# A Declarative Validator for GSOS Languages

Matteo Cimini
University of Massachusetts Lowell

April, 22nd 2023
PLACES 2023, Paris, France

# Language Validation

type sound?
strongly normalizing?
data-race free?
is bisim a congruence?


Expression $e \quad ::= \mid x \mid \lambda x : T.e \mid e\ e$

$\ldots$

$$(\lambda x : T.e)\ v \longrightarrow e[v/x]$$

$\texttt{if true then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_1$

$\texttt{if false then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_2$

$\ldots$

# An Example from functional PL: Lang-n-Check [*]

$$(\lambda x : T.e)\ v \longrightarrow e[v/x]$$

needs to be evaluated to a value

EvalCtx $\quad E ::= \square \mid \texttt{if}\ E\ \texttt{then}\ e\ \texttt{else}\ e \mid E\ e \mid v\ E \quad \ldots$

Then: Lang-n-Check checks that this evaluation context exists

[*] *Extrinsically Typed Operational Semantics for Functional Languages*. **SLE 2020**
**Matteo Cimini, Dale Miller, Jeremy Siek**

# An Example from process algebra: GSOS Rule Format [**]

**Sources of premises:**
- **only variables**
- **only from the arguments of op**

**Targets of premises:**
- **only variables**
- **ys distinct from xs**
- **ys pairwise distinct**

$$\frac{\{x_i \longrightarrow^{l_{ij}} y_{ij}\}}{(op\ x_1 \ldots x_n) \longrightarrow^{l} t}$$

**[**] among other restrictions that we shall see.**

# Rule Formats

- *strong bisimilarity is a congruence*
- *weak variants of bisimilarity are a congruence*
- *validity of algebraic law for operators*
- *determinism vs bounded non-determinism*
- *applied to binders*
- *applied to probabilistic transition systems*

# Rule Formats

## Our question:

*Can we identify a DSL that can express rule formats*

- *concisely*
- *declaratively*
- *can be applied to a test suite of process algebras at once*

## In this paper:

- *Explore the use of an existing DSL:* **Lang-n-Change** *(tailored for transforming operational semantics)*
- *Write a GSOS Validator in* **Lang-n-Change**
- *Evaluate our validator on 18 process algebra operators and negative tests*

# Language Transformations: Subtyping example

$$\frac{\Gamma \vdash\ e_1 : T_1 \to T_2 \qquad \Gamma \vdash\ e_2 : T_1}{\Gamma \vdash\ e_1\ e_2 : T_2} \quad (\text{T-APP})$$

**Intuitively:**

$f((\text{T-APP})) = (\text{T-APP'})$

$$\frac{\Gamma \vdash\ e_1 : T_{11} \to T_2 \qquad \Gamma \vdash\ e_2 : T_{12} \qquad T_{12} <: T_{11}}{\Gamma \vdash\ e_1\ e_2 : T_2} \quad (\text{T-APP'})$$

# Language Transformations: Subtyping example

$$\frac{\Gamma \vdash\ e_1 : \boxed{T_1 \to T_2} \qquad \Gamma \vdash\ e_2 : \boxed{T_1}}{\Gamma \vdash\ e_1\ e_2 : T_2}\ (\text{T-APP})$$

($T_{11}$ and $T_{12}$ point to the boxed $T_1$ and $T_1$ respectively)

**Intuitively:**

$f((\text{T-APP})) = (\text{T-APP'})$

$$\frac{\Gamma \vdash\ e_1 : T_{11} \to T_2 \qquad \Gamma \vdash\ e_2 : T_{12} \qquad T_{12} <: T_{11}}{\Gamma \vdash\ e_1\ e_2 : T_2}\ (\text{T-APP'})$$

## Step #1: get rid of type equality

# Language Transformations: Subtyping example

$$\frac{\Gamma \vdash\ e_1 : T_1 \to T_2 \qquad \Gamma \vdash\ e_2 : T_1 \qquad T_{12} <: T_{11}}{\Gamma \vdash\ e_1\ e_2 : T_2}\ (\text{T-APP})$$

