On Matching Concurrent Traces

Iliano Cervesato    Frank Pfenning    Jorge Luis Sacchini
Carsten Schürmann    Robert J. Simmons

July 1, 2012
Concurrent Traces

- A **concurrent trace** is a sequence of computational steps where independent steps can be permuted.
- Computational traces define a model for concurrent and distributed computations.
- We analyze the matching problem on concurrent traces.
Long-term objectives

- Develop a logical framework to reason about concurrent traces
- Unification and matching are prerequisites
- This work: matching on concurrent traces, defined by multiset rewriting system, with one variable standing for an unknown trace
Multiset Rewriting Systems

- A multiset rewriting system is a set of rules of the form

\[ r : \tilde{a} \rightarrow \tilde{b} \]

where \( \tilde{a} \) and \( \tilde{b} \) are multisets of elements of \( S \) (support set)

- Rules represent state transformations

- A state is a set of pairs of the form \( x : a \) where \( x \) is a unique name, and \( a \in S \)

- Applying a rule is represented by

\[ R \vdash s \xrightarrow{t} s' \]

\[ R, r : \tilde{a} \rightarrow \tilde{b} \vdash s, \tilde{x} : \tilde{a} \xrightarrow{r(\tilde{x},\tilde{y})} s, \tilde{y} : \tilde{b} \]
Concurrent traces

- A trace is a sequence of computational steps

  \[ t ::= \cdot \mid r(\tilde{x}, \tilde{y}) \mid t_1; t_2 \]

- Multistep computation

  \[ R \vdash s \xrightarrow{t} s' \]
Concurrent traces

Example

\[
x_1 : a_1 \rightarrow r_1(x_1 ; y_1 y_2) \rightarrow y_1 : b_1 \rightarrow r_2(y_1 ; z_2) \rightarrow z_1 : a_1
\]

\[
x_2 : a_2 \rightarrow y_2 : b_2 \rightarrow r_3(x_2 y_2 ; z_3) \rightarrow z_3 : a_3
\]

\[
x_1 : a_1, x_2 : a_2 \xrightarrow{r_1(x_1 , y_1 y_2) ; r_2(y_1 z_1) ; r_3(y_2 x_2 , z_2)} z_1 : a_1, z_3 : a_3
\]

Independent steps can be permuted
Concurrent traces

- Pre- and post-conditions:

\[
\left\{
\begin{align*}
\bullet(\cdot) & = \cdot \\
r(\tilde{x}; \tilde{y}) & = \tilde{x} : \tilde{a} \\
(t_1; t_2) & = \bullet t_1 \cup (\bullet t_2 \setminus t_1\bullet)
\end{align*}
\right.
\]

\[
\left\{
\begin{align*}
(\cdot)\bullet & = \cdot \\
r(\tilde{x}; \tilde{y})\bullet & = \tilde{y} : \tilde{b} \\
(t_1; t_2)\bullet & = (t_1\bullet \setminus \bullet t_2) \cup t_2\bullet
\end{align*}
\right.
\]
Concurrent traces

- **Trace independence**

\[ t_1 \parallel t_2 \iff \bullet t_1 \cap t_2 \bullet = t_1 \bullet \cap \bullet t_2 = \emptyset \]

- **Trace equality:** \( t = t' \)

\[ R \vdash s \xrightarrow{t} s' \iff R \vdash s \xrightarrow{t'} s' \]

- **Equations**

\[ (\vec{t}_1; \vec{t}_2); \vec{t}_3 = \vec{t}_1;(\vec{t}_2; \vec{t}_3) \]

\[ \cdot; \hat{t} = \hat{t} \]

\[ \hat{t}; \cdot = \hat{t} \]

\[ \vec{t}_1; \vec{t}_2 = \vec{t}_2; \vec{t}_1 \quad \text{if} \quad \vec{t}_1 \parallel \vec{t}_2 \]
Concurrent traces

- Trace equality is given by the binding structure
- Internal bindings can be renamed

\[
p(x; y); q(y; z) = p(x; y'); q(y'; z)
\]

but

\[
p(x; y); q(y; z) \neq p(x'; y); q(y; z')
\]

- A renaming is **legal** if it is the identity for \(\bullet t\) and \(t\bullet\)
Concurrent traces

Example

\[ r_1(x_1, y_1, y_2); r_2(y_1, z_1); r_3(y_2, x_2, z_2) =
\]

\[ r_1(x_1, y_1, y_2); r_3(y_2, x_2, z_3); r_2(y_1, z_1) \]
Concurrent traces

Concurrent traces are equivalent to executions on Petri nets

\[
s = x_1:a_1, x_2:a_2
\]
Concurrent traces

Concurrent traces are equivalent to executions on Petri nets

\[
s = x_1 : a_1, x_2 : a_2 \xrightarrow{r_1(x_2, y_1, y_2)} x_1 : a_1, y_1 : b_1, y_2 : b_2
\]
Concurrent traces

Concurrent traces are equivalent to executions on Petri nets

\[ s = x_1 : a_1, x_2 : a_2 \xrightarrow{r_1(x_2,y_1,y_2)} x_1 : a_1, y_1 : b_1, y_2 : b_2 \xrightarrow{r_3(x_1,y_2,z_3)} y_1 : b_1, y_2 : b_2, z_3 : a_3 \]
Concurrent traces

