Conditional Strategic Hedge Transformations

Temur Kutsia

What Is It About?

- Transforming term sequences into term sequences
- Provided that some given conditions hold
- Rules specify a single transformation step
- Strategies define how rules are applied
- All in one language

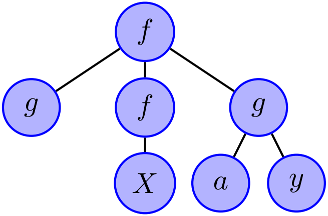
What Is It About?

- Terms are unranked
- A rule may transform the same sequence in (finitely many) different ways: nondeterministic transformations
- A strategy may specify, for instance, the following sequence of rule applications:
  - Apply the rule R_1 as long as possible
  - Transform the result with the first applicable rule from R_2 and R_3
  - Map the rule R_3 on the resulting sequence
  - Transform a subterm occurring somewhere deep in the result by a rule R_4
- Not only rules, but also more complex strategies can be combined in this way

Unranked Terms

Example

$f(g, f(X), g(a, y))$

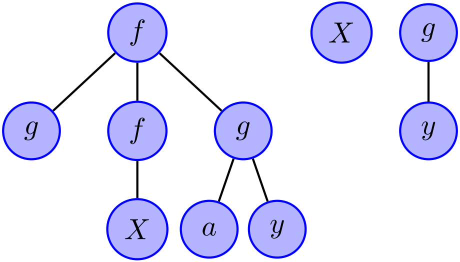


- Arity of function symbols is not fixed.
- Different occurrences of the same function symbol may have different number of arguments.

Hedges

Example

$$f(g, f(X), g(a, y)),\ X,\ g(y)$$



- Finite sequences of unranked terms.

Theories over Unranked Terms and Hedges

Active subject of study in recent years.

- Nearly ubiquitous in XML-related applications.
- Suitable data structures for knowledge representation.
- Model variadic procedures in programming languages.
- Appear in
  - automata theory,
  - rewriting,
  - program analysis and transformation,
  - etc.
- Most of the research activities focus on formal languages, automata, corresponding logics.

Variable Instantiations

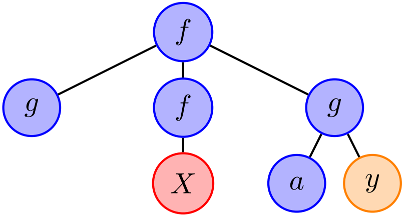
Variables (in the first-order case):

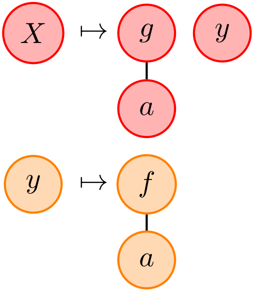
- Individual variables – can be instantiated by individual terms.
- Sequence variables – can be instantiated by hedges.

Variable Instantiations

Example

$$f(g, f(X), g(a, y)) \qquad \{X \mapsto (g(a), y), y \mapsto f(a)\}$$

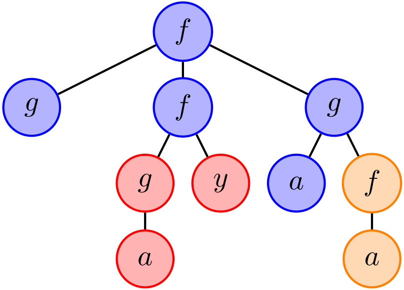


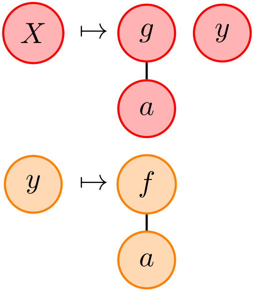


Variable Instantiations

Example

$$f(g, f(g(a), y), g(a, f(a))) \qquad \{X \mapsto (g(a), y), y \mapsto f(a)\}$$





Variable Instantiations

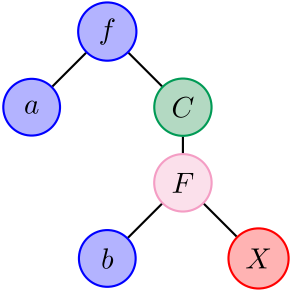
Variables (in the second-order case):

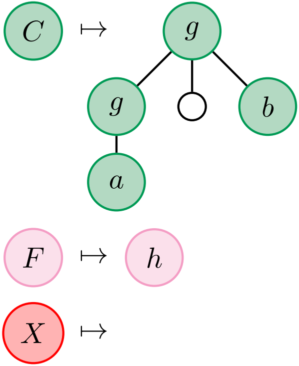
- Individual variables – can be instantiated by individual terms.
- Sequence variables – can be instantiated by hedges.
- Function variables – can be instantiated by function symbols.
- Context variables – can be instantiated by contexts (special unary functions).

