Generating Specialized Interpreters for Modular Structural Operational Semantics

Casper Bach Poulsen and Peter D. Mosses

Department of Computer Science
Swansea University
Swansea, UK

cscbp@swansea.ac.uk, p.d.mosses@swansea.ac.uk

18 September 2013,
LOPSTR, Madrid, Spain

Motivation

PLanCompS project (Programming Language Components and Specifications):

- reusable fundamental constructs (funcons)
- dynamic semantics specified in small-step Modular SOS

The Prolog MSOS Tool: MSOS rules $\Longrightarrow$ Prolog clauses

Goal

minimize overhead in generated interpreters
by refocusing and specialization

Overview

SOS vs. MSOS

MSOS in Prolog

Refocusing in MSOS

Striding

Structural Operational Semantics (SOS)

Specification of bound: $\boxed{\rho \vdash e \to e'}$

$$\frac{\rho(id) = v}{\rho \vdash \mathsf{bound}(id) \to v}$$

Specification of deref: $\boxed{\langle e, \sigma \rangle \to \langle e', \sigma' \rangle}$

$$\frac{\sigma(ref) = v}{\langle \mathsf{deref}(ref), \sigma \rangle \to \langle v, \sigma \rangle}$$

The modularity problem: Combining constructs requires reformulation: $\boxed{\rho \vdash \langle e, \sigma \rangle \to \langle e', \sigma' \rangle}$

$$\frac{\rho(id) = v}{\rho \vdash \langle \mathsf{bound}(id), \sigma \rangle \to \langle v, \sigma \rangle} \qquad \frac{\sigma(ref) = v}{\rho \vdash \langle \mathsf{deref}(ref), \sigma \rangle \to \langle v, \sigma \rangle}$$

Modular SOS

$\boxed{e \xrightarrow{L} e'}$

Specification of bound:

$$\frac{\rho(id) = v}{\mathsf{bound}(id) \xrightarrow{\{\mathbf{env}=\rho,-\}} v}\ [\textsc{bound}]$$

Specification of deref:

$$\frac{\sigma(ref) = v}{\mathsf{deref}(ref) \xrightarrow{\{\mathbf{sto}=\sigma,\mathbf{sto}'=\sigma,-\}} v}\ [\textsc{deref}]$$

Combining constructs requires no reformulation

Writable label components

C-style assignment:

$$\frac{e \xrightarrow{\{\ldots\}} e'}{\mathsf{assign}(ref, e) \xrightarrow{\{\ldots\}} \mathsf{assign}(ref, e')}\ [\textsc{asn1}]$$

$$\frac{\sigma' = \sigma[ref \mapsto v]}{\mathsf{assign}(ref, v) \xrightarrow{\{\mathbf{sto}=\sigma, \mathbf{sto}'=\sigma', \text{—}\}} v}\ [\textsc{asn2}]$$

SOS vs. MSOS

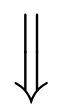
MSOS in Prolog

Refocusing in MSOS

Striding

From MSOS rule to Prolog clause

$$\frac{\sigma' = \sigma[ref \mapsto v]}{\mathsf{assign}(ref, v) \xrightarrow{\{\mathbf{sto}=\sigma,\mathbf{sto}'=\sigma',-\}} v}\ [\textsc{Asn2}]$$



```
%% [Asn2]
step(assign(Ref,v(V)), L, v(V)) :-
    label_instance(L, [sto=Sigma,sto+=Sigma_|unobs(LDots)]),
    map_update(Sigma, Ref, v(V), Sigma_).
```

MSOS evaluation in Prolog

The transitive closure $\rightarrow^*$:

