Trace Matching in a Concurrent Logical Framework

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

September 9, 2012

Concurrent Logical Framework

- LF is designed for encoding logics, deductive systems and programming languages
- However, it lacks feature to natively represent state, concurrency, or distributed computing
- Many extension have been developed to increase the expressive power of LF (LLF, HLF, OLF, ...)
- CLF is an extension of LF designed to specify concurrent and distributed systems

Syntax of CLF

$$
\begin{array}{lrcl}
\text{Kinds} & K & ::= & \Pi\Delta.\text{type} \\
\text{Types} & A & ::= & \Pi\Delta.a\cdot S \\
\text{Terms} & N & ::= & \lambda\Delta.M
\end{array}
$$

Syntax of CLF

$$
\begin{array}{rrcl}
\text{Kinds} & K & ::= & \Pi\Delta.\mathsf{type} \\
\text{Types} & A & ::= & \Pi\Delta.a{\cdot}S \mid \Pi\Delta.\{\Delta'\} \\
\text{Terms} & N & ::= & \lambda\Delta.M \mid \lambda\Delta.\{E\} \\
\textit{Contexts} & \Delta & ::= & \cdot \mid \Delta, !x{:}A \mid \Delta, @x{:}A \mid \Delta, \downarrow x{:}A
\end{array}
$$

Syntax of CLF

$$
\begin{array}{rrcl}
\text{Kinds} & K & ::= & \Pi\Delta.\mathsf{type} \\
\text{Types} & A & ::= & \Pi\Delta.a\cdot S \mid \Pi\Delta.\{\Delta'\} \\
\text{Terms} & N & ::= & \grave{} \\
\textit{Contexts} & \Delta & ::= & \cdot \mid \Delta, !x{:}A \mid \Delta, @x{:}A \mid \Delta, \downarrow x{:}A
\end{array}
$$

$$A \to B = !A \multimap B$$

Syntax of CLF

$$
\begin{array}{rrcl}
\text{Kinds} & K & ::= & \Pi\Delta.\mathsf{type} \\
\text{Types} & A & ::= & \Pi\Delta.a\cdot S \mid \Pi\Delta.\{\Delta'\} \\
\text{Terms} & N & ::= & \lambda\Delta.M \mid \lambda \\
\textit{Contexts} & \Delta & ::= & \cdot \mid \Delta, !x{:}A \mid \Delta, @x{:}A \mid \Delta, \downarrow x{:}A
\end{array}
$$

$$A \multimap @\ B = @A \multimap B$$

Syntax of CLF

$$
\begin{array}{rrcl}
\text{Kinds} & K & ::= & \Pi\Delta.\mathsf{type} \\
\text{Types} & A & ::= & \Pi\Delta.a{\cdot}S \mid \Pi\Delta.\{\Delta'\} \\
\text{Terms} & N & ::= & \lambda\Delta.M \mid \lambda\Delta.\{E\} \\
\textit{Contexts} & \Delta & ::= & \cdot \mid \Delta, !x{:}A \mid \Delta, @x{:}A \mid \Delta, \downarrow x{:}A
\end{array}
$$

$A \multimap B$

Syntax of CLF

$$
\begin{array}{rrcl}
\text{Kinds} & K & ::= & \Pi\Delta.\mathsf{type} \\
\text{Types} & A & ::= & \Pi\Delta.a{\cdot}S \mid \Pi\Delta.\{\Delta'\} \\
\text{Terms} & N & ::= & \lambda\Delta.M \mid \lambda\Delta.\{E\} \\
\textit{Contexts} & \Delta & ::= & \cdot \mid \Delta, !x{:}A \mid \Delta, @x{:}A \mid \Delta, \downarrow x{:}A \\
\text{Expressions} & E & ::= & \mathsf{let}\ \epsilon\ \mathsf{in}\ \Delta \\
\text{Traces} & \epsilon & ::= & \cdot \mid \{\Delta\} \leftarrow c{\cdot}S \mid \epsilon_1; \epsilon_2
\end{array}
$$

Syntax of CLF

$$
\begin{array}{rrcl}
\text{Kinds} & K & ::= & \Pi\Delta.\mathsf{type} \\
\text{Types} & A & ::= & \Pi\Delta.a{\cdot}S \mid \Pi\Delta.\{\Delta'\} \\
\text{Terms} & N & ::= & \lambda\Delta.M \mid \lambda\Delta.\{E\} \\
\textit{Contexts} & \Delta & ::= & \cdot \mid \Delta, !x{:}A \mid \Delta, @x{:}A \mid \Delta, \downarrow x{:}A \\
\text{Expressions} & E & ::= & \mathsf{let}\ \epsilon\ \mathsf{in}\ \Delta \\
\text{Traces} & \epsilon & ::= & \cdot \mid \{\Delta\} \leftarrow c{\cdot}S \mid \epsilon_1; \epsilon_2
\end{array}
$$

- The effects of a trace are encapsulated in a monad
  - Preserves canonical forms and adequacy of encodings
- Note: some features are missing, but they are orthogonal to the matching problem

Concurrent Traces

- CLF is based on the notion of concurrent trace

$$\epsilon ::= \cdot \mid \{\Delta\} \leftarrow c \cdot S \mid \epsilon_1; \epsilon_2$$

- 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
- CLF combines backward chaining (as LF) and forward chaining

Concurrent Traces

- CLF is based on the notion of concurrent trace

$$\epsilon ::= \cdot \mid \{\Delta\} \leftarrow c \cdot S \mid \epsilon_1; \epsilon_2$$

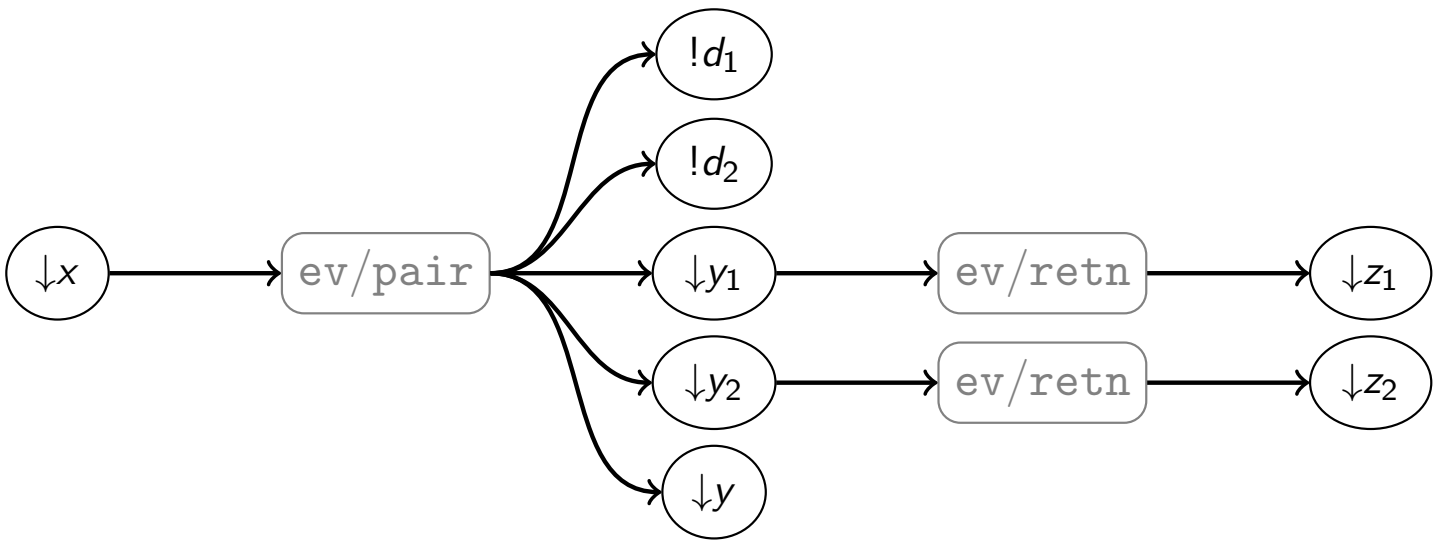
- A concurrent trace is a sequence of computational steps where independent steps can be permuted
- Computational traces define a model for concurrent and distributed computations
- CLF combines backward chaining (as LF) and forward chaining