$T_{11}$ $T_{12}$

contravariant

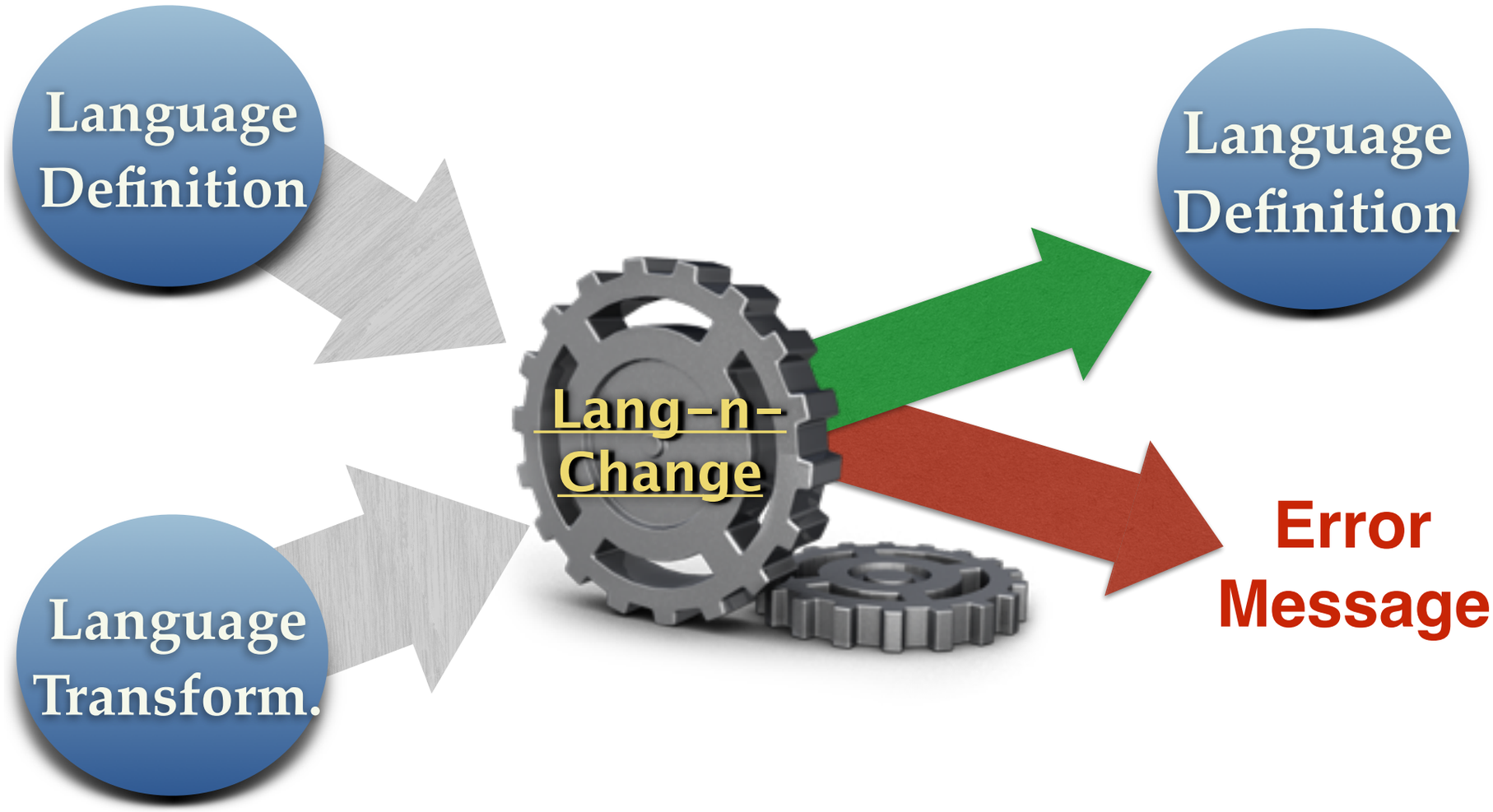
**Intuitively:**

$f((\text{T-APP})) = (\text{T-APP'})$

$$\frac{\Gamma \vdash\ e_1 : T_{11} \to T_2 \qquad \Gamma \vdash\ e_2 : T_{12} \qquad T_{12} <: T_{11}}{\Gamma \vdash\ e_1\ e_2 : T_2}\ (\text{T-APP'})$$

