Testing Languages with a Languages-as-Databases Approach

Matteo Cimini
University of Massachusetts Lowell

July, 19th 2023
TAP 2023, Leicester, UK

Language Validation

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

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

$\ldots$

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

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

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

$\ldots$

What Language Definitions?

Type	T	$::=$	$\texttt{Bool} \mid T \to T \mid \texttt{List}\ T$
Expression	e	$::=$	$\texttt{true} \mid \texttt{false} \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e$
			$\mid x \mid \lambda x : T.e \mid e\ e$
			$\mid \texttt{nil} \mid \texttt{cons}\ e\ e \mid \texttt{head}\ e$
			$\mid \texttt{let}\ x = e\ \texttt{in}\ e$
			$\mid \texttt{error}$
Value	v	$::=$	$\texttt{true} \mid \texttt{false} \mid \lambda x : T.e \mid \texttt{nil} \mid \texttt{cons}\ v\ v$
Error	er	$::=$	$\texttt{error}$
EvalCtx	E	$::=$	$\square \mid \texttt{if}\ E\ \texttt{then}\ e\ \texttt{else}\ e \mid E\ e \mid v\ E$
			$\mid \texttt{cons}\ E\ e \mid \texttt{cons}\ v\ E \mid \texttt{head}\ E$
			$\mid \texttt{let}\ x = E\ \texttt{in}\ e$

Reduction Semantics

$$\boxed{e \longrightarrow e}$$

$$\texttt{if true then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_1$$
$$\texttt{if false then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_2$$
$$(\lambda x : T.e)\ v \longrightarrow e[v/x]$$
$$\texttt{head nil} \longrightarrow \texttt{error}$$
$$\texttt{head}\ (\texttt{cons}\ v_1\ v_2) \longrightarrow v_1$$
$$\texttt{let}\ x = v\ \texttt{in}\ e \longrightarrow e[v/x]$$

$$\frac{e \longrightarrow e'}{E[e] \longrightarrow E[e']} \qquad E[er] \longrightarrow er$$

Type System

$$\boxed{\Gamma \vdash e : T}$$

$$\Gamma, x : T \vdash x : T \qquad \Gamma \vdash \texttt{true} : \texttt{Bool} \qquad \Gamma \vdash \texttt{false} : \texttt{Bool}$$

$$\Gamma \vdash \texttt{error} : T \qquad \frac{\Gamma \vdash e_1 : \texttt{Bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : T}$$

$$\frac{\Gamma, x : T_1 \vdash e : T_2}{\Gamma \vdash \lambda x : T_1.e : T_1 \to T_2} \qquad \frac{\Gamma \vdash e_1 : T_1 \to T_2 \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\ e_2 : T_2}$$

$$\Gamma \vdash \texttt{nil} : \texttt{List}\ T \qquad \frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : \texttt{List}\ T}{\Gamma \vdash \texttt{cons}\ e_1\ e_2 : \texttt{List}\ T}$$

$$\frac{\Gamma \vdash e : \texttt{List}\ T}{\Gamma \vdash \texttt{head}\ e : T} \qquad \frac{\Gamma \vdash e_1 : T_1 \quad \Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : T_2}$$

Language-parametrized Testing

- *Roberson et al.'s model checking approach* [*]
- *K framework, MPS, Spoofax, and others*
- *PLT Redex* [**]

