\langle \pi_1 s, \pi_2, \pi_3\rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3\rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow
\end{array}
$$

The Machine: An Example

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

$$
\begin{array}{ll}
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow
\end{array}
$$

The Machine: An Example

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

$$
\begin{array}{ll}
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow
\end{array}
$$

The Machine: An Example

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

$$
\begin{array}{ll}
\langle \pi_1 s, \pi_2, \pi_3 \rangle ; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle ; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle ; I_{35} ; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle ; W ; (id_2 \times \overline{add}) ; W^\circ ; I_{35}^\circ & \Rightarrow \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle ; I_{35} ; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle ; W ; (id_2 \times \overline{add}) ; W^\circ ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle) ; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle ; W ; (id_2 \times \overline{add}) ; W^\circ ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle) ; W ; (id_2 \times \overline{add}) ; W^\circ ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle) ; (id_2 \times \overline{add}) ; W^\circ ; I_{35}^\circ & \Rightarrow
\end{array}
$$

The Machine: An Example

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

$$
\begin{array}{ll}
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle); (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, [\langle \pi_1, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle] \rangle; W^\circ; I^\circ_{35} \cup \ldots & \Rightarrow
\end{array}
$$

The Machine: An Example

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

$$
\begin{array}{ll}
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle); (id_2 \times \overline{add}); W^\circ; I^\circ_{35} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, [\langle \pi_1, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle] \rangle; W^\circ; I^\circ_{35} \cup \ldots & \Rightarrow \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1 s, [\langle o, \pi_1, \pi_1 \rangle] \rangle; W^\circ; I^\circ_{35} \cup \ldots & \Rightarrow
\end{array}
$$

The Machine: An Example

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

$$
\begin{array}{lr}
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle); (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, [\langle \pi_1, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle] \rangle; W^\circ; I_{35}^\circ \cup \ldots & \Rightarrow \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1 s, [\langle o, \pi_1, \pi_1 \rangle] \rangle; W^\circ; I_{35}^\circ \cup \ldots & \Rightarrow \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1 s, o, \pi_1, \pi_1 \rangle); W^\circ; I_{35}^\circ \cup \ldots & \Rightarrow
\end{array}
$$

The Machine: An Example

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

$$
\begin{array}{lr}
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle); (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, [\langle \pi_1, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle] \rangle; W^\circ; I_{35}^\circ \cup \ldots & \Rightarrow \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1 s, [\langle o, \pi_1, \pi_1 \rangle] \rangle; W^\circ; I_{35}^\circ \cup \ldots & \Rightarrow \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1 s, o, \pi_1, \pi_1 \rangle); W^\circ; I_{35}^\circ \cup \ldots & \Rightarrow \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1, \pi_1 s, o, \pi_1 \rangle); I_{35}^\circ \cup \ldots & \Rightarrow
\end{array}
$$

The Machine: An Example

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

$$
\begin{array}{lr}
\langle \pi_1 s, \pi_2, \pi_3 \rangle; \overline{add} & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle) \cup \ldots & \Rightarrow \\
\mathbf{0} \cup \langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
\langle \pi_1 s, \pi_2, \pi_3 \rangle; I_{35}; \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 \rangle \mid \langle \pi_1 s, \pi_2, \pi_3, \pi_4, \pi_5 \rangle); \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle; W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_2, \pi_3 s, \pi_1, \pi_3 \rangle); W; (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, \pi_1, \pi_2, \pi_3 \rangle); (id_2 \times \overline{add}); W^\circ; I_{35}^\circ & \Rightarrow \\
(\langle \pi_1 s, \pi_2, \pi_3 s \rangle \mid \langle \pi_1 s, \pi_3 s, [\langle \pi_1, \pi_2, \pi_3 \rangle; \langle o, \pi_1, \pi_1 \rangle] \rangle; W^\circ; I_{35}^\circ \cup \ldots & \Rightarrow \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1 s, [\langle o, \pi_1, \pi_1 \rangle] \rangle; W^\circ; I_{35}^\circ \cup \ldots & \Rightarrow \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1 s, o, \pi_1, \pi_1 \rangle); W^\circ; I_{35}^\circ \cup \ldots & \Rightarrow \\
(\langle os, \pi_1, \pi_1 s \rangle \mid \langle os, \pi_1, \pi_1 s, o, \pi_1 \rangle); I_{35}^\circ \cup \ldots & \Rightarrow \\
\langle os, \pi_1, \pi_1 s \rangle \cup \ldots &
\end{array}
$$

then $\langle os, \pi_1, \pi_1 s \rangle$ 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%.

The End

Merci pour votre attention.

Questions?