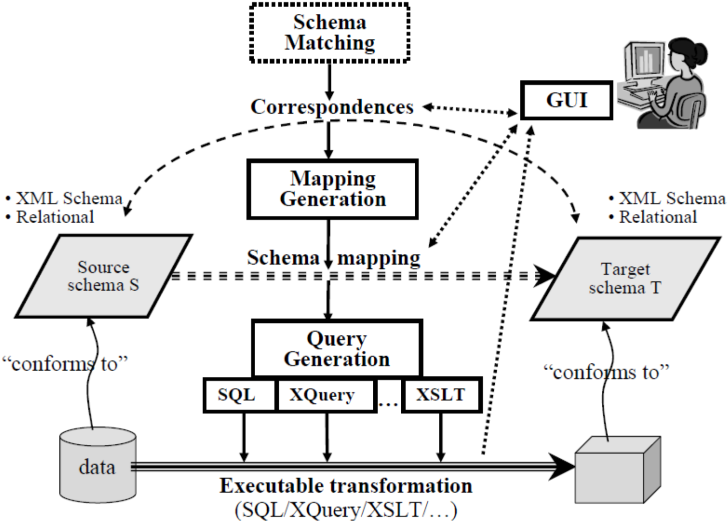
Mapping Polymorphism

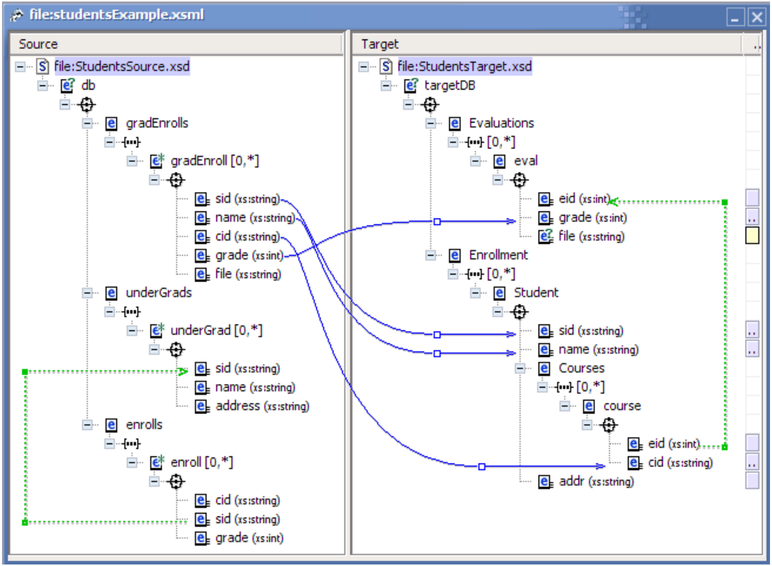
Ryan Wisnesky (Harvard)
Mauricio Hernandez (IBM Almaden)
Lucian Popa (IBM Almaden)