**Step # 2: relate new *T*s with subtyping according to variance**

# Pipeline of Lang-n-Change




**Lang-n-Change provides a DSL to express these transformations**
**Includes operations to interrogate/test operational semantics**

# Lang-n-Change: Syntax

$$
\begin{array}{llll}
\text{Expression} & e & ::= & x \mid str \mid t \mid [e \ldots e] \mid \texttt{head}\ e \mid \texttt{tail}\ e \mid e@e \mid e - e \mid \texttt{map}(e,e) \mid e(e) \\
& & & \mid \texttt{rules} \mid \texttt{premises} \mid \texttt{conclusion} \mid \texttt{self} \\
& & & \mid e[p] : e \mid \texttt{uniquefy}(e) \Rightarrow (x,x) : e \mid \texttt{getVars}(e) \\
& & & \mid \texttt{if}\ b\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{skip} \mid \texttt{error}\ str \\
\text{Boolean Expr.} & b & ::= & e = e \mid \texttt{isVar}(e) \mid b\ \texttt{and}\ b \mid b\ \texttt{or}\ b \mid \texttt{not}\ b
\end{array}
$$

***lists with common operations (including append and list difference)***

# Lang-n-Change: Syntax

$$
\begin{array}{lrll}
\text{Expression} & e & ::= & x \mid str \mid t \mid [e \ \ldots \ e] \mid \texttt{head}\ e \mid \texttt{tail}\ e \mid e@e \mid e - e \mid \texttt{map}(e,e) \mid e(e) \\
& & & \mid \texttt{rules} \mid \texttt{premises} \mid \texttt{conclusion} \mid \texttt{self} \\
& & & \mid e[p] : e \mid \texttt{uniquefy}(e) \Rightarrow (x,x) : e \mid \texttt{getVars}(e) \\
& & & \mid \texttt{if}\ b\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{skip} \mid \texttt{error}\ str \\
\text{Boolean Expr.} & b & ::= & e = e \mid \texttt{isVar}(e) \mid b\ \texttt{and}\ b \mid b\ \texttt{or}\ b \mid \texttt{not}\ b
\end{array}
$$

## Selector:

***list*[*pattern*]:*body***

*executes* ***body*** *for each element of* ***list*** *that matches* ***pattern***

# Keywords: *rules,* *conclusion*

***rules***[ **—>**]: ***body***

*executes* ***body*** *for each reduction rule*

11 lines (8 sloc) | 413 Bytes

```
1   Label L ::= (a) | (b)
2   Process P ::= (null) | (a P) | (b P) | (sequence P P)
3
4   (a P) --(a)--> P.
5   (b P) --(b)--> P.
6
7   (sequence P1 P2) --(a)--> (sequence P1' P2) <== P1 --(a)--> P1'.
8   (sequence P1 P2) --(a)--> P2' <== P2 --(a)--> P2' /\ P1 -/-(a)--> /\ P1 -/-(b)-->.
9
10  (sequence P1 P2) --(b)--> (sequence P1' P2) <== P1 --(b)--> P1'.
11  (sequence P1 P2) --(b)--> P2' <== P2 --(b)--> P2' /\ P1 -/-(a)--> /\ P1 -/-(b)-->.
```