- This work: we analyze the matching problem on concurrent traces (forward chaining fragment)

Concurrent traces

Example

$$!d{:}\mathtt{dest},\ !e_1, !e_2{:}\mathtt{exp},\ \downarrow x{:}\mathtt{eval}\cdot\langle e_1, e_2\rangle, d \ \vdash \left(\begin{array}{l} \{d_1, d_2, y_1, y_2, y\} \leftarrow ev/pair\cdot x; \\ \{z_1\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_1; \\ \{z_2\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_2 \end{array}\right) : \begin{array}{l} !d, !d_1, !d_2{:}\mathtt{dest}, \\ \downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1 \\ \downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1 \\ \downarrow y{:}\mathtt{cont}\cdot d_1, d_2, d \end{array}$$

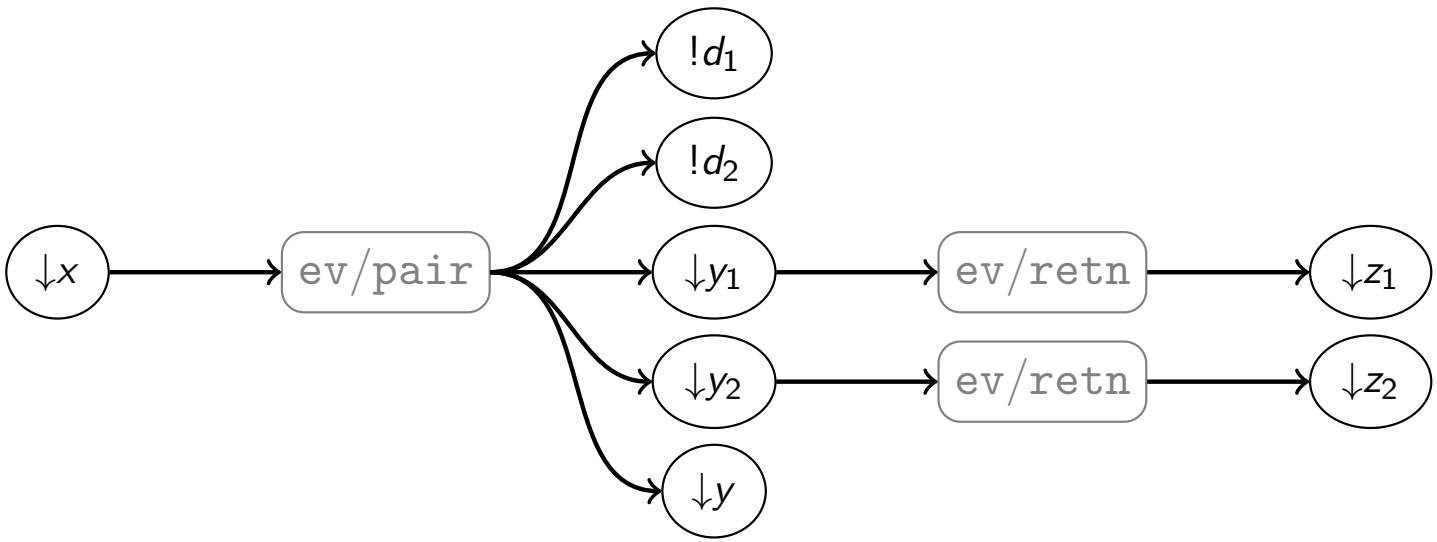


Concurrent traces

Example

$$\begin{array}{l} !d{:}\mathtt{dest}, \\ !e_1, !e_2{:}\mathtt{exp}, \\ \downarrow x{:}\mathtt{eval}\cdot\langle e_1, e_2\rangle, d \end{array} \vdash \left(\begin{array}{l} \{d_1, d_2, y_1, y_2, y\} \leftarrow ev/pair\cdot x; \\ \{z_1\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_1; \\ \{z_2\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_2 \end{array} \right) : \begin{array}{l} !d, !d_1, !d_2{:}\mathtt{dest}, \\ \downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1 \\ \downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1 \\ \downarrow y{:}\mathtt{cont}\cdot d_1, d_2, d \end{array}$$

$\mathtt{eval}\cdot e_1, d_1$

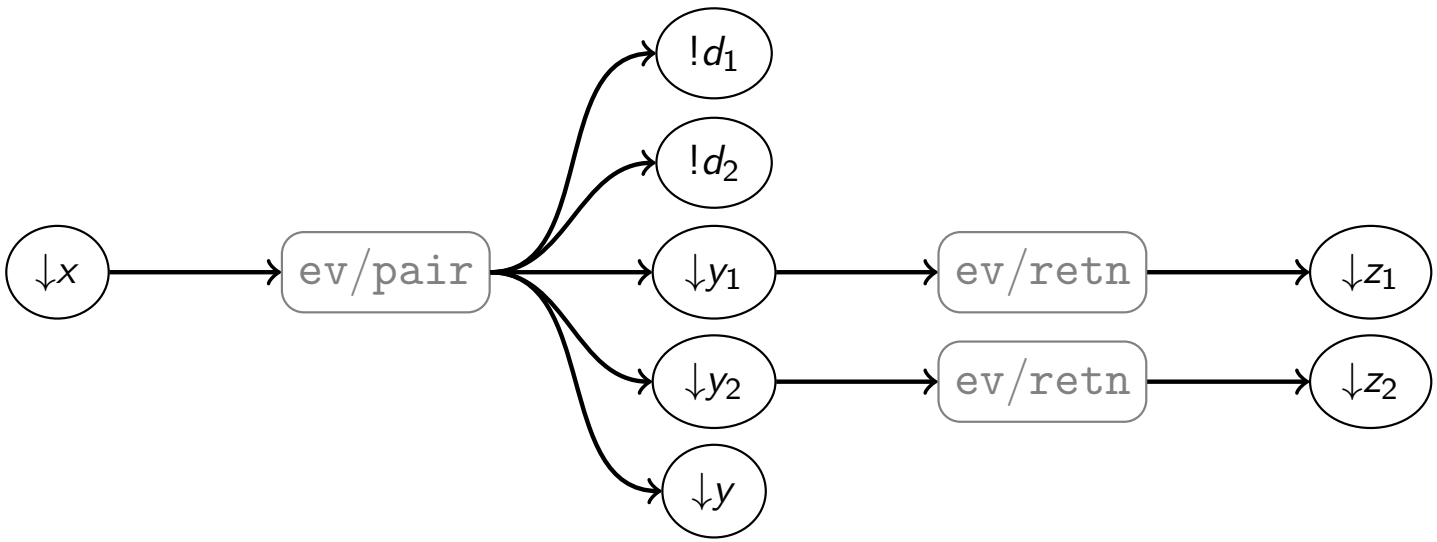


Concurrent traces

Example

$$
\begin{array}{l}
!d{:}\mathtt{dest},\\
!e_1, !e_2{:}\mathtt{exp},\\
\downarrow x{:}\mathtt{eval}\cdot\langle e_1, e_2\rangle, d
\end{array}
\vdash
\left(
\begin{array}{l}
\{d_1, d_2, y_1, y_2, y\} \leftarrow ev/pair\cdot x;\\
\{z_1\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_1;\\
\{z_2\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_2
\end{array}
\right)
:
\begin{array}{l}
!d, !d_1, !d_2{:}\mathtt{dest},\\
\downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1\\
\downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1\\
\downarrow y{:}\mathtt{cont}\cdot d_1, d_2, d
\end{array}
$$