Method: Testing languages by their *products*

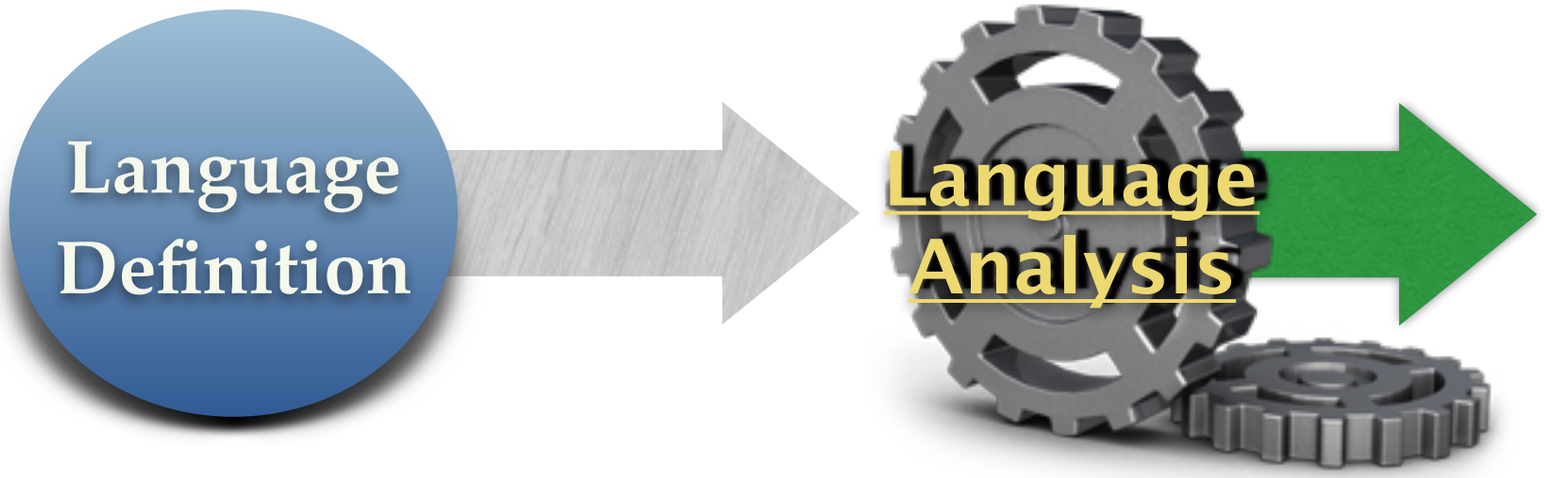
Would be beneficial: Testing languages by their *qualities*

[*] Roberson, M., Harries, M., Darga, P.T., Boyapati, C.
Efficient software model checking of soundness of type systems. OOPSLA'08.

[**] Klein, C., Clements, J., Dimoulas, C., Eastlund, C., Felleisen, M., Flatt, M., Mc- Carthy, J.A., Rafkind, J., Tobin-Hochstadt, S., Findler, C.
Run your research: On the effectiveness of lightweight mechanization. POPL'12.

Language Testing

small questions { **what are the canonical forms?**
what are the elimination forms?
evaluation underneath binders?
...



canonical forms

$true$	$bool$
$false$	$bool$
abs	$arrow$
nil	$list$
$cons$	$list$

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

$\ldots$

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

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

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

$\ldots$

helpful for debugging

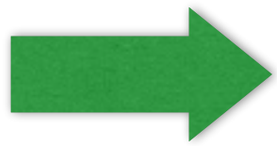
Prior Work [SEFM 2022]

"*Interrogating languages should be akin to interrogating databases*"

- **languages-as-databases** *representation of languages*
- **Lang-SQL**, *tailored for interrogating operational semantics*

canonical forms

```
SELECT GET-OPNAME(term) AS val,
       GET-OPNAME(LAST(args, 0)) AS type
FROM Value, rule
WHERE predname = ⊢ AND role = CONCL
AND GET-OPNAME(term) = GET-OPNAME(LAST(args, 1))
```



val	type
true	*bool*
false	*bool*
abs	*arrow*
nil	*list*
cons	*list*

[SEFM 2022] Matteo Cimini. ***A Query Language for Language Analysis*****. SEFM 20202.**

Prior Work [SEFM 2022]

"*Int*

SELE

FRON
WHER
AND

Our question:

Can we use languages-as-databases as a lightweight approach to language testing?