Variable Instantiations

Example

$f(a, C(F(b, X)))$ $\{C \mapsto g(g(a), \circ, b), X \mapsto (), F \mapsto h\}$

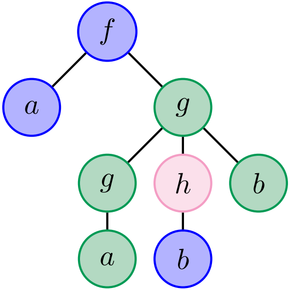


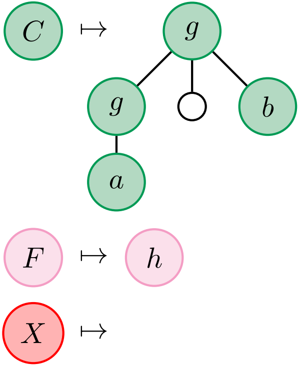


Variable Instantiations

Example

$$f(a, g(g(a), h(b), b)) \quad \{C \mapsto g(g(a), \circ, b), X \mapsto (), F \mapsto h\}$$





Variables

- Sequence variables are pragmatic necessity when function symbols are unranked.
- They help to select subsequences of arbitrary length.
- Context variables help to select subexpressions at arbitrary depth.
- Function variables are handy when one does not know the function symbol name.
- All of them greatly increase expressive power and flexibility.
- Have to be dealt with more involved symbolic techniques.

Matching

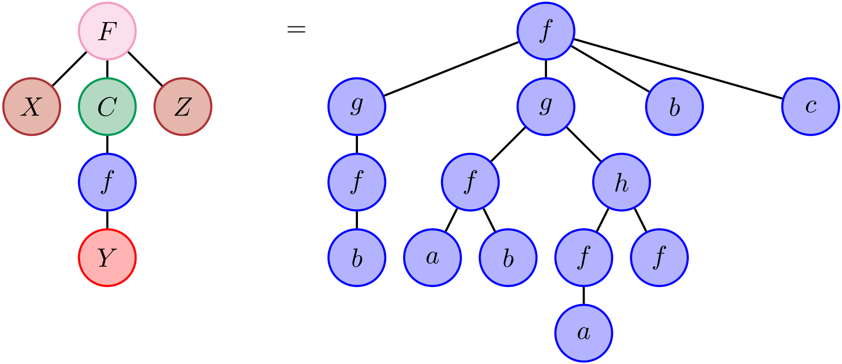
- When a rule is applied, its left hand side should match the hedge to be transformed.
- Requires a matching algorithm.

Syntactic matching for Unranked Terms

- Given: Two unranked terms: pattern and data.
- Find: A substitution that when applied to the pattern, makes it identical to the data.

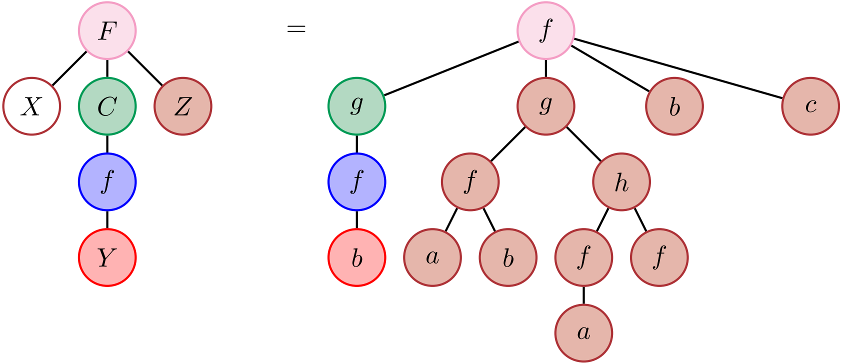
Syntactic Matching for Unranked Terms

$$F(X, C(f(Y)), Z) \quad = \quad f(g(f(b)), g(f(a, b), h(f(a), f)), b, c)$$



Syntactic Matching for Unranked Terms

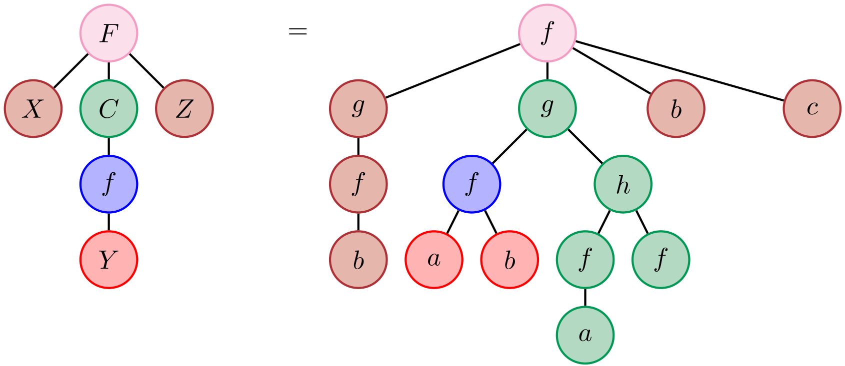
$$F(X, C(f(Y)), Z) \quad = \quad f(g(f(b)), g(f(a,b), h(f(a), f)), b, c)$$



$$\{F \mapsto f, X \mapsto (), C \mapsto g(\circ), Y \mapsto b, Z \mapsto (g(f(a,b), h(f(a), f)), b, c)\}$$