$\mathtt{eval}\cdot e_2, d_2$

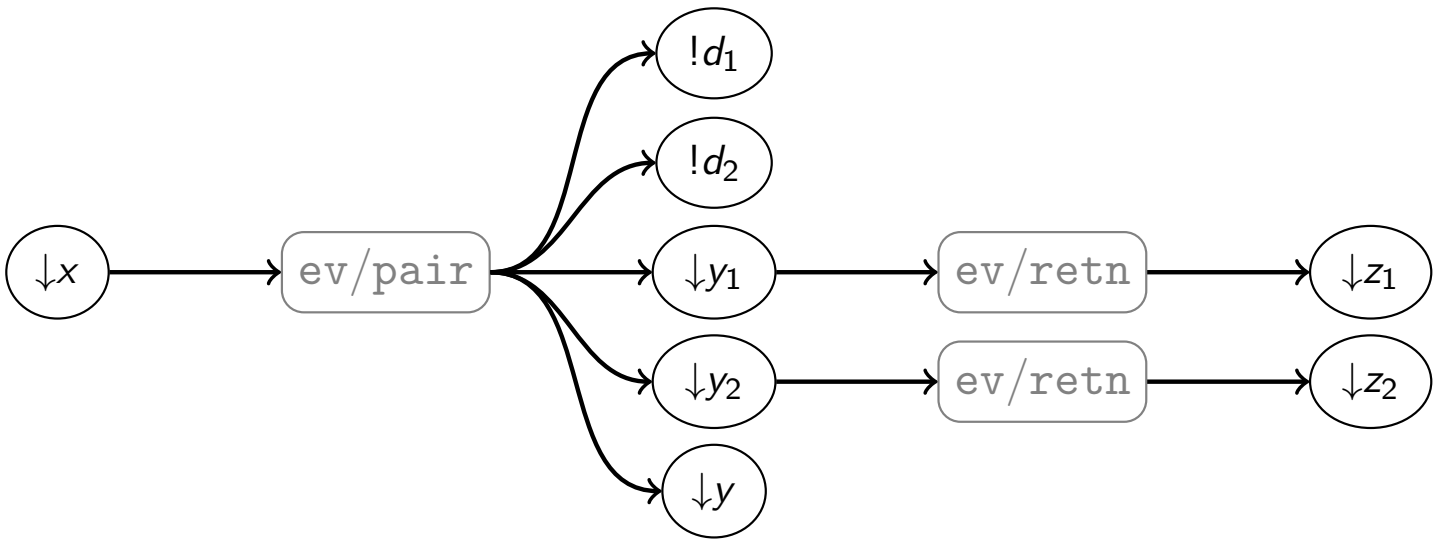


Concurrent traces

Example

$$
\begin{array}{l}
!d{:}\mathtt{dest}, \\
!e_1, !e_2{:}\mathtt{exp}, \\
\downarrow x{:}\mathtt{eval}\cdot\langle e_1, e_2\rangle, d
\end{array}
\quad \vdash \left(\begin{array}{l}
\{d_1, d_2, y_1, y_2, y\} \leftarrow ev/pair\cdot x; \\
\{z_1\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_1; \\
\{z_2\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_2
\end{array} \right) :
\begin{array}{l}
!d, !d_1, !d_2{:}\mathtt{dest}, \\
\downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1 \\
\downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1 \\
\downarrow y{:}\mathtt{cont}\cdot d_1, d_2, d
\end{array}
$$

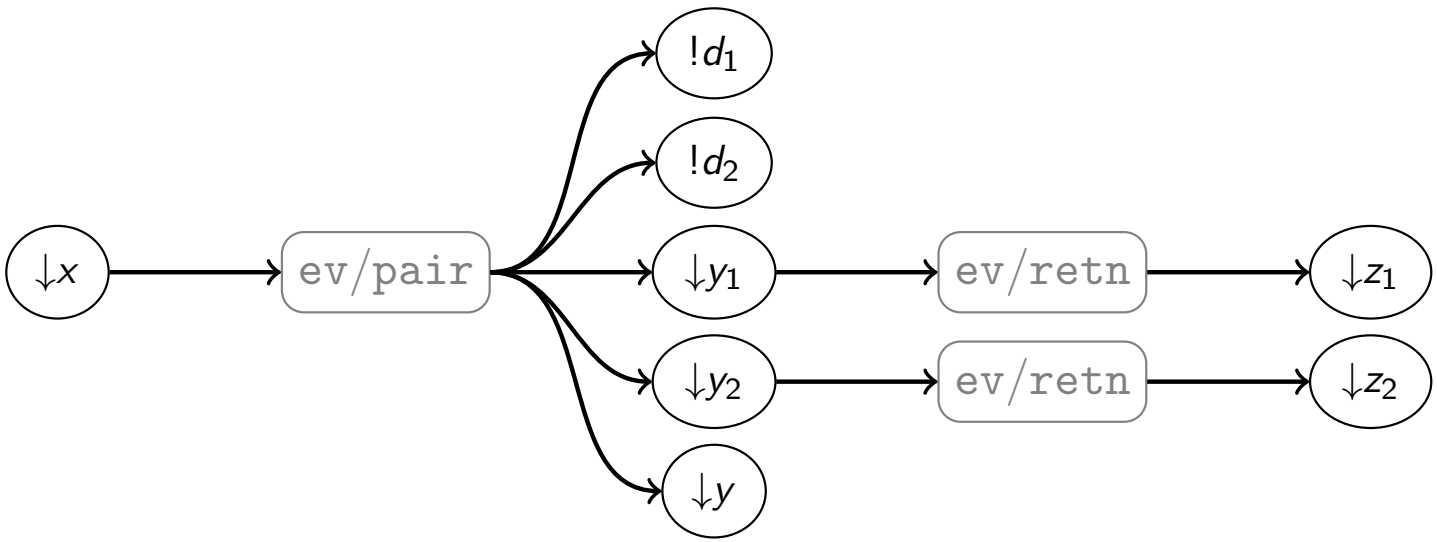
$\mathtt{cont}\cdot d_1, d_2, d$



Concurrent traces

Example

$$\begin{array}{l} !d{:}\mathtt{dest}, \\ !e_1, !e_2{:}\mathtt{exp}, \\ \downarrow x{:}\mathtt{eval}\cdot\langle e_1, e_2\rangle, d \end{array} \vdash \left(\begin{array}{l} \{d_1, d_2, y_1, y_2, y\} \leftarrow ev/pair\cdot x; \\ \{z_1\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_1; \\ \{z_2\} \leftarrow \mathtt{ev}/\mathtt{retn}\cdot y_2 \end{array} \right) : \begin{array}{l} !d, !d_1, !d_2{:}\mathtt{dest}, \\ \downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1 \\ \downarrow z_1{:}\mathtt{retn}\cdot v_1, d_1 \\ \downarrow y{:}\mathtt{cont}\cdot d_1, d_2, d \end{array}$$