ICDT 2010

Schema Mapping



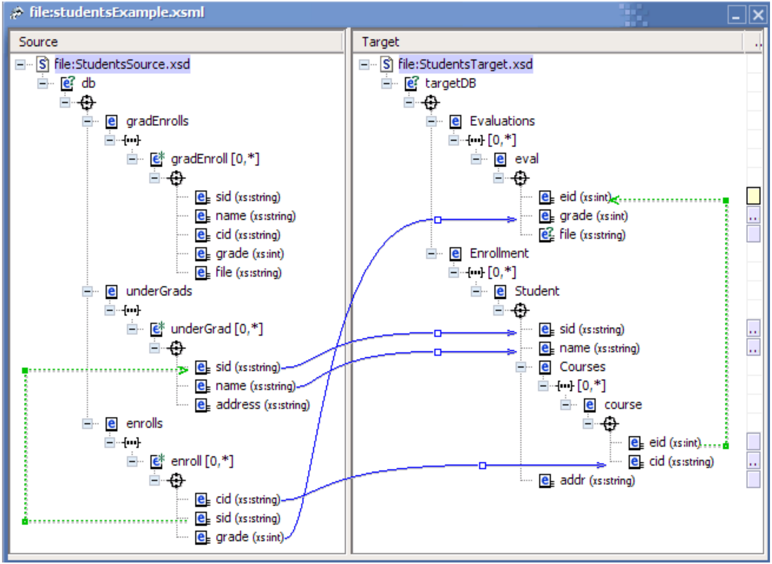
A Schema Mapping in Clio



A Schema Mapping in Clio

$$
\begin{array}{l}
\underline{\text{for}}\ g\ \underline{\text{in}}\ db.\text{gradEnrolls.gradEnroll} \\
\Rightarrow \underline{\text{exists}}\ \ s\ \underline{\text{in}}\ targetDB.\text{Enrollment.Student}, \\
\qquad\qquad\ \ c\ \underline{\text{in}}\ s.\text{Courses.course}, \\
\qquad\qquad\ \ e\ \underline{\text{in}}\ targetDB.\text{Evaluations.eval} \\
\quad\ \underline{\text{where}}\ s.\text{sid} = g.\text{sid}\ \wedge\ s.\text{name} = g.\text{name} \wedge \\
\qquad\qquad\ \ c.\text{cid} = g.\text{cid}\ \wedge\ e.\text{grade} = g.\text{grade} \wedge \\
\qquad\qquad\ \ e.\text{eid} = c.\text{eid}
\end{array}
$$

Nested, Alternating Quantifier Mappings



Nested, Alternating Quantifier Mappings

$$
\begin{array}{l}
\underline{\text{for}}\ u\ \underline{\text{in}}\ db.\text{underGrads.underGrad} \\
\Rightarrow \underline{\text{exists}}\ \ s\ \underline{\text{in}}\ targetDB.\text{Enrollment.Student} \\
\quad\ \underline{\text{where}}\ s.\text{sid} = u.\text{sid}\ \wedge\ s.\text{name} = u.\text{name} \wedge \\
\qquad\qquad (\underline{\text{for}}\ e\ \underline{\text{in}}\ db.\text{enrolls.enroll} \\
\qquad\qquad\ \ \underline{\text{where}}\ e.\text{sid} = u.\text{sid} \\
\qquad\qquad\ \ \Rightarrow \underline{\text{exists}}\ c\ \underline{\text{in}}\ s.\text{Courses.course}, \\
\qquad\qquad\qquad\qquad\quad e'\ \underline{\text{in}}\ targetDB.\text{Evaluations.eval} \\
\qquad\qquad\qquad\ \underline{\text{where}}\ c.\text{cid} = e.\text{cid}\ \wedge\ e'.\text{grade} = e.\text{grade} \wedge \\
\qquad\qquad\qquad\qquad\quad e'.\text{eid} = c.\text{eid})
\end{array}
$$

Motivation: Mapping Re-use

- How can we adapt a mapping $M : S \to T$ to new schema S' and T'? Key questions:
  - When can we reuse M at S', T', and
  - what does it mean when we do?
- Many approaches; for example: ask user for an $M_1 : S' \to S$ and $M_2 : T \to T'$; then compute $M_2 \circ M \circ M_1$.
- Our approach: reuse M directly when $S', T' \leq S, T$; this is, when S', T' are subtypes (expansions) of S, T.

Summary of Results

Practice

1. Alice creates $M : S \to T$.

2. Clio computes principal typing $\hat{S}, \hat{T}$ for M.

3. Bob wants to map $S' \to T'$, where $S', T' \leq \hat{S}, \hat{T}$.

4. Clio suggests using the mapping $M : S' \to T'$.

Summary of Results

Practice	Theory
1. Alice creates $M : S \to T$.	Type-checking ensures that M is satisfiable.
2. Clio computes principal typing $\hat{S}, \hat{T}$ for M.	Type-inference of $\hat{S}, \hat{T}$ is sound and complete.
3. Bob wants to map $S' \to T'$, where $S', T' \leq \hat{S}, \hat{T}$.	We coerce solutions to M from S', T' to $\hat{S}, \hat{T}$.
4. Clio suggests using the mapping $M : S' \to T'$.	Our semantics equates the meaning of $M : S \to T$, $M : S' \to T'$, and $M : \hat{S} \to \hat{T}$.