Syntactic Matching for Unranked Terms

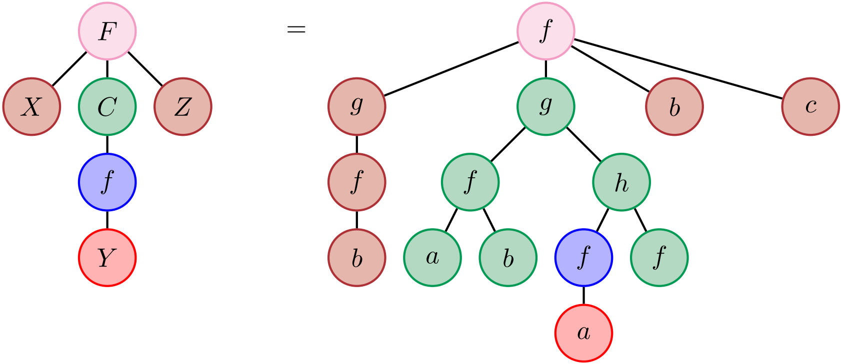
$$F(X, C(f(Y)), Z) \quad = \quad f(g(f(b)), g(f(a,b), h(f(a), f)), b, c)$$



$$\{F \mapsto f, X \mapsto g(f(b)), C \mapsto g(\circ, h(f(a), f)), Y \mapsto (a,b), Z \mapsto (b,c)\}$$

Syntactic Matching for Unranked Terms

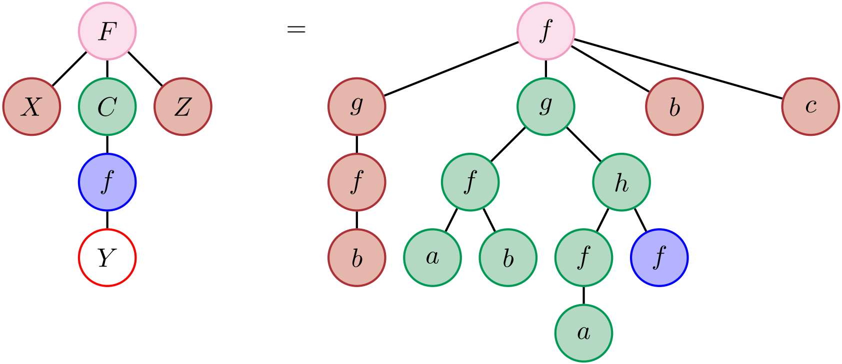
$$F(X, C(f(Y)), Z) \quad = \quad f(g(f(b)), g(f(a,b), h(f(a), f)), b, c)$$



$$\{F \mapsto f, X \mapsto g(f(b)), C \mapsto g(f(a,b), h(\circ, f)), Y \mapsto a, Z \mapsto (b,c)\}$$

Syntactic Matching for Unranked Terms

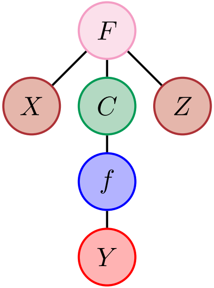
$$F(X, C(f(Y)), Z) \quad = \quad f(g(f(b)), g(f(a,b), h(f(a), f)), b, c)$$



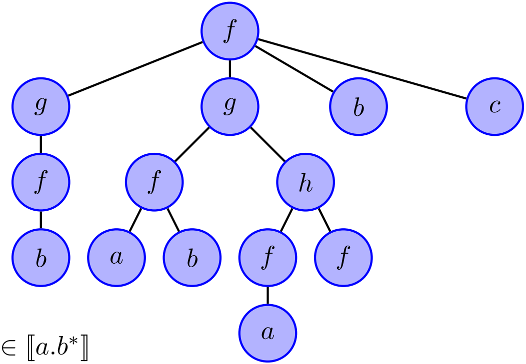
$$\{F \mapsto f, X \mapsto g(f(b)), C \mapsto g(f(a,b), h(f(a), \circ)), Y \mapsto (), Z \mapsto (b,c)\}$$