**Keywords:** ***rules, conclusion***

***rules***[ **—>**]: ***conclusion***

*returns all the conclusions of the rules*

11 lines (8 sloc) | 413 Bytes

```
1   Label L ::= (a) | (b)
2   Process P ::= (null) | (a P) | (b P) | (sequence P P)
3
4   (a P) --(a)--> P.
5   (b P) --(b)--> P.
6
7   (sequence P1 P2) --(a)--> (sequence P1' P2) <== P1 --(a)--> P1'.
8   (sequence P1 P2) --(a)--> P2' <== P2 --(a)--> P2' /\ P1 -/-(a)--> /\ P1 -/-(b)-->.
9
10  (sequence P1 P2) --(b)--> (sequence P1' P2) <== P1 --(b)--> P1'.
11  (sequence P1 P2) --(b)--> P2' <== P2 --(b)--> P2' /\ P1 -/-(a)--> /\ P1 -/-(b)-->.
```

**result = *[P,  P, (sequence P1’ P2), P2’, … ]***

## Keywords: *premises* Operation: *getVars*

***rules*[ —>]: *premises[t1 —> t2]: getVars(t1)***

*retrieves all the variables of the source of reduction premises*

11 lines (8 sloc) | 413 Bytes

```
1   Label L ::= (a) | (b)
2   Process P ::= (null) | (a P) | (b P) | (sequence P P)
3
4   (a P) --(a)--> P.
5   (b P) --(b)--> P.
6
7   (sequence P1 P2) --(a)--> (sequence P1' P2) <== P1 --(a)--> P1'.
8   (sequence P1 P2) --(a)--> P2' <== P2 --(a)--> P2' /\ P1 -/-(a)--> /\ P1 -/-(b)-->.
9
10  (sequence P1 P2) --(b)--> (sequence P1' P2) <== P1 --(b)--> P1'.
11  (sequence P1 P2) --(b)--> P2' <== P2 --(b)--> P2' /\ P1 -/-(a)--> /\ P1 -/-(b)-->.
```

**result = [ *[P1], [P2, P1, P1], [P1], [P2, P1, P1]* ]**

# Lang-n-Change: Syntax

Expression $e$ ::= $x \mid str \mid t \mid [e \ldots e] \mid \texttt{head}\ e \mid \texttt{tail}\ e \mid e@e \mid e - e \mid \texttt{map}(e,e) \mid e(e)$
$\mid \texttt{rules} \mid \texttt{premises} \mid \texttt{conclusion} \mid \texttt{self}$
$\mid e[p] : e \mid \texttt{uniquefy}(e) \Rightarrow (x,x) : e \mid \texttt{getVars}(e)$
$\mid \texttt{if}\ b\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{skip} \mid \texttt{error}\ str$

Boolean Expr. $b$ ::= $e = e \mid \texttt{isVar}(e) \mid b\ \texttt{and}\ b \mid b\ \texttt{or}\ b \mid \texttt{not}\ b$

**uniquefy:**

$$\frac{\Gamma \vdash\ e_1 : T_1 \to T_2 \qquad \Gamma \vdash\ e_2 : T_1}{\Gamma \vdash\ e_1\ e_2 : T_2}\ (\text{T-APP}) \qquad T_1 \quad T_{11} \quad T_{12}$$

**uniquefy(*premisesAbove*) => (*newPremises*, *map*): body**

***body*** *is executed with bindings:*

*newPremises* **=** $\Gamma \vdash\ e_1 : T_{11} \to T_2$
$\Gamma \vdash\ e_2 : T_{12}$

*map* **=** $\{T_1 \mapsto [T_{11}, T_{12}]\}$

# New Macros for Lang-n-Change

$e$ `must match` $p_1 \mid p_2 \mid \ldots \mid p_n$ `otherwise` $e' \triangleq$ `if not`$((e - e[p_1] - e[p_2] \ldots - e[p_n]) = [])$ `then` $e'$.