Outline

- **Formal Definitions of Schema, Mapping**
- Re-use walkthrough

Our Nested Relational Model

$$
\begin{array}{rcl}
Row & ::= & - \mid (\!|\, l : Schema,\ Row \,|\!) \\
Schema & ::= & \texttt{ATOMIC}\ \mathcal{A} \mid \texttt{RCD}\ Row \mid \\
 & & \texttt{SETRCD}\ Row \mid \texttt{SETCHC}\ Row
\end{array}
$$

- For this talk, we will assume rows are always well-formed.
- Order of labels in a row does not matter.
- We abbreviate $(\!|l_1 : t_1, (\!|l_2 : t_2, -|\!)|\!)$ as $(\!|l_1 : t_1, l_2 : t_2|\!)$.
- Let $[\![X]\!]$ denote the set of data instances corresponding to a schema X.

Schema Roots

- A *context* provides types for the free variables of a mapping M, and an *environment* provides data instances.

Definition (Context)

$\Gamma ::= - \mid (v, Schema)\,;\,\Gamma$

Definition (Environment)

$\Delta ::= - \mid (v, I); \Delta$

- We write $\Delta \in [\![\Gamma]\!]$ to indicate that for each $(v, t) \in \Gamma$, there is a corresponding $(v, I) \in \Delta$ such that $I \in [\![t]\!]$.
- When M has two free variables src and dst corresponding to source data and data to be materialized, and $(src, S); (dst, T)$ is a context for M, we write $M : S \to T$.

NR Schema example

$$
\begin{array}{ll}
src, & \texttt{RCD} \; (\!| \; \text{students} : \texttt{SETRCD} \; (\!| \\
& \qquad\qquad \text{fullname} : \text{String}, \\
& \qquad\qquad \text{status} : \texttt{SETCHC} \; (\!| \; \text{teaching} : \text{String}, \\
& \qquad\qquad\qquad\qquad\qquad \text{taking} : \text{String} \; |\!) \; |\!) \; |\!) \\
dst, & \texttt{RCD} \; (\!| \; \text{employees} : \texttt{SETRCD} \; (\!| \; \text{name} : \text{String}, \\
& \qquad\qquad\qquad\qquad\qquad \text{job} : \text{String}, \\
& \qquad\qquad\qquad\qquad\qquad \text{id} : \text{Int} \; |\!) \; |\!)
\end{array}
$$

- DTD representation (ordered):

  <!ELEMENT students (fullname , status)*>
  <!ELEMENT status (teaching | taking)*> ...
  <!ELEMENT employees (name, job, id)*> ...

NR Data example

$$
\begin{array}{ll}
src, & \texttt{RCD} \llparenthesis \text{students} : \texttt{SETRCD} \llparenthesis \\
 & \quad \text{fullname} : \text{String}, \\
 & \quad \text{status} : \texttt{SETCHC} \llparenthesis \text{teaching} : \text{String}, \\
 & \quad\quad \text{taking} : \text{String} \rrparenthesis \rrparenthesis \rrparenthesis \\
dst, & \texttt{RCD} \llparenthesis \text{employees} : \texttt{SETRCD} \llparenthesis \text{name} : \text{String}, \\
 & \quad \text{job} : \text{String}, \\
 & \quad \text{id} : \text{Int} \rrparenthesis \rrparenthesis
\end{array}
$$

$$
\begin{array}{l}
src, \big(\text{students} : \{(\text{fullname} : John\ Doe, \\
\quad \color{red}{\text{status} :\{(\text{teaching} : CS100),\ (\text{taking} : CS200)\})} \\
\quad , \\
\quad (\text{fullname} : Mary\ Jane, \\
\quad\quad \text{status} : \{(\text{taking} : CS100), (\text{taking} : CS200)\}) \\
\quad \}\big)
\end{array}
$$

$$
dst, \big(\text{employees} :\{(\text{name} : John\ Doe, \text{job} : CS100, \text{id} : 1)\}\big)
$$

Compositional Mapping Expressions

$$
\begin{array}{rcl}
Path & ::= & v \mid Path.l \\
\diamond & ::= & \underline{\textsf{for}} \mid \underline{\textsf{exists}} \\
\oplus & ::= & \wedge \mid \Rightarrow \\
M & ::= & \top \mid Path = Path \mid M \oplus M \mid \\
 & & \diamond\, v\ \underline{\textsf{in}}\ Path\,.\ M \mid \\
 & & \diamond\, v\ \underline{\textsf{of}}\ l\ \underline{\textsf{from}}\ Path\,.\ M
\end{array}
$$

- Our mappings are more expressive, and more compositional, than Clio mappings.