Syntactic Matching for Unranked Terms

$$F(X, C(f(Y)), Z) \quad = \quad f(g(f(b)), g(f(a,b), h(f(a), f)), b, c)$$



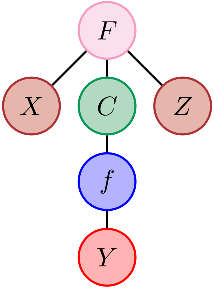
$=$



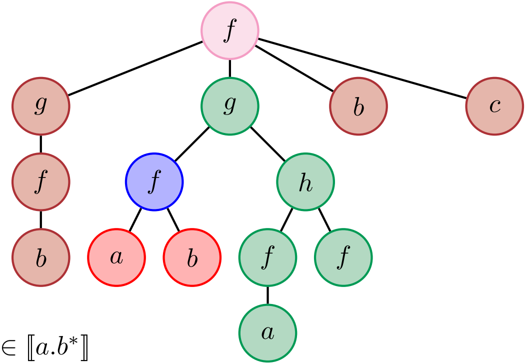
$$C \in [\![g(\_{}_{seq}, \circ, \_{}_{seq})^*]\!], Y \in [\![a.b^*]\!]$$

Syntactic Matching for Unranked Terms

$$\color{pink}{F}\color{black}{(}\color{brown}{X}\color{black}{,}\color{green}{C}\color{black}{(f(}\color{red}{Y}\color{black}{)),}\color{brown}{Z}\color{black}{)} \quad = \quad f(g(f(b)), g(f(a,b), h(f(a), f)), b, c)$$



=



$$C \in [\![g(\_{seq}, \circ, \_{seq})^*]\!], Y \in [\![a.b^*]\!]$$

$$\{\color{pink}{F \mapsto f}\color{black}{,} \color{brown}{X \mapsto g(f(b))}\color{black}{,} \color{green}{C \mapsto g(\circ, h(f(a), f))}\color{black}{,} \color{red}{Y \mapsto (a,b)}\color{black}{,} \color{brown}{Z \mapsto (b,c)}\color{black}{\}}$$

Solving Matching Problems

- A sound, terminating, and complete algorithm.
- Integrates membership constraints into matching.
- No generate-and-test.
- Computes the right answers directly.

Transformations

- Ternary predicate $::\rightarrow$.
- Atoms: $::\rightarrow(t, \langle h_1 \rangle, \langle h_2 \rangle)$, where
  - $\langle\ \rangle$ is an unranked function symbol.
  - t can not be a sequence variable.
  - h_1, h_2 – hedges.
  - The term t is called a strategy.
- Syntactic sugar: $t :: h_1 \rightarrow h_2$.
- Intuition: The strategy t transforms the hedge h_1 into the hedge h_2.
- (Conditional) hedge transformation rules: Nonnegative Horn clauses in this language.
- Queries: Negative clauses.

Rules and Queries

- Rules:

$$
\begin{aligned}
&\mathit{strategy}_0 :: \mathit{hedge}_0 \to \mathit{hedge}'_0 \Leftarrow \\
&\qquad \mathit{strategy}_1 :: \mathit{hedge}_1 \to \mathit{hedge}'_1, \\
&\qquad \ldots \\
&\qquad \mathit{strategy}_n :: \mathit{hedge}_n \to \mathit{hedge}'_n.
\end{aligned}
$$

- Queries

$$
\begin{aligned}
\Leftarrow \quad &\mathit{strategy}_1 :: \mathit{hedge}_1 \to \mathit{hedge}'_1, \\
&\ldots \\
&\mathit{strategy}_n :: \mathit{hedge}_n \to \mathit{hedge}'_n.
\end{aligned}
$$

Logic: Bad News

- Logic with unranked symbols and sequence variables is not compact.
- Counterexample of compactness. An infinite set consisting of:

$$
\begin{aligned}
\exists X.\ & p(X) \\
& \neg p \\
\forall x_1.\ & \neg p(x_1) \\
\forall x_1, x_2.\ & \neg p(x_1, x_2) \\
\forall x_1, x_2, x_3.\ & \neg p(x_1, x_2, x_3) \\
\dots &
\end{aligned}
$$

- Every finite subset of this set has a model, but the entire set does not.

Logic: Bad News

Consequences:

- No complete proof theory.
- A potentially serious blow to prospects of automated reasoning with sequence variables.

Good News

- The clausal fragment behaves well.
- Herbrand's theorem holds.
- Refutationally complete proof method possible.
- Clausal fragment covers many practical cases.

Inference System: The ρLog Calculus

- Resolution:

$$\frac{\Leftarrow str :: h_1 \to h_2, Q \qquad str' :: h_1' \to h_2' \Leftarrow Body}{(\Leftarrow Body, id :: h_2' \to h_2, Q)\sigma},$$

where $\sigma \in mcsm(\{str' \ll str, h_1' \ll h_1\})$.

Inference System: The ρLog Calculus

- Resolution:

$$\frac{\Leftarrow str :: h_1 \to h_2, Q \qquad str' :: h_1' \to h_2' \Leftarrow Body}{(\Leftarrow Body, id :: h_2' \to h_2, Q)\sigma},$$

  where $\sigma \in mcsm(\{str' \ll str, h_1' \ll h_1\})$.