Independent steps can be permuted

Concurrent traces

- Input and output interface:

$$\begin{cases} \bullet(\cdot) & = & \cdot \\ \bullet(\{\Delta\} \leftarrow c{\cdot}S) & = & \mathrm{FV}(S) \\ \bullet(\epsilon_1 ; \epsilon_2) & = & \bullet\epsilon_1 \cup (\bullet\epsilon_2 \setminus \epsilon_1\bullet) \end{cases}$$

$$\begin{cases} (\cdot)\bullet & = & \cdot \\ (\{\Delta\} \leftarrow c{\cdot}S)\bullet & = & \mathrm{dom}(\Delta) \\ (\epsilon_1 ; \epsilon_2)\bullet & = & (\epsilon_1\bullet \setminus \bullet\epsilon_2) \cup \epsilon_2\bullet \cup !(\epsilon_1\bullet) \end{cases}$$

Concurrent traces

- Trace independence

$$\epsilon_1 \parallel \epsilon_2 \iff \bullet\epsilon_1 \cap \epsilon_2\bullet = \epsilon_1\bullet \cap \bullet\epsilon_2 = \emptyset$$

- Equations

$$\frac{\epsilon_1 \equiv \epsilon_1'}{\epsilon_1;\epsilon_2 \equiv \epsilon_1';\epsilon_2} \qquad \frac{\epsilon_2 \equiv \epsilon_2'}{\epsilon_1;\epsilon_2 \equiv \epsilon_1;\epsilon_2'}$$

$$\frac{}{\epsilon_1;\cdot \equiv \epsilon_1} \qquad \frac{}{(\epsilon_1;\epsilon_2);\epsilon_3 \equiv \epsilon_1;(\epsilon_2;\epsilon_3)}$$

$$\frac{\epsilon_1 \parallel \epsilon_2}{\epsilon_1;\epsilon_2 \equiv \epsilon_2;\epsilon_1}$$

Concurrent traces

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

$$\{!d_1, !d_2, y_1, y_2, y\} \leftarrow \texttt{ev/pair}{\cdot}x; \quad \{z\} \leftarrow \texttt{ev/retn}{\cdot}y_1 \quad \equiv \quad \{!d_1, !d_2, y_1', y_2, y'\} \leftarrow \texttt{ev/pair}{\cdot}x; \quad \{z\} \leftarrow \texttt{ev/retn}{\cdot}y_1'$$

but

$$\{!d_1, !d_2, y_1, y_2, y\} \leftarrow \texttt{ev/pair}{\cdot}x; \quad \{z\} \leftarrow \texttt{ev/retn}{\cdot}y_1 \quad \not\equiv \quad \{!d_1, !d_2, y_1, y_2, y'\} \leftarrow \texttt{ev/pair}{\cdot}x; \quad \{z\} \leftarrow \texttt{ev/retn}{\cdot}y_1$$

Concurrent traces

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

$$\{!d_1, !d_2, y_1, y_2, y\} \leftarrow \texttt{ev/pair}{\cdot}x; \quad \{z\} \leftarrow \texttt{ev/retn}{\cdot}y_1 \quad \equiv \quad \{!d_1, !d_2, y_1', y_2, y'\} \leftarrow \texttt{ev/pair}{\cdot}x; \quad \{z\} \leftarrow \texttt{ev/retn}{\cdot}y_1'$$

but

$$\{!d_1, !d_2, y_1, y_2, y\} \leftarrow \texttt{ev/pair}{\cdot}x; \quad \{z\} \leftarrow \texttt{ev/retn}{\cdot}y_1 \quad \not\equiv \quad \{!d_1, !d_2, y_1, y_2, y'\} \leftarrow \texttt{ev/pair}{\cdot}x; \quad \{z\} \leftarrow \texttt{ev/retn}{\cdot}y_1$$

- But equality holds if we enclose the traces in a let

$$\begin{array}{l}\textsf{let}\\ \{!d_1, !d_2, y_1, y_2, y\} \leftarrow \texttt{ev/pair}{\cdot}x;\\ \{z\} \leftarrow \texttt{ev/retn}{\cdot}y_1\\ \textsf{in } y_1, y_2, y, z\end{array} \quad \equiv \quad \begin{array}{l}\textsf{let}\\ \{!d_1', !d_2', y_1, y_2, y'\} \leftarrow \texttt{ev/pair}{\cdot}x;\\ \{z'\} \leftarrow \texttt{ev/retn}{\cdot}y_1\\ \textsf{in } y_1, y_2, y, z'\end{array}$$

Concurrent traces

- Typing rule for traces represent state transformation

$$\Delta_1 \vdash \epsilon : \Delta_2$$

- Typing rules $\Delta_1 \vdash \epsilon : \Delta_2$

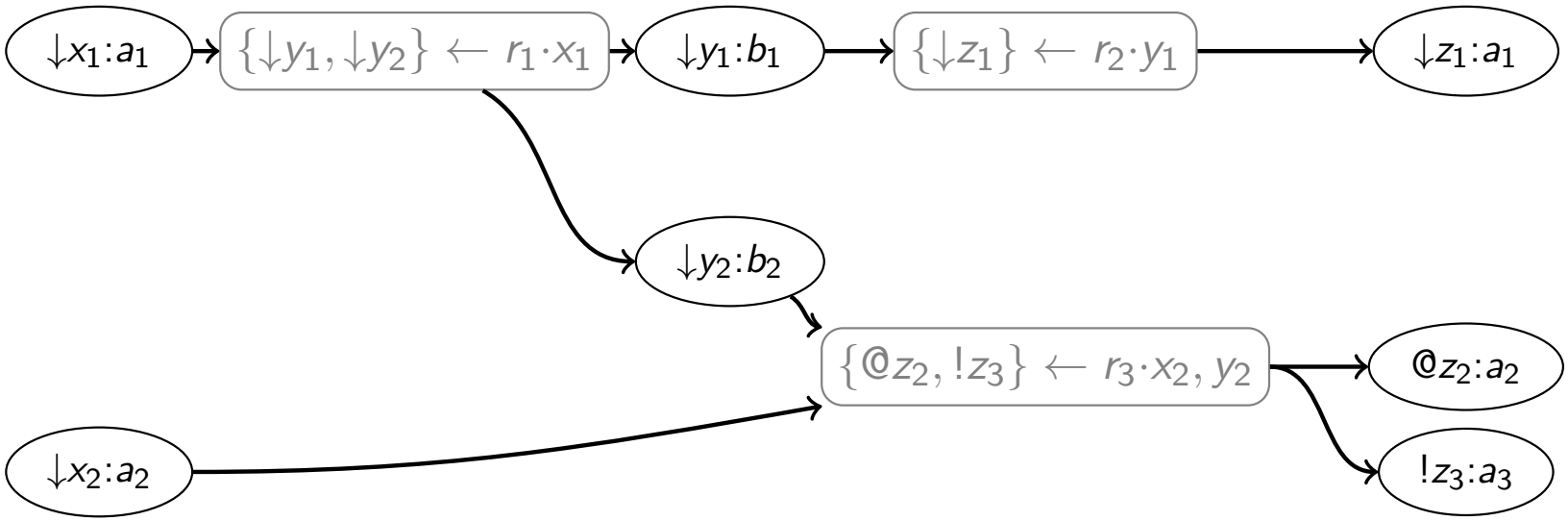
$$\frac{c{:}\Pi\Delta_c.\{\Delta_2\} \qquad \Delta_1 \vdash S : \Delta_c}{\Delta_0, \Delta_1 \vdash (\{\Delta_2\} \leftarrow c{\cdot}S) : \Delta_0, \Delta_2}$$

$$\frac{}{\Delta \vdash \cdot : \Delta} \qquad \frac{\Delta_0 \vdash \epsilon_1 : \Delta_1 \qquad \Delta_1 \vdash \epsilon_2 : \Delta_2}{\Delta_0 \vdash \epsilon_1; \epsilon_2 : \Delta_2}$$

$$\frac{\Delta \vdash \epsilon : \Delta_0 \qquad \Delta_0 \preccurlyeq \Delta}{\Delta \vdash \{\text{let } \epsilon \text{ in } \Delta\} : \{\Delta\}}$$