`match` $e$ `with` $p \rightarrow e'$ `otherwise` $e'' \triangleq$ `if` $[e][p] = []$ `then` $e''$ `else` $e'$.

$$\begin{aligned}
\texttt{premises.LTsources} &\triangleq (\texttt{premises}[P\ \texttt{--}L\texttt{-->}\ P']:P)\ @\ (\texttt{premises}[P\ \texttt{-/-}L\texttt{-->}]:P)\\
\texttt{premises.LTtargets} &\triangleq \texttt{premises}[P\ \texttt{--}L\texttt{-->}\ P']:P'\\
\texttt{conclusion.LTsource} &\triangleq \texttt{head}\,([\texttt{conclusion}][P\ \texttt{--}L\texttt{-->}\ P']:P)\\
\texttt{conclusion.LTtarget} &\triangleq \texttt{head}\,([\texttt{conclusion}][P\ \texttt{--}L\texttt{-->}\ P']:P')
\end{aligned}$$

$\texttt{distinctVars}(e)$ `otherwise` $e' \triangleq$

$\texttt{uniquefy}([(pname\ e)]) \Rightarrow (new, m) :$ `if not`$(m = \texttt{map}([],[]))$ `then` $e'$

# A GSOS Validator in Lang-n-Change

$$\frac{\{x_i \longrightarrow^{l_{ij}} y_{ij} \mid i \in I, 1 \le j \le m_i\} \cup \{x_j \not\longrightarrow^{l'_{jk}} \mid j \in J, 1 \le k \le n_j\}}{(op\ x_1 \ldots x_h) \longrightarrow^{l} t}$$

**Part 1:**

***All premises are positive or negative transition formulae, and they use constant labels***

```
rules[-->]: premises must match P --(op [])--> P' | P -/-(op [])-->
            otherwise error msg
```

# A GSOS Validator in Lang-n-Change

$$\frac{\{x_i \longrightarrow^{l_{ij}} y_{ij} \mid i \in I, 1 \leq j \leq m_i\} \cup \{x_j \not\longrightarrow^{l'_{jk}} \mid j \in J, 1 \leq k \leq n_j\}}{(op\, x_1 \ldots x_h) \longrightarrow^{l} t}$$

## Part 2:

***All conclusions are positive formulae that use a constant label, whose source is an operator applied to metavariables***

```
rules[-->]: match conclusion with (op1 Ps )--(op2 [])--> P' ->
                    Ps[P]: if not(isVar(P)) then error msg1
                  otherwise error msg2
```

# A GSOS Validator in Lang-n-Change

$$\frac{\{x_i \longrightarrow^{l_{ij}} y_{ij} \mid i \in I, 1 \leq j \leq m_i\} \cup \{x_j \not\longrightarrow^{l'_{jk}} \mid j \in J, 1 \leq k \leq n_j\}}{(op\, x_1 \ldots x_h) \longrightarrow^{l} t}$$

## Part 3:

***Sources of premises must come from xs of the conclusion, and ys must be metavariables***

```
rules[-->]:
  if not(premises.LTsources sublistOf getVars(conclusion.LTsource))
  then error msg_1
  else premises.LTtargets[P]: if not(isVar(P)) then error msg_2
```

# A GSOS Validator in Lang-n-Change

$$\frac{\{x_i \longrightarrow^{l_{ij}} y_{ij} \mid i \in I, 1 \leq j \leq m_i\} \cup \{x_j \not\longrightarrow^{l'_{jk}} \mid j \in J, 1 \leq k \leq n_j\}}{(op\, x_1 \ldots x_h) \longrightarrow^{l} t}$$

**Part 4:**

***xs in the source of the conclusion and ys in the premises must all be distinct***

```
rules[-->]:
  distinctVars (getVars(conclusion.LTsource) @ premises.LTtargets)
  otherwise error msg
```

# A GSOS Validator in Lang-n-Change

$$\frac{\{x_i \longrightarrow^{l_{ij}} y_{ij} \mid i \in I, 1 \leq j \leq m_i\} \cup \{x_j \not\longrightarrow^{l'_{jk}} \mid j \in J, 1 \leq k \leq n_j\}}{(op\, x_1 \ldots x_h) \longrightarrow^{l} t}$$