In this paper:

- **Extend Lang-SQL with testing constructs and macros**
- **Develop Lang-SQL language tests on various PL aspects**
- **Validator for GSOS rule format ("*tests that prove*")**

[SEFM 2022] Matteo Cimini. *A Query Language for Language Analysis*. **SEFM 20202.**

Languages-as-Databases

Type	T	$::= \texttt{Int} \mid \texttt{Float} \mid T \to T \mid \texttt{Ref}\ T$
Expression	e	$::= n \mid f \mid x \mid \lambda x : T.e \mid e\ e \mid l \mid \texttt{ref}\ e \mid\ !e \mid e := e \mid \texttt{error}\ e$
Value	v	$::= n \mid f \mid \lambda x : T.e \mid l$
EvalCtx	E	$::= E\ v \mid v\ E \mid \texttt{ref}\ E \mid !E \mid \lambda x : T.E \mid E := e \mid v := E \mid \texttt{error}\ E$
Error	er	$::= \texttt{error}\ v$

grammar-info

category	meta-var	obj-var
Type	T	–
Expression	e	x
Value	v	–
EvalCtx	E	–
Error	er	–

grammar

category	term
Type	int
Type	$arrow\ T\ T$
...	...
Expression	$abs\ T\ (x)e$
Expression	$app\ \ e\ e$
...	...
EvalCtx	$app\ E\ e$
...	...

Languages-as-Databases

Type	T	$::= \texttt{Int} \mid \texttt{Float} \mid T \to T \mid \texttt{Ref}\ T$
Expression	e	$::= n \mid f \mid x \mid \lambda x : T.e \mid e\ e \mid l \mid \texttt{ref}\ e \mid\ !e \mid e := e \mid \texttt{error}\ e$
Value	v	$::= n \mid f \mid \lambda x : T.e \mid l$
EvalCtx	E	$::= E\ v \mid v\ E \mid \texttt{ref}\ E \mid !E \mid \lambda x : T.E \mid E := e \mid v := E \mid \texttt{error}\ E$
Error	er	$::= \texttt{error}\ v$

**We did not intend to evaluate underneath a \lambda.
This is an oversight.**

grammar-info

category	meta-var	obj-var
Type	T	—
Expression	e	x
Value	v	—
EvalCtx	E	—
Error	er	—

category	term
Type	int
Type	$arrow\ T\ T$
...	...
Expression	$abs\ T\ (x)e$
Expression	$app\ \ e\ e$
...	...
EvalCtx	$app\ E\ e$
...	...

Languages-as-Databases

$$\frac{\Sigma \mid \Gamma \vdash\ e : T}{\Sigma \mid \Gamma \vdash\ \texttt{ref}\ e : \texttt{Ref}\ T} \qquad \frac{\Sigma \mid \Gamma \vdash\ e_1 : T_1 \to T_2 \quad \Sigma \mid \Gamma \vdash\ e_2 : T_3 \quad T_3 <: T_1}{\Sigma \mid \Gamma \vdash\ e_1\ e_2 : T_2} \qquad \frac{\Sigma \mid \Gamma \vdash\ e : T}{\Sigma \mid \Gamma \vdash\ \texttt{error}\ e : T}$$

$$\frac{T_1' <: T_1 \quad T_2 <: T_2'}{T_1 \to T_2 <: T_1' \to T_2'} \qquad \frac{T <: T'}{\texttt{Ref}\ T <: \texttt{Ref}\ T'}$$

`rule`

rulename	predname	args	role
(T-APP)	$\vdash$	$[\Sigma; \Gamma; e_1; T_1 \to T_2]$	PREM
(T-APP)	$\vdash$	$[\Sigma; \Gamma; e_2; T_3]$	PREM
(T-APP)	$<:$	$[T_3; T_1]$	PREM
(T-APP)	$\vdash$	$[\Sigma; \Gamma; app\ e_1\ e_2; T_2]$	CONCL
(S-ARROW)	$<:$	$[T_1'; T_1]$	PREM
(S-ARROW)	$<:$	$[T_2; T_2']$	PREM
(S-ARROW)	$<:$	$[arrow\ T_1\ T_2; arrow\ T_1'\ T_2']$	CONCL
...	...	...	...