Concurrent traces

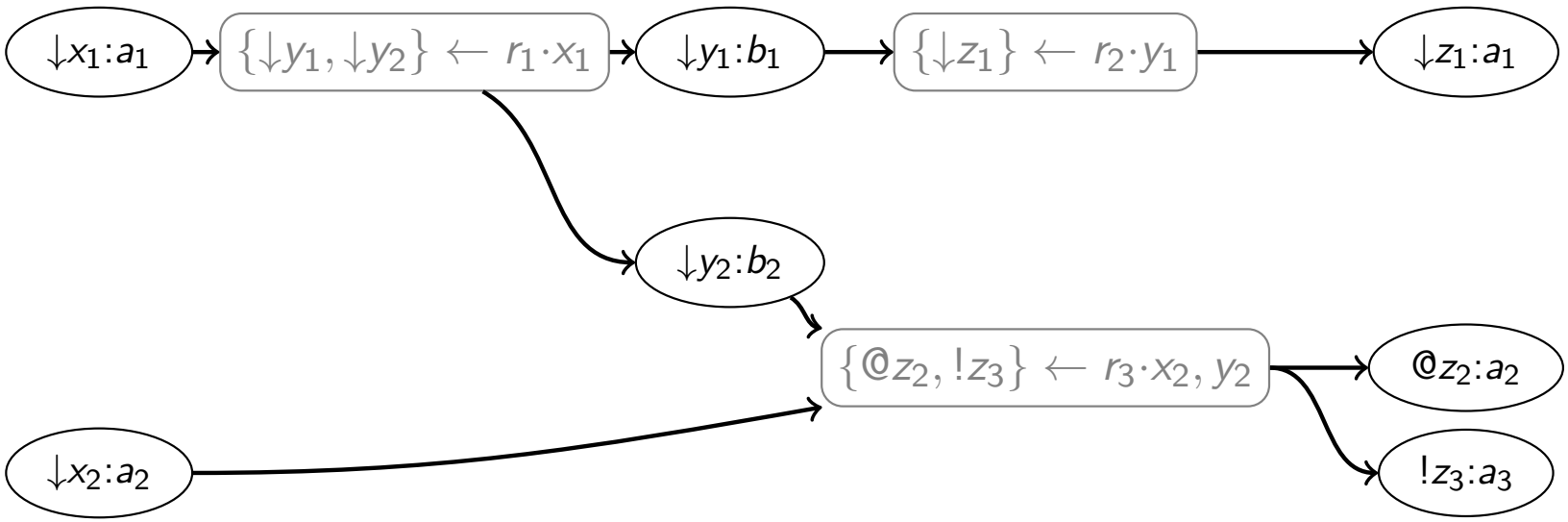
Typing



$$\downarrow x_1 : a_1,\ \downarrow x_2 : a_2 \ \vdash \ \left(\begin{array}{l} \{\downarrow y_1, \downarrow y_2\} \leftarrow r_1 \cdot x_1; \\ \{z_1\} \leftarrow r_2 \cdot y_1; \\ \{z_2, z_3\} \leftarrow r_3 \cdot x_2, y_2 \end{array} \right) \ : \ \begin{array}{l} \downarrow z_1 : a_1, \\ @z_2 : a_2, \\ !z_3 : a_3 \end{array}$$

Concurrent traces

Typing

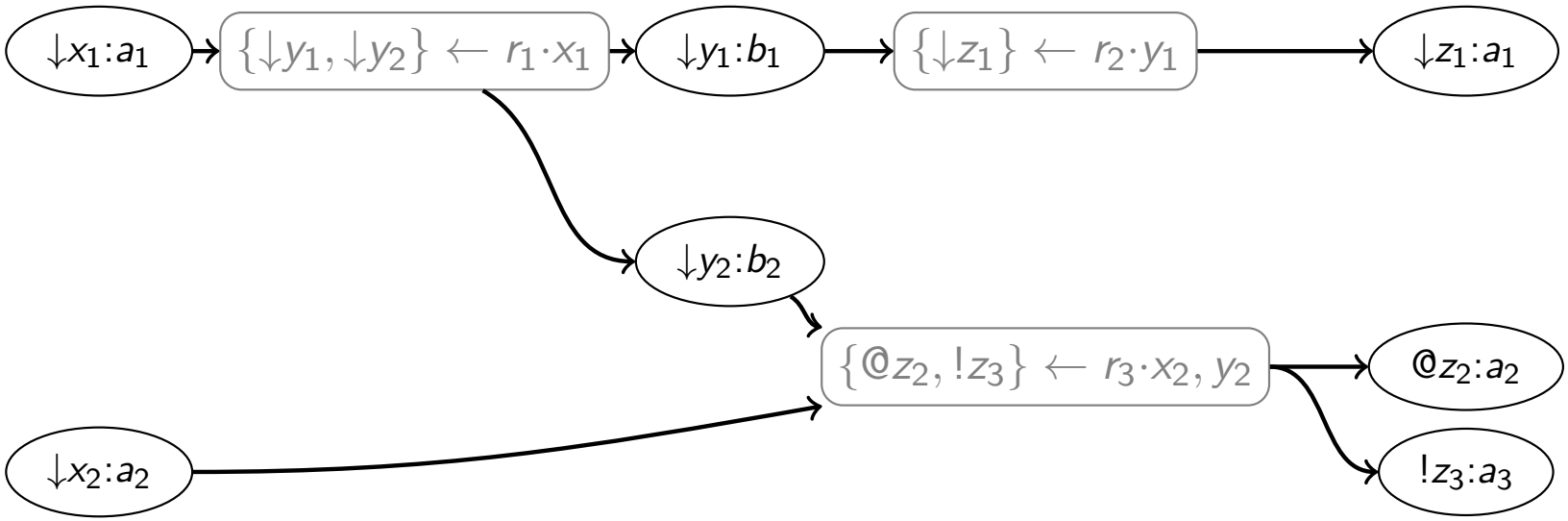


$$\downarrow x_1{:}a_1,\ \downarrow x_2{:}a_2 \ \vdash \text{let} \left(\begin{array}{l} \{\downarrow y_1, \downarrow y_2\} \leftarrow r_1 \cdot x_1; \\ \{z_1\} \leftarrow r_2 \cdot y_1; \\ \{z_2, z_3\} \leftarrow r_3 \cdot x_2, y_2 \end{array} \right) \text{in} \downarrow z_1, @z_2, !z_3 \ : \ \begin{array}{l} \downarrow z_1{:}a_1, \\ @z_2{:}a_2, \\ !z_3{:}a_3 \end{array}$$

$$\downarrow z_1{:}a_1, @z_2{:}a_2, !z_3{:}a_3 \preccurlyeq \downarrow z_1{:}a_1, @z_2{:}a_2, !z_3{:}a_3$$