**Part 5:**

***Metavariables in the target of the conclusion come from xs and ys***

```
rules[-->]:
 if not(getVars(conclusion.LTtarget)
                sublistOf
        (getVars(conclusion.LTsource) @ premises.LTtargets))
 then error msg
```

# Implementation: Input Example (.lan)

11 lines (8 sloc) | 413 Bytes

```
1   Label L ::= (a) | (b)
2   Process P ::= (null) | (a P) | (b P) | (sequence P P)
3
4   (a P) --(a)--> P.
5   (b P) --(b)--> P.
6
7   (sequence P1 P2) --(a)--> (sequence P1' P2) <== P1 --(a)--> P1'.
8   (sequence P1 P2) --(a)--> P2' <== P2 --(a)--> P2' /\ P1 -/-(a)--> /\ P1 -/-(b)-->.
9
10  (sequence P1 P2) --(b)--> (sequence P1' P2) <== P1 --(b)--> P1'.
11  (sequence P1 P2) --(b)--> P2' <== P2 --(b)--> P2' /\ P1 -/-(a)--> /\ P1 -/-(b)-->.
```

**GSOS-Validator:** ***This language is a GSOS language***

# Implementation: Input Example (.lan)

```
1  Label L ::= (a)
2  Process P ::= (null) | (a P) | (par P P) | (repl P)
3
4  (a P) --(a)--> P.
5  (par P1 P2) --(a)--> (par P1' P2) <== P1 --(a)--> P1'.
6  (par P1 P2) --(a)--> (par P1 P2') <== P2 --(a)--> P2'.
7
8  (repl P1) --(a)--> P1' <== (par (repl P1)) --(a)--> P1'.
```

**GSOS-Validator:** 

*Rule #4: Sources of premises must be variables that are arguments of the operator in the source of the conclusion*

# Implementation: Input Example (.lan)

```
1  Label L ::= (tau)
2  Process P ::= (action) | (ok) | (error) | (from P)
3
4  (action) --(tau)--> (ok).
5
6  (from P1) --(tau)--> P2 <== P2 --(tau)--> P1.
```

**ok** bisim **error**

**(from ok)** not bisim **(from error)**

**GSOS-Validator:** 

*Rule #2: Sources of premises must be variables that are arguments of the operator in the source of the conclusion*

# Evaluation

process_algebra.lan (called PA below): **(only terminated process and prefix operator)**

process_algebra_ACPprojection.lan: PA + projection operator of ACP

process_algebra_CCSchoice.lan: PA + (external) choice operator of CCS

process_algebra_CCSparallel.lan: PA + interleaving parallel operator of CCS with no communication

process_algebra_CCScommunication.lan: PA + CCS parallel operator.

process_algebra_CSPsynchParallel.lan: PA + the synchronous parallel operator of CSP

process_algebra_Internalchoice.lan: PA + CSP (internal) choice operator.

process_algebra_LOTOSdisrupt.lan: PA + disrupt operator from LOTOS (ISO/IEC standard 1989.

process_algebra_leftMerge.lan: PA + left merge operator of ACP

process_algebra_restriction.lan: : PA + CCS restriction operator.

process_algebra_priority.lan: : PA + the priority operator.

process_algebra_sequence.lan: : PA + the sequential operator ';'.

process_algebra_interrupt.lan: : PA + the interrupt operator.

**… and more. Total: 18 process algebra operators**

- **Concise: 6 lines of code**
- **Declaratively expresses the GSOS restrictions**
- **Testable all at once**

# In conclusion

*Evidence that* **Lang-n-Check** *can be used to write express formats*

- *concisely*
- *declaratively*
- *can be applied to a test suite of process algebras at once*

# Future Work

- *Develop more rule formats in* **Lang-n-Change**
- *Integrate* **Lang-n-Change** *within programming languages*
- **Lang-n-Change** *vs ``languages-as-databases*[*]*''?*

**Thank you!**

**[*] Matteo Cimini.** ***A Query Language for Language Analysis*****. SEFM 2022**