Mapping Example

$$
\begin{array}{ll}
src, & \texttt{RCD}\ \llparenthesis\ \textsf{students} : \texttt{SETRCD}\ \llparenthesis \\
& \qquad\qquad \textsf{fullname} : \textsf{String}, \\
& \qquad\qquad \textsf{status} : \texttt{SETCHC}\ \llparenthesis\ \textsf{teaching} : \textsf{String}, \\
& \qquad\qquad\qquad\qquad\qquad \textsf{taking} : \textsf{String}\ \rrparenthesis\ \rrparenthesis\ \rrparenthesis \\
dst, & \texttt{RCD}\ \llparenthesis\ \textsf{employees} : \texttt{SETRCD}\ \llparenthesis\ \textsf{name} : \textsf{String}, \\
& \qquad\qquad\qquad\qquad\qquad \textsf{job} : \textsf{String}, \\
& \qquad\qquad\qquad\qquad\qquad \textsf{id} : \textsf{Int}\ \rrparenthesis\ \rrparenthesis
\end{array}
$$

$$
\begin{array}{l}
\underline{\textsf{for}}\ s\ \underline{\textsf{in}}\ src.\textsf{students}\,. \\
\quad \underline{\textsf{for}}\ t\ \underline{\textsf{of}}\ \textsf{teaching}\ \underline{\textsf{from}}\ s.\textsf{status}\,. \\
\qquad \top \Rightarrow \\
\qquad \underline{\textsf{exists}}\ e\ \underline{\textsf{in}}\ dst.\textsf{employees}\,. \\
\qquad\quad e.\textsf{name} = s.\textsf{fullname}\ \wedge\ e.\textsf{job} = t
\end{array}
$$

Standard Satisfaction Semantics

$$\frac{}{\Delta \models \top} \qquad \frac{\Delta \models m_1 \qquad \Delta \models m_2}{\Delta \models m_1 \wedge m_2} \qquad \frac{\Delta \models m_1 \;\rightarrow\; \Delta \models m_2}{\Delta \models m_1 \Rightarrow m_2}$$

$$\frac{\Delta \models p_1 \leadsto I \qquad \Delta \models p_2 \leadsto I}{\Delta \models p_1 = p_2} \qquad \frac{\Delta \models p \leadsto I \qquad \forall i \in I,\ (v, i); \Delta \models m}{\Delta \models \underline{\mathsf{for}}\ v\ \underline{\mathsf{in}}\ p.\ m}$$

$$\frac{\Delta \models p \leadsto I \qquad \exists i \in I,\ (v, i); \Delta \models m}{\Delta \models \underline{\mathsf{exists}}\ v\ \underline{\mathsf{in}}\ p.\ m}$$

$$\frac{\Delta \models p \leadsto I \qquad \forall (l : i) \in I,\ (v, i); \Delta \models m}{\Delta \models \underline{\mathsf{for}}\ v\ \underline{\mathsf{of}}\ l\ \underline{\mathsf{from}}\ p.\ m}$$

$$\frac{\Delta \models p \leadsto I \qquad \exists (l : i) \in I,\ (v, i); \Delta \models m}{\Delta \models \underline{\mathsf{exists}}\ v\ \underline{\mathsf{of}}\ l\ \underline{\mathsf{from}}\ p.\ m}$$

Outline

- Formal Definitions of Schema, Mapping
- **Re-use walkthrough**

Summary of Results

Practice	Theory
1. Alice creates $M : S \to T$.	**Type-checking ensures that M is satisfiable.**
2. Clio computes principal typing $\hat{S}, \hat{T}$ for M.	Type-inference of $\hat{S}, \hat{T}$ is sound and complete.
3. Bob wants to map $S' \to T'$, where $S', T' \leq \hat{S}, \hat{T}$.	We coerce solutions to M from S', T' to $\hat{S}, \hat{T}$.
4. Clio suggests using the mapping $M : S' \to T'$.	Our semantics equates the meaning of $M : S \to T$, $M : S' \to T'$, and $M : \hat{S} \to \hat{T}$.