Concurrent traces

Typing

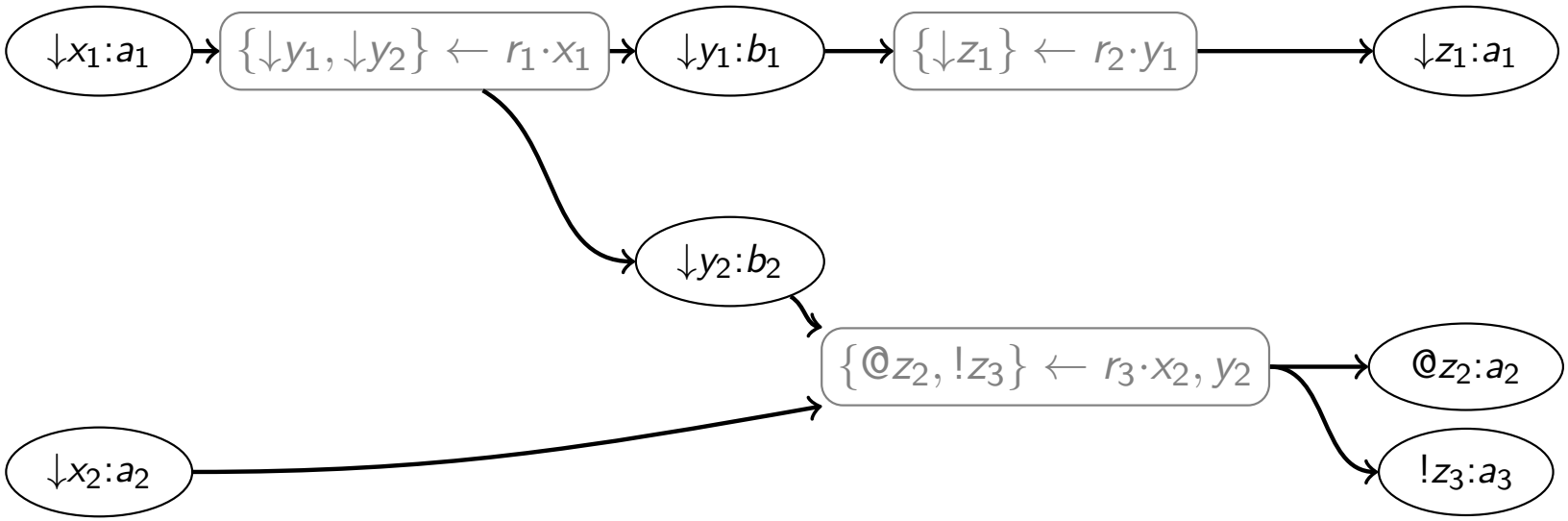


$$\downarrow x_1 : a_1, \downarrow x_2 : a_2 \vdash \text{let} \begin{pmatrix} \{\downarrow y_1, \downarrow y_2\} \leftarrow r_1 \cdot x_1; \\ \{z_1\} \leftarrow r_2 \cdot y_1; \\ \{z_2, z_3\} \leftarrow r_3 \cdot x_2, y_2 \end{pmatrix} \text{in} \downarrow z_1, \quad !z_3 \; : \; \begin{matrix} \downarrow z_1 : a_1, \\ \\ !z_3 : a_3 \end{matrix}$$

$$\downarrow z_1 : a_1, @z_2 : a_2, !z_3 : a_3 \preccurlyeq \downarrow z_1 : a_1, \quad !z_3 : a_3$$

Concurrent traces

Typing



$$\downarrow x_1{:}a_1,\ \downarrow x_2{:}a_2 \ \vdash \text{let} \left(\begin{array}{l} \{\downarrow y_1, \downarrow y_2\} \leftarrow r_1 \cdot x_1; \\ \{z_1\} \leftarrow r_2 \cdot y_1; \\ \{z_2, z_3\} \leftarrow r_3 \cdot x_2, y_2 \end{array} \right) \text{in} \downarrow z_1, \quad : \quad \downarrow z_1{:}a_1,$$

$$\downarrow z_1{:}a_1, @z_2{:}a_2, !z_3{:}a_3 \preccurlyeq \downarrow z_1{:}a_1,$$

Long-term objectives

- Develop a logical framework to reason about concurrent traces
- Unification and matching are prerequisites
- This work: matching on concurrent traces (with one variable standing for an unknown trace)
- Matching can also be used as the basis for a moded semantics

Equations over traces

- Trace variables

$$\epsilon ::= \cdot \mid \{\Delta\} \leftarrow c{\cdot}S \mid \epsilon_1; \epsilon_2 \mid \{\Delta\} \leftarrow X[\theta]$$

- θ and Δ represent the input and output interface of X
- X can be instantiated with a trace ϵ such that $\bullet\epsilon \succcurlyeq \mathrm{FV}(\theta)$ and $\epsilon\bullet \preccurlyeq \mathrm{dom}(\Delta)$
- An equation is given by a pair of traces containing variables well typed under the same context

$$\Delta \vdash \epsilon_1 \stackrel{?}{=} \epsilon_2$$

Matching on traces

- Given

$$\Delta \vdash \epsilon_1 \stackrel{?}{=} \epsilon_2 \qquad \text{with } \epsilon_2 \text{ ground}$$

Find an assignment for the logic variables in ϵ_1

$$\rho = X_1 \leftarrow \epsilon_1, \ldots, X_n \leftarrow \epsilon_n$$

such that

$$\rho\epsilon_1 \equiv \epsilon_2$$

Matching on traces

- Given

$$\Delta \vdash \epsilon_1 \stackrel{?}{=} \epsilon_2 \qquad \text{with } \epsilon_2 \text{ ground}$$

  Find an assignment for the logic variables in ϵ_1

$$\rho = X_1 \leftarrow \epsilon_1, \ldots, X_n \leftarrow \epsilon_n$$

  such that

$$\rho\epsilon_1 \equiv \epsilon_2$$

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

$$\{\cdot\} \leftarrow X[\cdot]; \{\cdot\} \leftarrow Y[\cdot] \stackrel{?}{=} \{\cdot\} \leftarrow c_1; \ldots; \{\cdot\} \leftarrow$$

  Partition $\{c_1, \ldots, c_n\}$ in two sets X and Y.

Matching on traces

- Matching is decidable:

$$\epsilon_1; \{\Delta\} \leftarrow X[\theta]; \epsilon_1' \quad \stackrel{?}{=} \quad \epsilon_2$$

  X must be instantiated with a subtrace of ϵ_2
- In a problem

$$\epsilon_1^1; (\{\Delta_1\} \leftarrow X_1[\theta_1]); \epsilon_2^1; \ldots; (\{\Delta_1\} \leftarrow X_1[\theta_1]); \epsilon_n^1 \quad \stackrel{?}{=} \quad \epsilon_2$$

  we can try all possible combinations of subtraces of ϵ_2

Matching on traces

- Matching is decidable:

$$\epsilon_1; \{\Delta\} \leftarrow X[\theta]; \epsilon_1' \quad \stackrel{?}{=} \quad \epsilon_2$$

  X must be instantiated with a subtrace of ϵ_2
- In a problem

$$\epsilon_1^1; (\{\Delta_1\} \leftarrow X_1[\theta_1]); \epsilon_2^1; \ldots; (\{\Delta_1\} \leftarrow X_1[\theta_1]); \epsilon_n^1 \quad \stackrel{?}{=} \quad \epsilon_2$$

  we can try all possible combinations of subtraces of ϵ_2
- Expensive

Matching on traces

- We consider 1-variable matching

$$\epsilon_1; \{\Delta_1\} \leftarrow X_1[\theta_1]; \epsilon_1' \quad \stackrel{?}{=} \quad \epsilon_2$$

where ϵ_1, ϵ_1' and ϵ_2 are ground

Matching on traces

- We consider 1-variable matching

$$\epsilon_1; \{\Delta_1\} \leftarrow X_1[\theta_1]; \epsilon_1' \quad \stackrel{?}{=} \quad \epsilon_2$$

  where ϵ_1, ϵ_1' and ϵ_2 are ground
- Reduces search space

Matching on traces

Intuition of the algorithm

- Match individual known computation steps until we are left with the logic variable, reordering traces if necessary

$$\Delta \vdash \{\Delta_0\} \leftarrow c \cdot S; \epsilon_1 \quad \stackrel{?}{=} \quad \epsilon_2$$