Concurrent traces are equivalent to executions on Petri nets

\[ s = x_1 : a_1, x_2 : a_2 \xrightarrow{r_1(x_2,y_1,y_2)} x_1 : a_1, y_1 : b_1, y_2 : b_2 \]
\[ \xrightarrow{r_3(x_1,y_2,z_3)} y_1 : b_1, y_2 : b_2, z_3 : a_3 \]
\[ \xrightarrow{r_2(y_1,z_2)} z_2 : a_2, z_3 : a_3 \]
Concurrent traces

- Tokens are named in our representation

\[ x_1: a, x_2: a \xrightarrow{r(x_1, y)} y: b, x_2: a \]

\[ x_1: a, x_2: a \xrightarrow{r(x_2, y)} x_1: a, y: b \]
Equations over traces

- Trace variables

\[ t ::= \cdot | r(\tilde{x}; \tilde{y}) | t_1; t_2 | X(\tilde{x}; \tilde{y}) \]

- \( \tilde{x} \) and \( \tilde{y} \) represent the input and output interface of \( X \)
- \( X \) can be instantiated with a trace \( t \) such that \( \bullet t = \tilde{x} \) and \( t\bullet = \tilde{y} \)
- An equation is given by a pair of traces containing variables

\[ \vec{t}_1 \equiv \vec{t}_2 \]
Matching on traces

Given

\[ t_1 \equiv t_2 \quad \text{with } t_2 \text{ ground} \]

Find a substitution \( \theta = X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n \) such that

\[ t_1[\theta] = t_2 \]
Matching on traces

- Given

  \[ t_1 = t_2 \quad \text{with } t_2 \text{ ground} \]

  Find a substitution \( \theta = X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n \) such that

  \[ t_1[\theta] = t_2 \]

- Matching is inherently non-deterministic: the following problem encodes multiset matching

  \[ X(\cdot;\cdot); Y(\cdot;\cdot) \equiv c_1(\cdot;\cdot); \ldots; c_n(\cdot;\cdot) \]

  Partition \( \{c_1, \ldots, c_n\} \) in two sets \( X \) and \( Y \).
Matching on traces

- Matching is decidable:

  \[ t_1; X(\tilde{x}, \tilde{y}); t'_1 \equiv t_2 \]

  \(X(\tilde{x}, \tilde{y})\) must be instantiated with a subtrace of \(t_2\)

- In a problem

  \[ t^1_1; X_1(\tilde{x}_1, \tilde{y}_1); t^1_2; \ldots X_n(\tilde{x}_n, \tilde{y}_n); t^1_n \equiv t_2 \]

  we can try all possible combinations of subtraces of \(t_2\)
Matching on traces

- Matching is decidable:

\[ t_1; X(\tilde{x}, \tilde{y}); t'_1 \equiv t_2 \]

- \( X(\tilde{x}, \tilde{y}) \) must be instantiated with a subtrace of \( t_2 \)

- In a problem

\[ t_1^1; X_1(\tilde{x}_1, \tilde{y}_1); t_2^1; \ldots X_n(\tilde{x}_n, \tilde{y}_n); t_n^1 \equiv t_2 \]

- we can try all possible combinations of subtraces of \( t_2 \)

- Expensive
Matching on traces

- We consider 1-variable matching

\[
t_1; X(\tilde{x}_1, \tilde{y}_1); t_1' \overset{?}{=} t_2
\]

where \( t_1, t'_1 \) and \( t_2 \) are ground
Matching on traces

- We consider 1-variable matching

\[ t_1; X(\tilde{x}_1, \tilde{y}_1); t'_1 \equiv t_2 \]

where \( t_1, t'_1 \) and \( t_2 \) are ground

- Reduces search space
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[ t_1; X(\tilde{x}_1; \tilde{y}_1); t_2 \quad ? \quad u_1 \]
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[ t_1; X(\tilde{x}_1; \tilde{y}_1); t_2 \quad ? \quad p(\tilde{x}; \tilde{y}); u_2 \]
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[ p(\tilde{x}; \tilde{y}); t_3; X(\tilde{x}_1; \tilde{y}_1); t_2 \equiv p(\tilde{x}; \tilde{y}); u_2 \]
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[ t_3; X(\tilde{x}_1; \tilde{y}_1); t_2 \overset{?}{=} u_2 \]
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[
t_3; X(\tilde{x}_1; \tilde{y}_1); t_2 \quad \equiv \quad u_2 \\
\vdots \\
X(\tilde{x}_1; \tilde{y}_1); t_4 \quad \equiv \quad u_3
\]
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[
\begin{align*}
\text{Solution:} & \\
X(\tilde{x}_1; \tilde{y}_1; t_3; X(\tilde{x}_1; \tilde{y}_1); t_2 & \overset{?}{=} u_2 \\
\vdots & \\
X(\tilde{x}_1; \tilde{y}_1; t_4 & \overset{?}{=} u_4; p'(\tilde{x}_2; \tilde{y}_2)
\end{align*}
\]
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[
t_3; X(\tilde{x}_1; \tilde{y}_1); t_2 \quad ? \quad u_2
\]

\[
\vdots
\]

\[
X(\tilde{x}_1; \tilde{y}_1); t_5; p'(\tilde{x}_2; \tilde{y}_2) \quad ? \quad u_4; p'(\tilde{x}_2; \tilde{y}_2)
\]