Languages-as-Databases

$$\frac{\Sigma \mid \Gamma \vdash\ e : T}{\Sigma \mid \Gamma \vdash\ \texttt{ref}\ e : \texttt{Ref}\ T} \qquad \frac{\Sigma \mid \Gamma \vdash\ e_1 : T_1 \to T_2 \quad \Sigma \mid \Gamma \vdash\ e_2 : T_3 \quad T_3 <: T_1}{\Sigma \mid \Gamma \vdash\ e_1\ e_2 : T_2} \qquad \frac{\Sigma \mid \Gamma \vdash\ e : T}{\Sigma \mid \Gamma \vdash\ \texttt{error}\ e : T}$$

$$\frac{T_1' <: T_1 \quad T_2 <: T_2'}{T_1 \to T_2 <: T_1' \to T_2'} \qquad \frac{T <: T'}{\texttt{Ref}\ T <: \texttt{Ref}\ T'}$$

references must be invariant

rule		args	role
(T-APP)	$\vdash$	$[\Sigma; \Gamma; e_1; T_1 \to T_2]$	PREM
(T-APP)	$\vdash$	$[\Sigma; \Gamma; e_2; T_3]$	PREM
(T-APP)	$<:$	$[T_3; T_1]$	PREM
(T-APP)	$\vdash$	$[\Sigma; \Gamma; app\ e_1\ e_2; T_2]$	CONCL
(S-ARROW)	$<:$	$[T_1'; T_1]$	PREM
(S-ARROW)	$<:$	$[T_2; T_2']$	PREM
(S-ARROW)	$<:$	$[arrow\ T_1\ T_2; arrow\ T_1'\ T_2']$	CONCL
...	...	...	...

Languages-as-Databases

errors must be typed at any type
!(ref (error 0)) —> !(error 0)
well-typed ill-typed

typically: |- error : *T'* for a fresh *T'*

$$\frac{\Sigma \mid \Gamma \vdash e : T}{\Sigma \mid \Gamma \vdash \texttt{error}\ e : T}$$

$$\frac{T <: T'}{\texttt{ef}\ T <: \texttt{Ref}\ T'}$$

rule

rulename	predname	args	role
(T-APP)	$\vdash$	$[\Sigma; \Gamma; e_1; T_1 \to T_2]$	PREM
(T-APP)	$\vdash$	$[\Sigma; \Gamma; e_2; T_3]$	PREM
(T-APP)	$<:$	$[T_3; T_1]$	PREM
(T-APP)	$\vdash$	$[\Sigma; \Gamma; app\ e_1\ e_2; T_2]$	CONCL
(S-ARROW)	$<:$	$[T_1'; T_1]$	PREM
(S-ARROW)	$<:$	$[T_2; T_2']$	PREM
(S-ARROW)	$<:$	$[arrow\ T_1\ T_2; arrow\ T_1'\ T_2']$	CONCL
...	...	...	...

A Query Language for Querying Languages

Lang-SQL

Table	τ	$::=$	$\texttt{grammar} \mid \texttt{grammar-info} \mid \texttt{rule} \mid name$
Expression	e	$::=$	$t \mid attr \mid name \mid \texttt{CONCL} \mid \texttt{PREM} \mid [e; e \cdots ; e] \mid \texttt{NTH}(e, e)$
			$\mid \texttt{GET-OPNAME}(e) \mid \texttt{GET-ARGS}(e) \mid \texttt{GET-VARS}(e)$
			$\mid \texttt{GET-BOUND-TERM}(e) \mid \texttt{GET-BOUND-VAR}(e)$
Formula	f	$::=$	$e = e \mid e\ \texttt{IS}\ e\ \texttt{VAR} \mid e\ \texttt{IS BOUND} \mid e\ \texttt{IS CONSTANT}$
			$\mid f\ \texttt{AND}\ f \mid f\ \texttt{OR}\ f \mid \texttt{NOT}\ f$
Select Item	e^*	$::=$	$\star \mid e\ \texttt{AS}\ (\texttt{ROWS}^\star)\ attr$
Query	q	$::=$	$\texttt{SELECT}\ (\texttt{DISTINCT})\ e^*\ \ \texttt{FROM}\ \overline{q}\ \texttt{WHERE}\ f \mid \tau \mid q\ \texttt{UNION}\ q$