- Identity factoring:

$$\frac{\Leftarrow id :: h_1 \to h_2, Q}{Q\sigma},$$

  where $\sigma \in mcsm(\{h_2 \ll h_1\})$.

Inference System: The ρLog Calculus

- Resolution:

$$\frac{\Leftarrow str :: h_1 \to h_2, Q \qquad str' :: h_1' \to h_2' \Leftarrow Body}{(\Leftarrow Body, id :: h_2' \to h_2, Q)\sigma},$$

  where $\sigma \in mcsm(\{str' \ll str, h_1' \ll h_1\})$.

- Identity factoring:

$$\frac{\Leftarrow id :: h_1 \to h_2, Q}{Q\sigma},$$

  where $\sigma \in mcsm(\{h_2 \ll h_1\})$.

- Resolution + identity factoring is refutationally complete for conditional hedge transformations.
- We have to guarantee that at each step there is a matching problem (and not unification).

Well-Modedness Guarantees Matching

Well-moded queries and clauses:

- A query

$$\Leftarrow t_1 :: h_1 \to h'_1, \ldots, t_n :: h_n \to h'_n$$

is well-moded, if for all $1 \leq i \leq n$,

$$\cup_{j=1}^{i-1} vars(h'_j) \supseteq vars(t_i, h_i).$$

Well-Modedness Guarantees Matching

Well-moded queries and clauses:

- A query

$$\Leftarrow t_1 :: h_1 \to h'_1, \ldots, t_n :: h_n \to h'_n$$

  is well-moded, if for all $1 \leq i \leq n$,

$$\cup_{j=1}^{i-1} vars(h'_j) \supseteq vars(t_i, h_i).$$

- A clause

$$t_0 :: h'_0 \to h_{n+1} \Leftarrow t_1 :: h_1 \to h'_1, \ldots, t_n :: h_n \to h'_n$$

  is well-moded if for all $1 \leq i \leq n+1$,

$$\cup_{j=0}^{i-1} vars(t_0, h'_j) \supseteq vars(t_i, h_i).$$

Negation and Anonymous Variables

- Anonymous variables (for each kind of variable we have) are very handy.
- They need a special treatment in matching (not hard).
- Clause bodies and queries may contain negative literals.
- They are interpreted as "negation as finite failure".
- $t :: h_1 \not\to h_2$: All attempts to transform h_1 into h_2 by t terminate with failure.
- Well-modedness has to be extended to clauses and queries with anonymous variables and negation.

Simple Example: First-Order Rewriting

Clauses:

$$rewrite(z) :: C(x) \to C(y) \Leftarrow z :: x \to y.$$
$$strat :: f(x) \to g(x).$$
$$strat :: f(f(x)) \to x.$$

Goal:

$$rewrite(strat) :: h(f(f(a)), f(a)) \to x.$$

Answers:

$$x = h(g(f(a)), f(a)).$$
$$x = h(a, f(a)).$$
$$x = h(f(g(a)), f(a)).$$
$$x = h(f(f(a)), g(a)).$$

Defining and Combining Strategies

Composition:

$$
\begin{aligned}
&compose(x_{str}, X_{strs}) :: X \to Y \Leftarrow \\
&\qquad x_{str} :: X \to Z, \\
&\qquad compose(X_{strs}) :: Z \to Y. \\
&compose() :: X \to X.
\end{aligned}
$$

Choice:

$$
\begin{aligned}
&choice(x_{str}, X_{strs}) :: X \to Y \Leftarrow \\
&\qquad x_{str} :: X \to Y. \\
&choice(x_{str}, y_{str}, X_{strs}) :: X \to Y \Leftarrow \\
&\qquad choice(y_{str}, X_{strs}) :: X \to Y.
\end{aligned}
$$

Defining and Combining Strategies

Closure:

$$\begin{aligned}
&closure(x_{str}) :: X \to X\,.\\
&closure(x_{str}) :: X \to Y \Leftarrow\\
&\qquad x_{str} :: X \to Z\,,\\
&\qquad closure(x_{str}) :: Z \to Y\,.
\end{aligned}$$

Normal form:

$$\begin{aligned}
&nf(x_{str}) :: X \to Y \Leftarrow\\
&\qquad closure(x_{str}) :: X \to Y\,,\\
&\qquad x_{str} :: Y \not\to_{-seq}\,.
\end{aligned}$$

Defining and Combining Strategies

First applicable strategy:

$$
\begin{aligned}
& \mathit{first}(x_{str}, X_{strs}) :: X \rightarrow Y \Leftarrow \\
& \qquad x_{str} :: X \rightarrow Y. \\
& \mathit{first}(x_{str}, y_{str}, X_{strs}) :: X \rightarrow Y \Leftarrow \\
& \qquad x_{str} :: X \not\rightarrow {}_{-seq}, \\
& \qquad \mathit{first}(y_{str}, X_{strs}) :: X \rightarrow Y.
\end{aligned}
$$