$$s \xrightarrow{L_1} s' \xrightarrow{L_2} s'' \rightarrow \cdots$$

such that each consecutive label pair composes

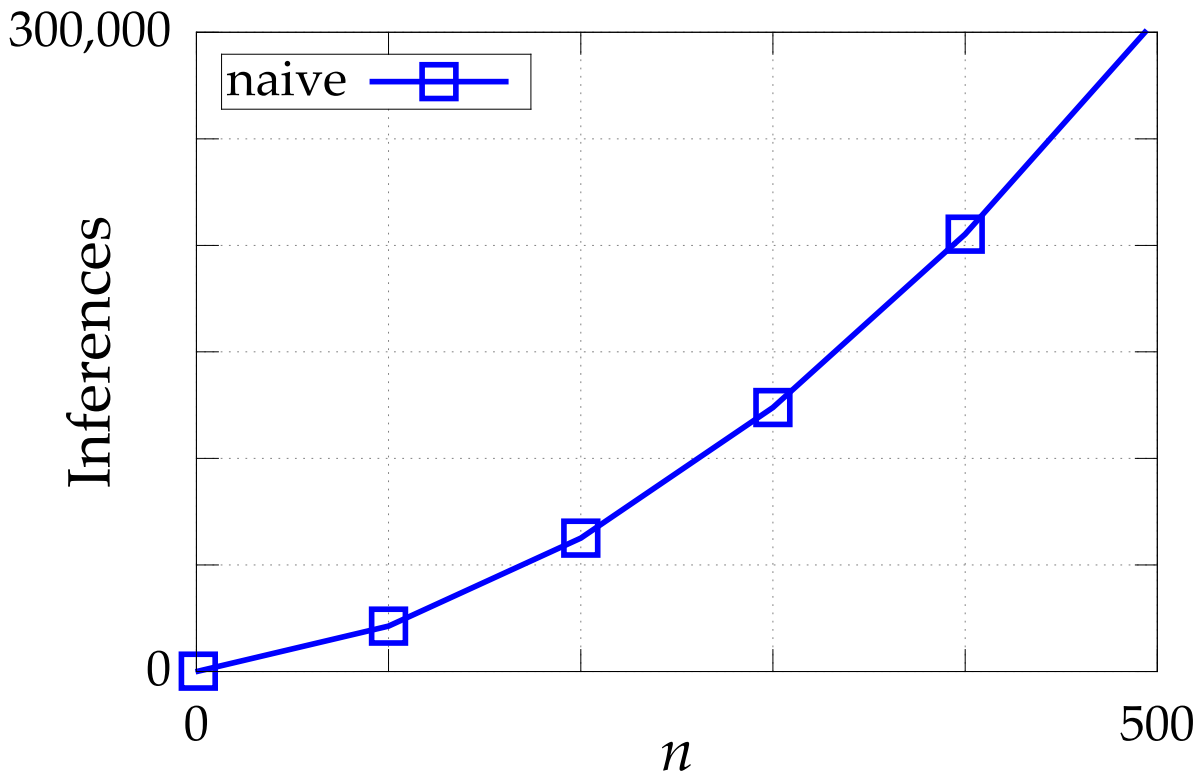
```
%% [Trans]
steps(E1, L, E3) :-
    pre_comp(L, L1),
    step(E1, L1, E2),
    mid_comp(L1, L2),
    steps(E2, L2, E3),
    post_comp(L1, L2, L).

%% [Refl-v]
steps(v(V), L, v(V)) :-
    label_instance(L, unobs(L)).
```

Small-step inefficiency

$$\underbrace{\mathsf{assign}(\mathsf{x}_1, \mathsf{assign}(\mathsf{x}_2 \cdots \mathsf{assign}}_{n}(\mathsf{x}_n, 42) \cdots))$$

Generated Prolog interpreter uses n traversals of terms of depth $O(n)$, i.e., $O(n^2)$ inferences:



SOS vs. MSOS

MSOS in Prolog

Refocusing in MSOS

Striding

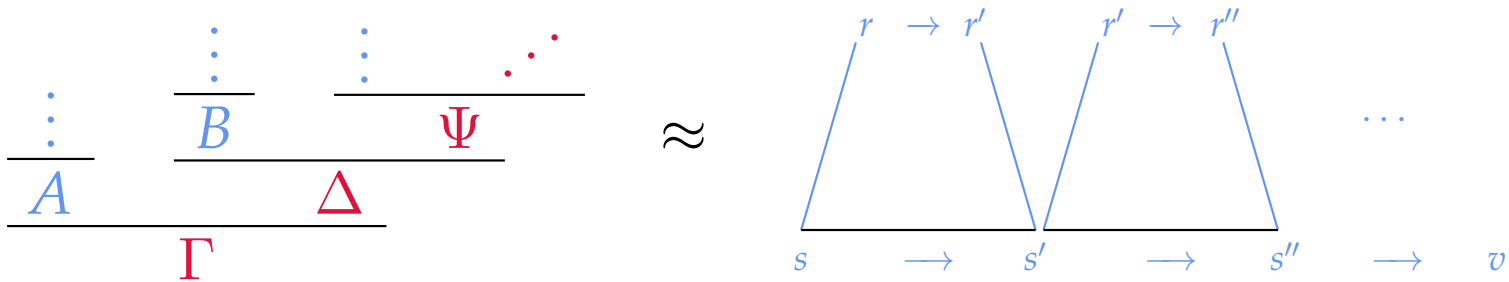
Evaluation in MSOS

Internalizing `steps` in MSOS:

$$\boxed{e \xrightarrow{L}{}^{*} v}$$

$$\frac{x \xrightarrow{L_1} y \qquad y \xrightarrow{L_2}{}^{*} v}{x \xrightarrow{L_2 \circ L_1}{}^{*} v}\ [\textsc{Trans}] \qquad \frac{}{v \xrightarrow{\{-\}}{}^{*} v}\ [\textsc{Refl-v}]$$

Derivation tree structure:



where $A, B, \ldots$ are ordinary transitions ($\rightarrow$) and
$\Gamma, \Delta, \ldots$ are transitive steps ($\rightarrow^{*}$)