Matching on traces

Intuition of the algorithm

- Match individual known computation steps until we are left with the logic variable, reordering traces if necessary

$$\Delta \vdash \{\Delta_0\} \leftarrow c{\cdot}S; \epsilon_1 \quad \stackrel{?}{=} \quad \{\Delta_0'\} \leftarrow c{\cdot}S'; \epsilon_2'$$

Matching on traces

Intuition of the algorithm

- Match individual known computation steps until we are left with the logic variable, reordering traces if necessary

$$\Delta \vdash \{\Delta_0\} \leftarrow c \cdot S; \epsilon_1 \quad \stackrel{?}{=} \quad \{\Delta_0'\} \leftarrow c \cdot S'; \epsilon_2'$$

- Compute partial renamings at each step

Matching on traces

Intuition of the algorithm

- Match individual known computation steps until we are left with the logic variable, reordering traces if necessary

$$\Delta \vdash \{\Delta_0\} \leftarrow c \cdot S; \epsilon_1 \quad \stackrel{?}{=} \quad \{\Delta_0'\} \leftarrow c \cdot S'; \epsilon_2'$$

- Compute partial renamings at each step

$$\Delta \vdash \left(\begin{array}{l} \{\downarrow x_1\} \leftarrow c \\ \{!y_1\} \leftarrow c' \cdot x_1 \end{array} \right) \stackrel{?}{=} \left(\begin{array}{l} \{\downarrow x_2\} \leftarrow c \\ \{!y_2\} \leftarrow c' \cdot x_2 \end{array} \right)$$

Matching on traces

Intuition of the algorithm

- Match individual known computation steps until we are left with the logic variable, reordering traces if necessary

$$\Delta \vdash \{\Delta_0\} \leftarrow c \cdot S; \epsilon_1 \stackrel{?}{=} \{\Delta_0'\} \leftarrow c \cdot S'; \epsilon_2'$$

- Compute partial renamings at each step

$$\Delta \vdash \left(\begin{array}{l} \{\downarrow x_1\} \leftarrow c \\ \{!y_1\} \leftarrow c' \cdot x_1 \end{array} \right) \stackrel{?}{=} \left(\begin{array}{l} \{\downarrow x_2\} \leftarrow c \\ \{!y_2\} \leftarrow c' \cdot x_2 \end{array} \right)$$

Matching on traces

Intuition of the algorithm

- Match individual known computation steps until we are left with the logic variable, reordering traces if necessary

$$\Delta \vdash \{\Delta_0\} \leftarrow c \cdot S; \epsilon_1 \quad \stackrel{?}{=} \quad \{\Delta_0'\} \leftarrow c \cdot S'; \epsilon_2'$$

- Compute partial renamings at each step

$$\Delta \vdash \left(\begin{array}{l} \{\downarrow x_1\} \leftarrow c \\ \{!y_1\} \leftarrow c' \cdot x_1 \end{array} \right) \stackrel{?}{=} \left(\begin{array}{l} \{\downarrow x_2\} \leftarrow c \\ \{!y_2\} \leftarrow c' \cdot x_2 \end{array} \right)$$

$$\varphi = y/y_1, x/x_1; \quad \varphi' = y/y_2, x/x_2$$

Matching on traces

Intuition of the algorithm

- Match individual known computation steps until we are left with the logic variable, reordering traces if necessary

$$\Delta \vdash \{\Delta_0\} \leftarrow c \cdot S; \epsilon_1 \quad \stackrel{?}{=} \quad \{\Delta_0'\} \leftarrow c \cdot S'; \epsilon_2'$$

- Compute partial renamings at each step

$$\Delta \vdash \begin{pmatrix} \{\downarrow x_1\} \leftarrow c \\ \{!y_1\} \leftarrow c' \cdot x_1 \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} \{\downarrow x_2\} \leftarrow c \\ \{!y_2\} \leftarrow c' \cdot x_2 \end{pmatrix}$$

$$\varphi = y/y_1, x/x_1; \quad \varphi' = y/y_2, x/x_2$$

- Variables are marked to ensure they are not renamed twice

$$\Delta \vdash \begin{pmatrix} \epsilon_1; \\ \{\cdot\} \leftarrow c \cdot !x_1 \\ \{\cdot\} \leftarrow c \cdot !x_1 \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} \epsilon_2; \\ \{\cdot\} \leftarrow c \cdot !y_1 \\ \{\cdot\} \leftarrow c \cdot !z_1 \end{pmatrix}$$

Algorithm

- Matching at the head

$$\Delta \vdash \left(\begin{array}{l} \{\downarrow y_1\} \leftarrow c \cdot X[\theta], a' \\ \{!z_1\} \leftarrow Y[y_1] \end{array} \right) \stackrel{?}{=} \left(\begin{array}{l} \{\downarrow y_2\} \leftarrow c \cdot a, a' \\ \{!z_2\} \leftarrow c' \cdot y_1 \end{array} \right)$$

Algorithm

- Matching at the head

$$\Delta \vdash \begin{pmatrix} \{\downarrow y_1\} \leftarrow c \cdot X[\theta], a' \\ \{!z_1\} \leftarrow Y[y_1] \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} \{\downarrow y_2\} \leftarrow c \cdot a, a' \\ \{!z_2\} \leftarrow c' \cdot y_1 \end{pmatrix}$$

$$(X \leftarrow \theta^{-1} a); \quad y/y_1; \quad y/y_2$$

Algorithm

- Matching at the head

$$\Delta \vdash \left(\ \{!z_1\} \leftarrow X[y] \ \right) \stackrel{?}{=} \left(\ \{!z_2\} \leftarrow c' \cdot y \ \right)$$

$$(X \leftarrow \theta^{-1} a); \quad y/y_1; \quad y/y_2$$

Algorithm

- Matching at the head

$$\Delta \vdash \left(\ \{!z_1\} \leftarrow X[y] \ \right) \stackrel{?}{=} \left(\ \{!z_2\} \leftarrow c' \cdot y \ \right)$$

$$(X \leftarrow \theta^{-1} a); \quad y/y_1; \quad y/y_2$$

- Rule step-head

$$\frac{(\Delta_{S_1} \vdash S_1) \stackrel{?}{=} (\Delta_{S_2} \vdash S_2) \mapsto \sigma \qquad \sigma\Delta_1 \stackrel{?}{=} \Delta_2 \mapsto \varphi_1; \varphi_2 \qquad \Delta' \vdash \sigma\varphi_1\epsilon_1 \stackrel{?}{=} \varphi_2\epsilon_2 \mapsto \sigma'; \varphi_1'; \varphi_2'}{\Delta \vdash (\{\Delta_1\} \leftarrow c \cdot S_1; \epsilon_1) \stackrel{?}{=} (\{\Delta_2\} \leftarrow c \cdot S_2; \epsilon_2) \mapsto \sigma\sigma'; \varphi_1\varphi_1'; \varphi_2\varphi_2'}$$

Algorithm

- Matching at the tail

$$\Delta \vdash \left(\begin{array}{l} \{y_1, y_2\} \leftarrow X[a]; \\ \{@z_1\} \leftarrow c' \cdot Y[y_1], y_2 \end{array} \right) \stackrel{?}{=} \left(\begin{array}{l} \{y_1', y_2'\} \leftarrow c \cdot a; \\ \{@z_2\} \leftarrow c' \cdot (c_0 \cdot y_1'), y_2' \end{array} \right)$$

Algorithm

- Matching at the tail

$$\Delta \vdash \left(\begin{array}{l} \{y_1, y_2\} \leftarrow X[a]; \\ \{@z_1\} \leftarrow c' \cdot Y[y_1], y_2 \end{array} \right) \stackrel{?}{=} \left(\begin{array}{l} \{y'_1, y'_2\} \leftarrow c \cdot a; \\ \{@z_2\} \leftarrow c' \cdot (c_0 \cdot y'_1), y'_2 \end{array} \right)$$
$$Y[x] \leftarrow c_0 \cdot [x]; \quad y^1/y_1, y^2/y_2; \quad y^1/y'_1, y^2/y'_2$$