Map:

$$
\begin{aligned}
& map(x_{str}) :: () \rightarrow (). \\
& map(x_{str}) :: (x, X) \rightarrow (y, Y) \Leftarrow \\
& \qquad x_{str} :: x \rightarrow y, \\
& \qquad map(x_{str}) :: X \rightarrow Y.
\end{aligned}
$$

Simple Example. Sorting.

$$reorder(F_{ord}) :: (X, x, Y, y, Z) \rightarrow (X, y, Y, x, Z) \Leftarrow F_{ord}(y, x).$$

Simple Example. Sorting.

$$reorder(F_{ord}) :: (X, x, Y, y, Z) \to (X, y, Y, x, Z) \Leftarrow F_{ord}(y, x).$$

- $reorder(F_{ord})$ reorders two elements in the input hedge that are in the reversed order with respect to F_{ord}.
- $reorder(>) :: (1, 3, 2) \to Y$ nondeterministically returns two instantiations for Y: $(3, 1, 2)$ and $(2, 3, 1)$.

Simple Example. Sorting

$$sort(F_{ord}) := nf(reorder(F_{ord}))$$

Simple Example. Sorting

$$sort(F_{ord}) := nf(reorder(F_{ord}))$$

- The query

$$sort(>) :: (3, 3, 1, 2, 4) \to Y.$$

computes the instantiation of Y: $(4, 3, 3, 2, 1)$.

Simple Example. Zip

$$
\begin{aligned}
zipstep :: (F_{op}, F(x, X), F(y, Y), F(Z)) \rightarrow \\
(F_{op}, F(X), F(Y), F(Z, F_{op}(x, y))).
\end{aligned}
$$

$$zipstep :: (\_{fun}, F, F, z) \rightarrow z.$$

$$
\begin{aligned}
& zip :: (F_{op}, F(X), F(Y)) \rightarrow z \Leftarrow \\
& \quad nf(zipstep) :: (F_{op}, F(X), F(Y), F) \rightarrow z.
\end{aligned}
$$

Simple Example. Zip

$$
\begin{aligned}
zipstep &:: (F_{op}, F(x, X), F(y, Y), F(Z)) \to \\
&\quad\; (F_{op}, F(X), F(Y), F(Z, F_{op}(x, y))). \\
zipstep &:: (\_{}_{fun}, F, F, z) \to z.
\end{aligned}
$$

$$
\begin{aligned}
&zip :: (F_{op}, F(X), F(Y)) \to z \Leftarrow \\
&\quad nf(zipstep) :: (F_{op}, F(X), F(Y), F) \to z.
\end{aligned}
$$

- The query

$$zip :: (g, f(1, 2, 3), f(a, b, c)) \to z.$$

computes the instantiation of z: $f(g(1, a), g(2, b), g(3, c))$.

Simple Example. Substitution Application

$$applystep :: (x \mapsto y, C(x)) \to (x \mapsto y, C(y)).$$

$$apply :: (x_{subst}, y_{expr}) \to z_{instance} \Leftarrow$$
$$nf(applystep) :: (x_{subst}, y_{expr}) \to (\_{ind}, z_{instance}).$$

Simple Example. Substitution Application

$$applystep :: (x \mapsto y, C(x)) \to (x \mapsto y, C(y)).$$

$$apply :: (x_{subst}, y_{expr}) \to z_{instance} \Leftarrow$$
$$nf(applystep) :: (x_{subst}, y_{expr}) \to (\_{ind}, z_{instance}).$$

- The query

$$apply :: (v \mapsto f(a), f(v, g(b, v))) \to z.$$

computes the instantiation of z: $f(f(a), g(b, f(a)))$.

Simple Example. Occurrence Check

$$occurs :: (x, \_ctx(x)) \to true.$$

Simple Example. Occurrence Check

$$occurs :: (x, \_{ctx}(x)) \to true.$$

- The query $occurs :: (v, f(v, g(b, v))) \to true$ succeeds.
- The query $occurs :: (g(b, v), f(v, g(b, v))) \to true$ succeeds.
- The query $occurs :: (u, f(v, g(b, v))) \to true$ fails.

Example. First-Order Unification Rules

$$\begin{aligned}
\mathit{decomposition} :: (\{F(X_1) \doteq F(X_2), X_{eqs}\}, z_{subst}) \to \\
(\{Y_{eqs}, X_{eqs}\}, z_{subst}) \Leftarrow \\
\mathit{zip} :: (\doteq, F(X_1), F(X_2)) \to F(Y_{eqs}).
\end{aligned}$$

$$\begin{aligned}
&\mathit{orient} :: (\{x \doteq y, X_{eqs}\}, z_{subst}) \to (\{y \doteq x, X_{eqs}\}, z_{subst}) \Leftarrow \\
&\qquad \mathit{variable} :: y \to \mathit{true}, \\
&\qquad \mathit{variable} :: x \not\to \mathit{true}.
\end{aligned}$$