Type-checking

$$\frac{(v,t) \in \Gamma}{\Gamma \vdash v :: t} \qquad \frac{\Gamma \vdash p :: \texttt{RCD}\ (\!|l : t, r|\!)}{\Gamma \vdash p.l :: t}$$

$$\frac{\Gamma \vdash p_1 :: \texttt{ATOMIC}\ a \qquad \Gamma \vdash p_2 :: \texttt{ATOMIC}\ a}{\Gamma \vdash p_1\ =\ p_2} \qquad \frac{}{\Gamma \vdash \top}$$

$$\frac{\Gamma \vdash M_1 \qquad \Gamma \vdash M_2}{\Gamma \vdash M_1\ \oplus\ M_2} \qquad \frac{\Gamma \vdash p :: \texttt{SETRCD}\ r \qquad (v, \texttt{RCD}\ r); \Gamma \vdash M}{\Gamma \vdash \diamond\ v\ \underline{\text{in}}\ p.\ M}$$

$$\frac{\Gamma \vdash p :: \texttt{SETCHC}\ (\!|l : t, r|\!) \qquad (v,t); \Gamma \vdash M}{\Gamma \vdash \diamond\ v\ \underline{\text{of}}\ l\ \underline{\text{from}}\ p.\ M}$$

Type Safety

Theorem

Suppose $\Gamma \vdash M$. *Then* M *is satisfiable. That is, there exists a* $\Delta \in [\![\Gamma]\!]$ *such that* $\Delta \models M$.

- An example ill-typed, unsatisfiable mapping is:
  $\underline{\text{exists}}\ v\ \underline{\text{in}}\ t.\ v = v.l$
- Given an $M : S \to T$ and an instance $I \in [\![S]\!]$, typeability of M does not guarantee existence of a $J \in [\![T]\!]$ such that $I; J \models M$.
  - Language of s-t tgds in [Fagin et al, TCS 2005] always admits solutions.
  - Language of nested mappings in [Fuxman et al, VLDB 2006] uses a complex syntactic check to guarantee solutions.

Summary of Results

Practice	Theory
1. Alice creates $M : S \to T$.	Type-checking ensures that M is satisfiable.
2. Clio computes principal typing $\hat{S}, \hat{T}$ for M.	**Type-inference of $\hat{S}, \hat{T}$ is sound and complete.**
3. Bob wants to map $S' \to T'$, where $S', T' \leq \hat{S}, \hat{T}$.	We coerce solutions to M from S', T' to $\hat{S}, \hat{T}$.
4. Clio suggests using the mapping $M : S' \to T'$.	Our semantics equates the meaning of $M : S \to T$, $M : S' \to T'$, and $M : \hat{S} \to \hat{T}$.

Polymorphism

Definition (Polymorphic NR Schema)

$$
\begin{array}{rcl}
Row & ::= & - \mid (\!|\, Row,\ \mathcal{L} : Schema \,|\!) \mid \rho \\
Schema & ::= & \texttt{ATOMIC}\ \alpha \mid \texttt{RCD}\ Row \mid \\
 & & \texttt{SETRCD}\ Row \mid \texttt{SETCHC}\ Row \mid \sigma
\end{array}
$$

Definition (Principal Typing)

$\hat{\Gamma}$ is a *principal typing* for M iff

1. for every substitution ϕ, we have that $\phi\hat{\Gamma} \vdash M$.
2. for every Γ such that $\Gamma \vdash M$, there is some substitution ϕ such that $\phi\hat{\Gamma} = \Gamma$.

Principal Typing Example

$$
\begin{aligned}
&src,\;\; \mathtt{RCD}\;\llparenthesis\; \rho_1,\; \textsf{students} : \mathtt{SETRCD}\;\llparenthesis\; \rho_2, \\
&\qquad\qquad\qquad \textsf{fullname} : \mathtt{ATOMIC}\;\alpha_1,\; \textsf{status} : \mathtt{SETCHC}\;\llparenthesis\; \rho_3, \\
&\qquad\qquad\qquad\quad \textsf{teaching} : \mathtt{ATOMIC}\;\alpha_2\;\rrparenthesis\;\rrparenthesis\;\rrparenthesis \\
&dst,\;\; \mathtt{RCD}\;\llparenthesis\; \rho_4,\; \textsf{employees} : \mathtt{SETRCD}\;\llparenthesis\; \rho_5,\; \textsf{name} : \mathtt{ATOMIC}\;\alpha_1, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textsf{job} : \mathtt{ATOMIC}\;\alpha_2\;\rrparenthesis\;\rrparenthesis
\end{aligned}
$$