Solution: \[X \leftarrow u_5\]
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[
\begin{align*}
t_3; X(\tilde{x}_1; \tilde{y}_1); t_2 & \equiv u_2 \\
& \vdots \\
X(\tilde{x}_1; \tilde{y}_1); t_5 & \equiv u_4
\end{align*}
\]
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[ t_3; X(\tilde{x}_1; \tilde{y}_1); t_2 \equiv \ u_2 \]
\[ \vdots \]
\[ X(\tilde{x}_1; \tilde{y}_1); t_5 \equiv \ u_4 \]
\[ \vdots \]
\[ X(\tilde{x}_1; \tilde{y}_1) \equiv \ u_5 \]
Intuition of the algorithm

- Match individual computation steps until we are left with the logic variable

\[
\begin{align*}
t_3; X(\tilde{x}_1; \tilde{y}_1); t_2 & \equiv u_2 \\
\vdots \\
x(X(\tilde{x}_1; \tilde{y}_1); t_5 & \equiv u_4 \\
\vdots \\
x(X(\tilde{x}_1; \tilde{y}_1) & \equiv u_5 \\
\end{align*}
\]

Solution: \( X \leftarrow u_5 \)
Algorithm

1. \( p(\tilde{x}; \tilde{y}); \vec{t}_1 \equiv p(\tilde{x}; \tilde{y}'); \vec{t}_2: \)
   
   If \( \tilde{y}' / \tilde{y} \) is legal for \( p(\tilde{x}; \tilde{y}); \vec{t}_1 \), then solve \( [\tilde{y}' / \tilde{y}]\vec{t}_1 \equiv \vec{t}_2 \), otherwise fail.

2. \( \vec{t}_1; p(\tilde{x}; \tilde{y}) \equiv \vec{t}_2; p(\tilde{x}'; \tilde{y}): \)
   
   If \( \tilde{x}' / \tilde{x} \) is legal for \( \vec{t}_1; p(\tilde{x}; \tilde{y}) \), then solve \( [\tilde{x}' / \tilde{x}]\vec{t}_1 \equiv \vec{t}_2 \), otherwise fail.

3. \( X(\tilde{x}; \tilde{y}) \equiv \vec{t}_2: \) Simply return the solution \( X \leftarrow \vec{t}_2 \).

Invariant: in \( t_1 \equiv t_2 \), \( \bullet t_1 = \bullet t_2 \) and \( t_1\bullet = t_2\bullet \)
Example

- Match failure:

\[ p(x_1; y_1 y_2); X(y_1 y_2; z_1 z_2) \overset{?}{=} p(x_1; y'_1 z_2); q(y'_1; z_1) \]

Matching fails because there is no legal renaming between

\[ y_1 y_2 \longrightarrow y'_1 z_2 \]
Correctness of the algorithm

Soundness

Given a matching problem $\vec{t}_1; X(\tilde{x}; \tilde{y}); \vec{t}_1 \equiv \vec{t}_2$, if the matching algorithm reports a solution $X \leftarrow \vec{t}$, then there is a legal renaming $\rho$ such that $\rho \vec{t}_1; \vec{t}; \rho \vec{t}_1 = \vec{t}_2$.

Proof

By induction on the size of the trace.
Correctness of the algorithm

Completeness

Given a matching problem \( t_1; X(\tilde{x}; \tilde{y}); t_1' \equiv t_2 \), if there is a trace \( \vec{t} \) such that \( t_1; \vec{t}; t_1' = t_2 \), then the matching algorithm will report the solution \( X \leftarrow \rho \vec{t} \) for some renaming \( \rho \).

Proof

By induction on the size of the trace.
Extensions

- Extend matching to CLF
  - Affine and persistent functions
  - Dependent types
  - This work: propositional linear fragment of CLF

- Current results
  - 1-var matching for larger fragments CLF
  - 1-var unification for simply-typed fragment of CLF:

\[ t_1; X_1; t'_1 \equiv t_2; X_2; t'_2 \]
Conclusions

- We presented an algorithm for 1-var matching on concurrent traces based on multiset rewrite systems.
- These results have been extended to larger systems (fragments of CLF).
- Future work:
  - Same variable on both sides: $X[x \ y, \cdot] \equiv X[y \ x, \cdot]$
  - $n$-var unification
- Long-term objectives:
  - Reason about concurrent traces
Conclusions

- We presented an algorithm for 1-var matching on concurrent traces based on multiset rewrite systems.
- These results have been extended to larger systems (fragments of CLF).
- Future work
  - Same variable on both sides: $X[x \ y, \cdot] \equiv X[y \ x, \cdot]$
  - $n$-var unification
- Long-term objectives
  - Reason about concurrent traces

Thank you!