$$\mathit{variable} :: \mathsf{x} \to \mathit{true}.$$

$$\mathit{variable} :: \mathsf{y} \to \mathit{true}.$$

$\ldots$

Example. First-Order Unification Rules

$$
\begin{aligned}
&\mathit{elimination} :: (\{x \doteq y, X_{eqs}\}, \{Z\}) \to (\{Y_{eqs}\}, \{U, x \mapsto y\}) \Leftarrow \\
&\qquad \mathit{variable} :: x \to \mathit{true}, \\
&\qquad \mathit{occurs} :: (x, y) \not\to \mathit{true}, \\
&\qquad \mathit{apply} :: (x \mapsto y, \{X_{eqs}\}) \to \{Y_{eqs}\}, \\
&\qquad \mathit{apply} :: (x \mapsto y, \{Z\}) \to \{U\}.
\end{aligned}
$$

Example. First-Order Unification Strategy

$$transform := \\ \quad choice(decomposition, elimination, orient).$$

$$unify :: X_{eqs} \rightarrow U_{unifier} \Leftarrow \\ \quad first_{one}(nf(transform)) :: (\{X_{eqs}\}, \{\}) \rightarrow (\{\}, \{U_{unifier}\}).$$

Example. First-Order Unification Strategy

$$\begin{aligned}&\mathit{transform} := \\ &\qquad \mathit{choice}(\mathit{decomposition}, \mathit{elimination}, \mathit{orient}).\end{aligned}$$

$$\begin{aligned}&\mathit{unify} :: X_{eqs} \to U_{unifier} \Leftarrow \\ &\qquad \mathit{first}_{one}(\mathit{nf}(\mathit{transform})) :: (\{X_{eqs}\}, \{\}) \to (\{\}, \{U_{unifier}\}).\end{aligned}$$

- Query: $\mathit{unify} :: (\mathsf{f}(\mathsf{x}) \doteq \mathsf{f}(\mathsf{h}(\mathsf{y})), \mathsf{g}(\mathsf{x},\mathsf{x}) \doteq \mathsf{g}(\mathsf{z},\mathsf{h}(\mathsf{a}))) \to U$
- Answer: $U = (\mathsf{x} \mapsto \mathsf{h}(\mathsf{a}), \mathsf{y} \mapsto \mathsf{a}, \mathsf{z} \mapsto \mathsf{h}(\mathsf{a}))$

Example. First-Order Matching

- The same rules can be used for matching.
- To make it more efficient, we can replace the elimination rule with the new one:

$$
\begin{aligned}
&\mathit{elimination}' :: (\{x \doteq y, X_{eqs}\}, \{Z\}) \to (\{Y_{eqs}\}, \{Z, x \mapsto y\}) \Leftarrow \\
&\qquad \mathit{variable} :: x \to \mathit{true}, \\
&\qquad \mathit{apply} :: (x \mapsto y, \{X_{eqs}\}) \to \{Y_{eqs}\}.
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{transform}' := \\
&\qquad \mathit{choice}(\mathit{decomposition}, \mathit{elimination}', \mathit{orient}).
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{match} :: X_{eqs} \to U_{matcher} \Leftarrow \\
&\qquad \mathit{first}_{one}(\mathit{nf}(\mathit{transform}')) :: (\{X_{eqs}\}, \{\}) \to (\{\}, \{U_{matcher}\}).
\end{aligned}
$$

Potential Use in Web-Related Topics

Querying and transforming XML.

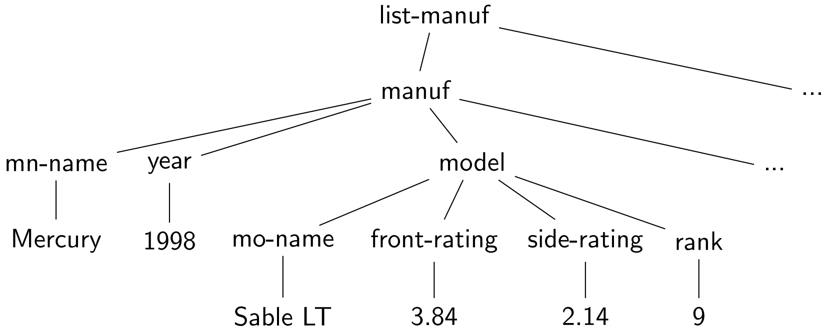
- A list of query operations that are desirable for an XML query and transformation language: selection, extraction, reduction, restructuring, and combination.
- We demonstrate, on the car dealer office example, how these operations can be expressed in ρLog calculus.