A Query Language for Querying Languages

Lang-SQL

Table τ ::= `grammar` | `grammar-info` | `rule` | $name$

Expression e ::= t | $attr$ | $name$ | `CONCL` | `PREM` | $[e; e \cdots ; e]$ | `NTH`(e, e)
| `GET-OPNAME`(e) | `GET-ARGS`(e) | `GET-VARS`(e)
| `GET-BOUND-TERM`(e) | `GET-BOUND-VAR`(e)

Formula f ::= $e = e$ | e `IS` e `VAR` | e `IS BOUND` | e `IS CONSTANT`
| f `AND` f | f `OR` f | `NOT` f

Select Item e^* ::= $\star$ | e `AS` (`ROWS`*) $attr$

Query q ::= `SELECT` (`DISTINCT`) e^* `FROM` $\overline{q}$ `WHERE` f | τ | q `UNION` q

GET-OPNAME((if e1 e2 e3)) = if
GET-ARGS((if e1 e2 e3)) = [e1 ; e2 ; e3]
GET-VARS(fst (pair v1 v2)) = [v1 ; v2]

Implementation

output table

$\texttt{column}_1$	$\texttt{column}_2$	$\cdots$
a_{11}	a_{12}	$\cdots$
a_{21}	a_{22}	$\cdots$
a_{31}	a_{32}	$\cdots$
$\cdots$	$\cdots$	$\cdots$

https://github.com/mcimini/lang-sql

Implementation: Input Example (.lan)

Code | Blame | Executable File · 23 lines (13 loc) · 603 Bytes

```
1   Expression E ::= x | (abs T (x)E) | (tt ) | (ff ) | (app E E) | (if E E E)
2   Type T ::= (arrow T T) | (bool )
3   Value V ::= (abs T1 R2) | (tt ) | (ff )
4   Error ::=
5   Context C ::= [] | (app C e) | (app v C) | (if C e e)
6
7   Gamma |- (abs T1 R) : (arrow T1 T2) <== Gamma, x : T1 |- R : T2.
8
9   Gamma |- (tt ) : (bool ).
10
11  Gamma |- (ff ) : (bool ).
12
13  Gamma |- (app E1 E2) : T2 <== Gamma |- E1 : (arrow T1 T2) /\ Gamma |- E2 : T1.
14
15  (app (abs T R) V) --> R[V/x] <== value V.
16
17  Gamma |- (if E1 E2 E3) : T <== Gamma |- E1 : (bool ) /\ Gamma |- E2 : T /\ Gamma |- E3 : T.
18
19  (if (tt ) E1 E2) --> E1.
20
21  (if (ff ) E1 E2) --> E2.
```

Implementation

output table

$\texttt{column}_1$	$\texttt{column}_2$	$\cdots$
a_{11}	a_{12}	$\cdots$
a_{21}	a_{22}	$\cdots$
a_{31}	a_{32}	$\cdots$
$\cdots$	$\cdots$	$\cdots$

fail tables

$\texttt{column}_1$	$\texttt{column}_2$	$\cdots$
a_{11}	a_{12}	$\cdots$
a_{21}	a_{22}	$\cdots$
a_{31}	a_{32}	$\cdots$
$\cdots$	$\cdots$	$\cdots$

https://github.com/mcimini/lang-sql

A Query Language for Querying Languages

Lang-SQL

Table τ ::= `grammar` | `grammar-info` | `rule` | $name$

Expression e ::= t | $attr$ | $name$ | `CONCL` | `PREM` | $[e; e \cdots ; e]$ | `NTH`(e, e)
| `GET-OPNAME`(e) | `GET-ARGS`(e) | `GET-VARS`(e)
| `GET-BOUND-TERM`(e) | `GET-BOUND-VAR`(e)