Algorithm

- Matching at the tail

$$\Delta \vdash \left(\ \{y^1, y^2\} \leftarrow X[a] \ \right) \stackrel{?}{=} \left(\ \{y^1, y^2\} \leftarrow c{\cdot}a \ \right)$$

$$Y[x] \leftarrow c_0{\cdot}[x]; \quad y^1/y_1, y^2/y_2; \quad y^1/y'_1, y^2/y'_2$$

Algorithm

- Matching at the tail

$$\Delta \vdash \left(\ \{y^1, y^2\} \leftarrow X[a] \ \right) \stackrel{?}{=} \left(\ \{y^1, y^2\} \leftarrow c{\cdot}a \ \right)$$

$$Y[x] \leftarrow c_0{\cdot}[x]; \quad y^1/y_1, y^2/y_2; \quad y^1/y'_1, y^2/y'_2$$

- Rule step-tail

$$\frac{\begin{array}{c}(\Delta_{S_1} \vdash S_1) \stackrel{?}{=} (\Delta_{S_2} \vdash S_2) \mapsto \sigma; \varphi_1; \varphi_2 \\ \sigma\Delta_1 \stackrel{?}{=} \Delta_2 \mapsto \varphi'_1; \varphi'_2 \\ \Delta' \vdash \sigma\epsilon_1\{\varphi_1\} \stackrel{?}{=} \epsilon_2\{\varphi_2\} \mapsto \sigma'; \varphi''_1; \varphi''_2\end{array}}{\Delta \vdash (\epsilon_1; \{\Delta_1\} \leftarrow c{\cdot}S_1) \stackrel{?}{=} (\epsilon_2; \{\Delta_2\} \leftarrow c{\cdot}S_2) \mapsto \sigma\sigma'; \varphi_1\varphi'_1\varphi''_1; \varphi_2\varphi'_2\varphi''_2}$$

Algorithm

- Last step: logic variable

$$\Delta \vdash \left(\ \{@z\} \leftarrow X[a] \ \right) \stackrel{?}{=} \left(\begin{array}{l} \{!y\} \leftarrow c \cdot a; \\ \{@z_1\} \leftarrow c' \cdot !y \\ \{@z_2\} \leftarrow c' \cdot !y \end{array} \right)$$

Algorithm

- Last step: logic variable

$$\Delta \vdash \left(\ \{@z\} \leftarrow X[a] \ \right) \stackrel{?}{=} \left(\begin{array}{l} \{!y\} \leftarrow c{\cdot}a; \\ \{@z_1\} \leftarrow c'{\cdot}!y \\ \{@z_2\} \leftarrow c'{\cdot}!y \end{array} \right)$$

$$X \leftarrow \{\text{let } \{!y\} \leftarrow c{\cdot}a; \{@z_1\} \leftarrow c'{\cdot}!y; \{@z_2\} \leftarrow c'{\cdot}!y \text{ in } @z_1\}$$

Algorithm

- Last step: logic variable

$$\Delta \vdash \left(\ \{@z\} \leftarrow X[a] \ \right) \stackrel{?}{=} \left(\begin{array}{l} \{!y\} \leftarrow c{\cdot}a; \\ \{@z_1\} \leftarrow c'{\cdot}!y \\ \{@z_2\} \leftarrow c'{\cdot}!y \end{array} \right)$$

$$X \leftarrow \{\text{let } \{!y\} \leftarrow c{\cdot}a; \{@z_1\} \leftarrow c'{\cdot}!y; \{@z_2\} \leftarrow c'{\cdot}!y \text{ in } @z_1\}$$

$$\text{or} \quad X \leftarrow \{\text{let } \{!y\} \leftarrow c{\cdot}a; \{@z_1\} \leftarrow c'{\cdot}!y; \{@z_2\} \leftarrow c'{\cdot}!y \text{ in } @z_2\}$$

Algorithm

- Last step: logic variable

$$\Delta \vdash \left(\ \{@z\} \leftarrow X[a] \ \right) \stackrel{?}{=} \left(\begin{array}{l} \{!y\} \leftarrow c{\cdot}a; \\ \{@z_1\} \leftarrow c'{\cdot}!y \\ \{@z_2\} \leftarrow c'{\cdot}!y \end{array} \right)$$

$$X \leftarrow \{\text{let } \{!y\} \leftarrow c{\cdot}a; \{@z_1\} \leftarrow c'{\cdot}!y; \{@z_2\} \leftarrow c'{\cdot}!y \text{ in } @z_1\}$$

or $\quad X \leftarrow \{\text{let } \{!y\} \leftarrow c{\cdot}a; \{@z_1\} \leftarrow c'{\cdot}!y; \{@z_2\} \leftarrow c'{\cdot}!y \text{ in } @z_2\}$

- Rule inst

$$\frac{\Delta \vdash \epsilon : \Delta' \qquad \Delta_0 \gtrdot \Delta' \mapsto \varphi_1; \varphi_2}{\Delta \vdash (\{\Delta_0\} \leftarrow X[\theta]) \stackrel{?}{=} \epsilon \mapsto (X \leftarrow \theta^{-1}\{\text{let } \epsilon \text{ in } \Delta_0\}); \varphi_1; \varphi_2}$$

Correctness of the algorithm

Soundness

If

$$\Delta \vdash (\epsilon_1; \Delta_0 \leftarrow X[\theta]; \epsilon_1') \stackrel{?}{=} \epsilon_2 \mapsto \sigma; \varphi_1; \varphi_2$$

then $\sigma\epsilon_1\{\varphi_1\} \equiv \epsilon_2\{\varphi_2\}$.

Proof

By induction on the algorithm derivation. (We rely on soundness of higher-order matching for the pattern fragment.)

Correctness of the algorithm

Completeness

If there exists σ, φ_1, and φ_2 such that

$$\sigma\epsilon_1\{\varphi_1\} \equiv \epsilon_2\{\varphi_2\}$$

then there exists φ_1' and φ_2' such that then

$$\Delta \vdash (\epsilon_1; \Delta_0 \leftarrow X[\theta]; \epsilon_1') \stackrel{?}{=} \epsilon_2 \mapsto \sigma; \varphi_1'; \varphi_2'$$

Proof

By induction on the size of the trace. (We rely on completeness of higher-order matching for the pattern fragment.)

Conclusions

- We presented an algorithm for 1-var matching on concurrent traces based on CLF
- Implementation in `celf` is under way
- Future work
  - Unification
  - Current results: 1-var unification for simply-typed fragment of CLF:

    $$\epsilon_1; X_1; \epsilon_1' \stackrel{?}{=} \epsilon_2; X_2; \epsilon_2'$$

    Includes affine and persistent functions, but no dependencies
  - Same variable on both sides: $X[x, y] \stackrel{?}{=} X[y, x]$
- Long-term objectives
  - Reason about concurrent traces

Conclusions

- We presented an algorithm for 1-var matching on concurrent traces based on CLF
- Implementation in `celf` is under way
- Future work
  - Unification
  - Current results: 1-var unification for simply-typed fragment of CLF:

    $$\epsilon_1; X_1; \epsilon_1' \stackrel{?}{=} \epsilon_2; X_2; \epsilon_2'$$

    Includes affine and persistent functions, but no dependencies
  - Same variable on both sides: $X[x, y] \stackrel{?}{=} X[y, x]$
- Long-term objectives
  - Reason about concurrent traces

Thank you!