Car Dealer Office Example

```
<list-manuf>
 <manuf>
   <mn-name>Mercury</mn-name>
   <year>1998</year>
   <model>
     <mo-name>Sable LT</mo-name>
     <front-rating>
        3.84
     </front-rating>
     <side-rating>
        2.14
     </side-rating>
     <rank>9</rank>
   </model> ...
 </manuf> ...
</list-manuf>
```

```
<list-vehicle>
   <vehicle>
      <vendor>
         Scott Thomason
      </vendor>
      <make>Mercury</make>
      <model>Sable LT</model>
      <year>1999</year>
      <color>
         metallic blue
      </color>
      <price>26800</price>
   </vehicle> ...
</list-vehicle>
```

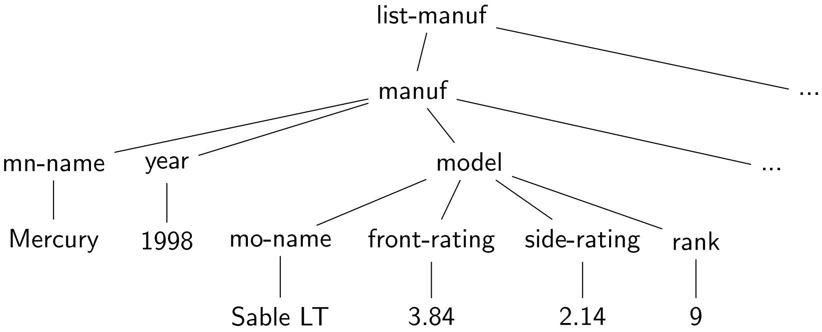
Select and Extract



Select and extract *manuf* elements where some *model* has *rank* ≤ 10:

$$sel\_and\_extr :: list\text{-}manuf(\_{seq}, C(rank(x)), \_{seq}) \to C(rank(x)) \Leftarrow x \leq 10.$$

Reduction



- From the *manufacturer* elements, we want to drop those *model* sub-elements whose *rank* is greater than 10.
- We also want to elide the *front-rating* and *side-rating* elements from the remaining models.

Reduction

One-step reduction:

$$red\_step :: manuf(X_1, model(\_seq, rank(x)), X_2) \rightarrow manuf(X_1, X_2) \Leftarrow$$
$$x > 10.$$
$$red\_step :: manuf(X_1, model(y, \_ind, \_ind, rank(x)), X_2) \rightarrow$$
$$manuf(X_1, model(y, rank(x)), X_2) \Leftarrow$$
$$x \leq 10.$$

Reduction: reduce each element of *list-manuf* (i.e., each *manuf*) by the *red_step* as much as possible.

$$reduce :: list\text{-}manuf(X_1) \rightarrow list\text{-}manuf(X_2) \Leftarrow$$
$$map(nf(red\_step)) :: X_1 \rightarrow X_2.$$

Extended Rule Syntax

- Matching problems extended with membership constraints can be tailored in the atoms.
- $\mathit{strategy} :: h_1 \to h_2 \ \mathit{where}\ \{v_1 \in \mathsf{L}_1, \ldots, v_n \in \mathsf{L}_n\}$.
- Well-modedness extends to the corresponding rules and queries.
- Such rules can be used to validate documents against DTDs (for quite a large class of DTDs).

Incomplete Queries

- Often, a query author does not know or is not interested in the entire structure of a Web document.
- Queries are incomplete.
- Classification of incompleteness (Schaffert, 2004): in breadth, in depth, with respect to order, with respect to optional elements.
- Pretty easily expressed in the ρLog calculus.

Incompleteness in Breadth

- ρLog does do not need any extra construct for incomplete queries in breadth.
- Anonymous sequence variables can be used as wildcards for arbitrary sequences of nodes.
- Named sequence variables can extract arbitrary sequences of nodes without knowing the exact structure.

Incompleteness in Depth

- ρLog does do not need any extra construct for incomplete queries in depth either.
- Anonymous context variables can be used to descend in arbitrary depth in terms to reach a query subterm, skipping the content in between.
- Named context variables can extract the entire context above the query subterm without knowing the structure of the context.

Incompleteness with Respect to Order

- It allows to specify neighboring nodes in a different order than the one in that they occur in the data tree.
- Can be incorporated into ρLog calculus with the help of equational matching modulo orderless theory.
- Without it, an extra line of code is required to get the same effect.

Incompleteness with Respect to Optional Elements

- Since sequence variables can be instantiated with the empty hedge, such queries are trivially expressed in ρLog.

Related Applications

- Logic-based XML querying and transformation in Xcerpt (Bry, Schaffert et al. 2002).
- XML processing in XDuce (Hosoya and Pierce, 2003).
- Rule-based verification of Web sites (Alpuente et al. 2006)
- Access control via strategic rewriting (Dougherty et al. 2007).

Summary

- Necessary ingredients for computing via strategic conditional hedge transformations:
  - Matching with context and sequence variables (solving): Basic mechanism for instantiating variables.
  - Resolution and identity factoring (proving): Inference mechanism.
  - Conditional hedge transformations (transforming): Computation via deduction.
- Separating control and transformations.
- Modeling nondeterministic computations.