Formula f ::= $e = e$ | e `IS` e `VAR` | e `IS BOUND` | e `IS CONSTANT`
| f `AND` f | f `OR` f | `NOT` f

Select Item e^* ::= $\star$ | $\overline{e \texttt{ AS } (\texttt{ROWS}^\star)\ attr}$

Query q ::= `SELECT` (`DISTINCT`) e^* `FROM` $\overline{q}$ `WHERE` f | τ | q `UNION` q

Addition

Test Preamble tp ::= `TEST` ϕ

Test ϕ ::= $q = q$ | q `CONTAINS` q | q `DISJOINT` q | q `IS EMPTY`
| q `DISTINCT ROWS` | ϕ `AND` ϕ | ϕ `OR` ϕ | `NOT` ϕ

Macros ::= ***we will see as we encounter them***

Test: No Evaluation Underneath Binders

$whereEvalUnderBinders \triangleq$

```
SELECT * FROM
                (SELECT GET-OPNAME(term) AS opname,
                 GET-ARGS(term) AS ROWS* arg
                 FROM EvalCtx)
WHERE arg IS BOUND AND GET-BOUND-TERM(arg) = E

TEST whereEvalUnderBinders IS EMPTY
```

EvalCtx

term
$abs\ [T; (x)E]$
$app\ [E; e]$
$app\ [v; E]$
$\ldots$

Test: No Evaluation Underneath Binders

$whereEvalUnderBinders \triangleq$

```
SELECT * FROM
                (SELECT GET-OPNAME(term) AS opname,
                 GET-ARGS(term) AS ROWS* arg
                 FROM EvalCtx)
WHERE arg IS BOUND AND GET-BOUND-TERM(arg) = E

TEST whereEvalUnderBinders IS EMPTY
```

EvalCtx

term
$abs\ [T; (x)E]$
$app\ [E; e]$
$app\ [v; E]$
$\ldots$

Nested `SELECT`

opname	*arg*
abs	T
abs	$(x)E$
app	E
app	e
$\ldots$	$\ldots$

Test: No Evaluation Underneath Binders

$whereEvalUnderBinders \triangleq$

```
SELECT * FROM
                (SELECT GET-OPNAME(term) AS opname,
                 GET-ARGS(term) AS ROWS* arg
                 FROM EvalCtx)
WHERE arg IS BOUND AND GET-BOUND-TERM(arg) = E
```

```
TEST whereEvalUnderBinders IS EMPTY
```

EvalCtx

term
$abs\ [T; (x)E]$
$app\ [E; e]$
$app\ [v; E]$
...

Nested `SELECT`

opname	*arg*
abs	T
abs	$(x)E$
app	E
app	e
...	...

whereEvalUnderBinders

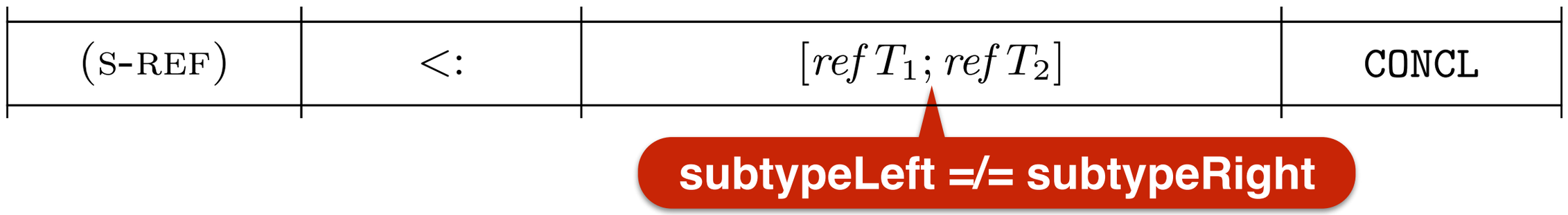
opname	*arg*
abs	$(x)E$

Test: Reference must be invariant

$invariantTypes \triangleq$

```
SELECT GET-OPNAME(subtypeLeft) AS type FROM rule
WHERE predname = <: AND role = CONCL
      AND subtypeLeft = subtypeRight
```

```
TEST invariantTypes CONTAINS ROW type = ref
```