$$
\begin{aligned}
&\underline{\textsf{for}}\; s\; \underline{\textsf{in}}\; src.\textsf{students}\,. \\
&\quad \underline{\textsf{for}}\; t\; \underline{\textsf{of}}\; \textsf{teaching}\; \underline{\textsf{from}}\; s.\textsf{status}\,. \\
&\qquad \top \Rightarrow \\
&\qquad \underline{\textsf{exists}}\; e\; \underline{\textsf{in}}\; dst.\textsf{employees}\,. \\
&\qquad\quad e.\textsf{name} = s.\textsf{fullname} \;\wedge\; e.\textsf{job} = t
\end{aligned}
$$

Type-inference

- The input to type inference is a mapping M and a context Γ, and the result is a substitution S. We write this as $S\Gamma \Vdash M$.
- Our particular algorithm is based on qualified types [Gaster and Jones, Hugs Haskell].
- Extend Hindley-Milner algorithm to account for permutation of rows: $(\!|l_1 : t_1, l_2 : t_2|\!)$ must unify with $(\!|l_2 : t_2, l_1 : t_1|\!)$.

Theorem (Soundness)

For all $\varphi\Gamma \Vdash M$, $\varphi\Gamma \vdash M$.

Theorem (Completeness)

For all $\varphi\Gamma \vdash M$, *there exists* S *and* s *such that* $S\Gamma \Vdash M$ *and* $\varphi = s \circ S$.

Summary of Results

Practice	Theory
1. Alice creates $M : S \to T$.	Type-checking ensures that M is satisfiable.
2. Clio computes principal typing $\hat{S}, \hat{T}$ for M.	Type-inference of $\hat{S}, \hat{T}$ is sound and complete.
3. Bob wants to map $S' \to T'$, where $S', T' \leq \hat{S}, \hat{T}$.	**We coerce solutions to M from S', T' to $\hat{S}, \hat{T}$.**
4. Clio suggests using the mapping $M : S' \to T'$.	Our semantics equates the meaning of $M : S \to T$, $M : S' \to T'$, and $M : \hat{S} \to \hat{T}$.

Structural Subtyping

$$\frac{}{(\!|l : t, r|\!) \prec r} \qquad \frac{t' < t \qquad r' \preceq r}{(\!|l : t', r'|\!) \prec (\!|l : t, r|\!)} \qquad \frac{r' \prec r}{\texttt{SETCHC}\ r' < \texttt{SETCHC}\ r}$$

$$\frac{r' \prec r}{\texttt{SETRCD}\ r' < \texttt{SETRCD}\ r} \qquad \frac{r' \prec r}{\texttt{RCD}\ r' < \texttt{RCD}\ r}$$

$$\begin{array}{ll} src, & \texttt{RCD}\ (\!|\ \textsf{students} : \texttt{SETRCD}\ (\!|\ \textsf{fullname} : \textsf{String}, \\ & \qquad\qquad\qquad \textsf{status} : \texttt{SETCHC}\ (\!|\ \textsf{teaching} : \textsf{String},\ \textbf{taking} : \textbf{String}\ |\!)\ |\!)\ |\!) \\ \leq & \\ src, & \texttt{RCD}\ (\!|\ \textsf{students} : \texttt{SETRCD}\ (\!|\ \textsf{fullname} : \textsf{String}, \\ & \qquad\qquad\qquad \textsf{status} : \texttt{SETCHC}\ (\!|\ \textsf{teaching} : \textsf{String}\ |\!)\ |\!)\ |\!) \end{array}$$

$$\begin{array}{ll} dst, & \texttt{RCD}\ (\!|\ \textsf{employees} : \texttt{SETRCD}\ (\!|\ \textsf{name} : \textsf{String}, \textsf{job} : \textsf{String},\ \textbf{id} : \textbf{Int}\ |\!)\ |\!) \\ \leq & \\ dst, & \texttt{RCD}\ (\!|\ \textsf{employees} : \texttt{SETRCD}\ (\!|\ \textsf{name} : \textsf{String}, \textsf{job} : \textsf{String}\ |\!)\ |\!) \end{array}$$

Subtyping and Semantics

- It is not the case that $X \leq Y$ implies $[\![X]\!] \subseteq [\![Y]\!]$.
  - Example: $\texttt{RCD}\ (\!|l : \texttt{Int}|\!) \not\lesssim \texttt{RCD}\ (\!|\,|\!)$
- But we can still relate subtyping to *erasure*.