invariantTypes ***is the empty table***

subtypeLeft subtypeRight

Ref T <: Ref T

Test: Reference must be invariant

$invariantTypes \triangleq$

```
SELECT GET-OPNAME(subtypeLeft) AS type FROM rule
WHERE predname = <: AND role = CONCL
      AND subtypeLeft = subtypeRight
```

```
TEST invariantTypes CONTAINS ROW type = ref
```

suppose we had

(S-REF)	<:	$[ref\ T; ref\ T]$	CONCL

subtypeLeft = subtypeRight

$invariantTypes$

$type$
ref

Evaluation [*]

- Test that reduction does not occur underneath binders
- Test that references are invariant
- Test that errors are typed at any type
- Test that types do not bind expressions (as in dependent types)

- Tested on the repo of Lang-SQL
  - debugged the running language example with references
  - several correct languages pass the tests
  - strong calculi: fail the first test
  - dependently typed language fails the last test

- Tests are concise and declarative

[*] https://github.com/mcimini/lang-sql

Example of Process Algebra

Label $l ::= a \mid b \mid c$ Process $P ::= \mathbf{0} \mid l.P \mid \theta(P)$

$$a.P \longrightarrow^{a} P \qquad b.P \longrightarrow^{b} P \qquad c.P \longrightarrow^{c} P$$

$$\frac{P \longrightarrow^{a} P'}{\theta(P) \longrightarrow^{a} \theta(P')} \qquad \frac{P \longrightarrow^{b} P' \quad P \not\longrightarrow^{a}}{\theta(P) \longrightarrow^{b} \theta(P')} \qquad \frac{P \longrightarrow^{c} P' \quad P \not\longrightarrow^{a}}{\theta(P) \longrightarrow^{c} \theta(P')}$$

GSOS Rule Format: *each rule must be such that*

premises are of the form $x \longrightarrow^{l} y$ *or* $x \not\longrightarrow^{l}$

source vars in premises come from the source of the conclusion

conclusion source vars and premises target vars are distinct

… and others

"*Tests that prove*"[*]: *If all the rules adhere to the GSOS restrictions then bisimilarity is a congruence for the language*

[*] Bloom, B., Istrail, S., Meyer, A.R. *Bisimulation can't be traced.* **J. of the ACM 1995**