Definition (Erasure)

When $\Gamma' \leq \Gamma$, we can define an operation

$$erase(\Gamma' \leq \Gamma) : [\![\Gamma']\!] \to [\![\Gamma]\!]$$

that removes data from instances in $[\![\Gamma']\!]$ so that they become instances in $[\![\Gamma]\!]$.

Erasure

- The $erase$ operation generalizes relational projection.
- For example,

$$erase(\texttt{SETRCD}\ (\!| A : t_1, B : t_2 |\!) \leq \texttt{SETRCD}\ (\!| A : t_1 |\!))$$

is the same as projection from A, B to A.

Theorem

Suppose $\Gamma \vdash M$ *and* $\Gamma' \leq \Gamma$ *and* $\Delta' \in [\![\Gamma']\!]$*. Then* $\Delta' \models M$ *if and only if* $erase(\Gamma' \leq \Gamma)\ (\Delta') \models M$*.*

Summary of Results

Practice	Theory
1. Alice creates $M : S \to T$.	Type-checking ensures that M is satisfiable.
2. Clio computes principal typing $\hat{S}, \hat{T}$ for M.	Type-inference of $\hat{S}, \hat{T}$ is sound and complete.
3. Bob wants to map $S' \to T'$, where $S', T' \leq \hat{S}, \hat{T}$.	We coerce solutions to M from S', T' to $\hat{S}, \hat{T}$.
4. Clio suggests using the mapping $M : S' \to T'$.	**Our semantics equates the meaning of $M : S \to T$, $M : S' \to T'$, and $M : \hat{S} \to \hat{T}$.**

Parametricity

Definition

A mapping meaning function $[\![]\!]$ is *parametric* if $\Gamma \vdash M$ and $\Gamma' \vdash M$ imply $[\![(\Gamma', M)]\!] = [\![(\Gamma, M)]\!]$.

- Parametric semantics are insensitive to irrelevant schema structure, and so are appropriate for re-use.
- A standard satisfaction-based semantics:

$$[\![(\Gamma, M)]\!] = \{\ \Delta \mid \Delta \in [\![\Gamma]\!]\ \wedge\ \Delta \models M\ \}$$

  is not parametric because as the schemas in Γ vary, so do the spaces of instances.
- Principal typings $\hat{\Gamma}$ are unique, so a parametric semantics is:

$$[\![M]\!] = \{\ \Delta \mid \Delta \in [\![\hat{\Gamma}]\!]\ \wedge \Delta \models M\ \}$$

Conclusion

Practice	Theory
1. Alice creates $M : S \to T$.	Type-checking ensures that M is satisfiable.
2. Clio computes principal typing $\hat{S}, \hat{T}$ for M.	Type-inference of $\hat{S}, \hat{T}$ is sound and complete.
3. Bob wants to map $S' \to T'$, where $S', T' \leq \hat{S}, \hat{T}$.	We coerce solutions to M from S', T' to $\hat{S}, \hat{T}$.
4. Clio suggests using the mapping $M : S' \to T'$.	Our semantics equates the meaning of $M : S \to T$, $M : S' \to T'$, and $M : \hat{S} \to \hat{T}$.

Related Work

- Nested Mappings (generation, compilation, etc)
  [Fuxman et al, VLDB 2006]
- Mapping semantics (cores, universal solutions, etc)
  [Fagin, TODS 2005]
- Operations over mappings (composition, inversion, etc)
  [Bernstein et al, VLDB 2006]
  [Fagin, TODS 2007]
- Mapping re-use (line matching, schema templates)
  [Madhavan et al, ICDE 2005]
  [Papotti et al, Data Knowl. Eng. 2009]

Future Work

A types-based approach to mappings can also be used to study:

- Integration with programming languages
- Reuse under schema containment
- Recursion