A GSOS Validator in Lang-SQL

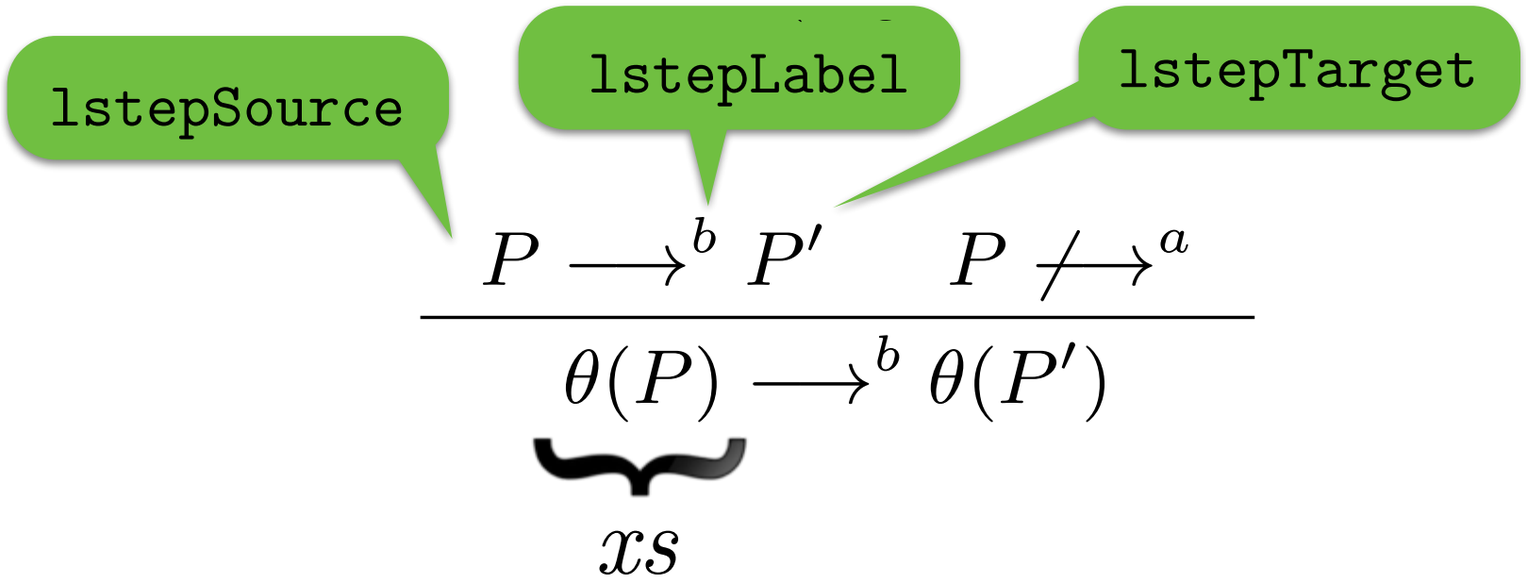
lstepSource lstepLabel lstepTarget

$$\frac{P \longrightarrow^{b} P' \qquad P \not\longrightarrow^{a}}{\theta(P) \longrightarrow^{b} \theta(P')}$$

premises are of the form $x \longrightarrow^{l} y$ **or** $x \not\longrightarrow^{l}$

```
validPositivePrems ≜
SELECT * FROM rule WHERE role = PREM AND predname = ⟶
         AND lstepSource IS Process VAR
         AND lstepLabel IS CONSTANT
         AND lstepTarget IS Process VAR
validNegativePrems ≜
SELECT * FROM rule WHERE role = PREM AND predname = ⟶/
         AND lstepSource IS Process VAR
         AND lstepLabel IS CONSTANT
TEST (validPositivePrems UNION validNegativePrems) = premises
```

A GSOS Validator in Lang-SQL



source vars $xsInPremises$ ***in premises come from the conclusion source***

$xs \triangleq$ SELECT rulename, GET-VARS(lstepSource) AS ROWS* var
FROM conclusions

$xsInPremises \triangleq$ SELECT rulename, lstepSource AS var FROM premises

TEST xs CONTAINS $xsInPremises$

A GSOS Validator in Lang-SQL

$$\frac{P \overbrace{\longrightarrow^{b} P'}^{ys} \qquad P \not\longrightarrow^{a}}{\underbrace{\theta(P)}_{xs} \longrightarrow^{b} \theta(P')}$$

conclusion source vars xs
and premises target vars ys
are all distinct

$xs \triangleq$ `SELECT rulename, GET-VARS(lstepSource) AS ROWS`* var
`FROM conclusions`

$ys \triangleq$ `SELECT rulename, lstepTarget AS` var `FROM premises`

`TEST (`xs `UNION` ys`) DISTINCT ROWS`

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: 14 lines of code**
- **Declaratively expresses the GSOS restrictions**

[*] https://github.com/mcimini/lang-sql

Message

Some evidence that languages-as-databases can be a lightweight approach to language testing

Future Work

- **Add recursion. (There are queries we cannot express)**
- **User-friendly error messages rather than tables**
- **More tests**
- **Integrate Lang-SQL within programming languages**

Thank you!

Circular Evaluation Contexts

$$\text{EvalCtx} \qquad E ::= E\ v \mid v\ E$$

Detecting circularity requires topological sort in the general case

User-friendly Error Messages

Not:

$whereEvalUnderBinders$

$opname$	arg
abs	$(x)E$

But

EvalCtx $E ::= \square \mid E\ e \mid v\ E \mid \texttt{ref}\ E \mid !E \mid \lambda x : T.E \mid E := e \mid v := E \mid \texttt{error}\ E$