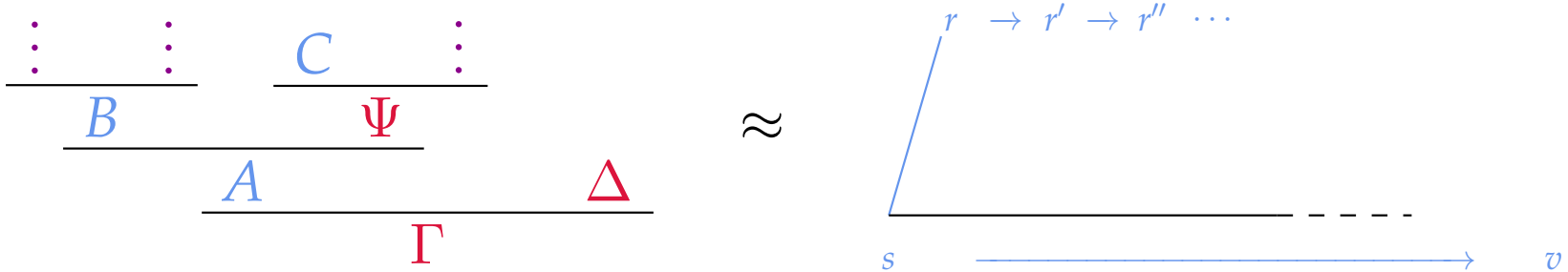
Refocused evaluation

$$\frac{x \xrightarrow{L_1} y \qquad y \xrightarrow{L_2}{}^{*} v}{x \xrightarrow{L_2 \circ L_1}{}^{*} v}\ [\textsc{Trans}] \qquad \frac{}{v \xrightarrow{\{-\}}{}^{*} v}\ [\textsc{Refl-v}]$$

Refocusing rule:

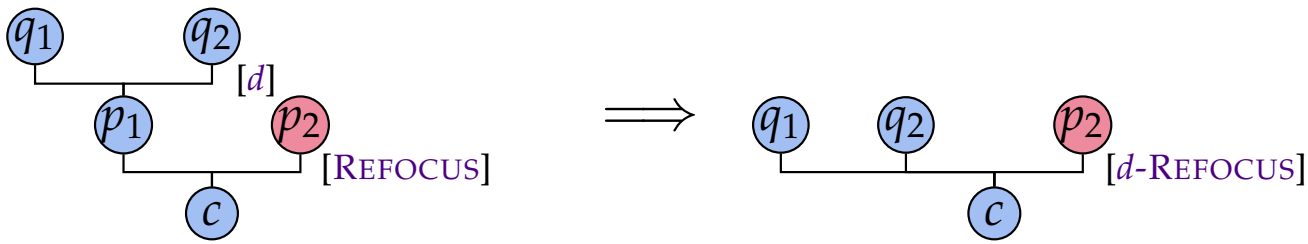
$$\frac{x \xrightarrow{L_1} y \qquad y \xrightarrow{L_2}{}^{*} v}{x \xrightarrow{L_2 \circ L_1} v}\ [\textsc{Refocus}]$$

Derivation tree structure:



Refocusing specialization

Deriving refocused rules:



Replacing ordinary rules by refocused rules forces sub-term evaluation inside derivation trees

Refocusing specialization of assign

Specializing

$$[\textsc{Refocus}]\ \frac{x \xrightarrow{L_1} y \qquad y \xrightarrow{L_2}{}^{*}\ v}{x \xrightarrow{L_2 \circ L_1} v}\ \text{wrt}\ \frac{e \xrightarrow{\{\ldots\}} e'}{\mathsf{assign}(ref, e) \xrightarrow{\{\ldots\}} \mathsf{assign}(ref, e')}\ [\textsc{asn1}]$$

$$\frac{\dfrac{e \xrightarrow{L_1} e'}{\mathsf{assign}(ref, e) \xrightarrow{L_1} \mathsf{assign}(ref, e')}\ [\textsc{asn1}] \qquad \mathsf{assign}(ref, e') \xrightarrow{L_2}{}^{*}\ v}{\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} v}\ [\textsc{Refocus}]$$

$$\Longrightarrow \quad \frac{e \xrightarrow{L_1} e' \qquad \mathsf{assign}(ref, e') \xrightarrow{L_2}{}^{*}\ v}{\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} v}\ [\textsc{asn1-Refocus}]$$

Refocusing specialization of assign

Specializing

$$[\textsc{Refocus}]\ \frac{x \xrightarrow{L_1} y \qquad y \xrightarrow{L_2}{}^{*}\ v}{x \xrightarrow{L_2 \circ L_1} v}\ \text{wrt}\ \frac{e \xrightarrow{\{\ldots\}} e'}{\mathsf{assign}(ref, e) \xrightarrow{\{\ldots\}} \mathsf{assign}(ref, e')}\ [\textsc{asn1}]$$

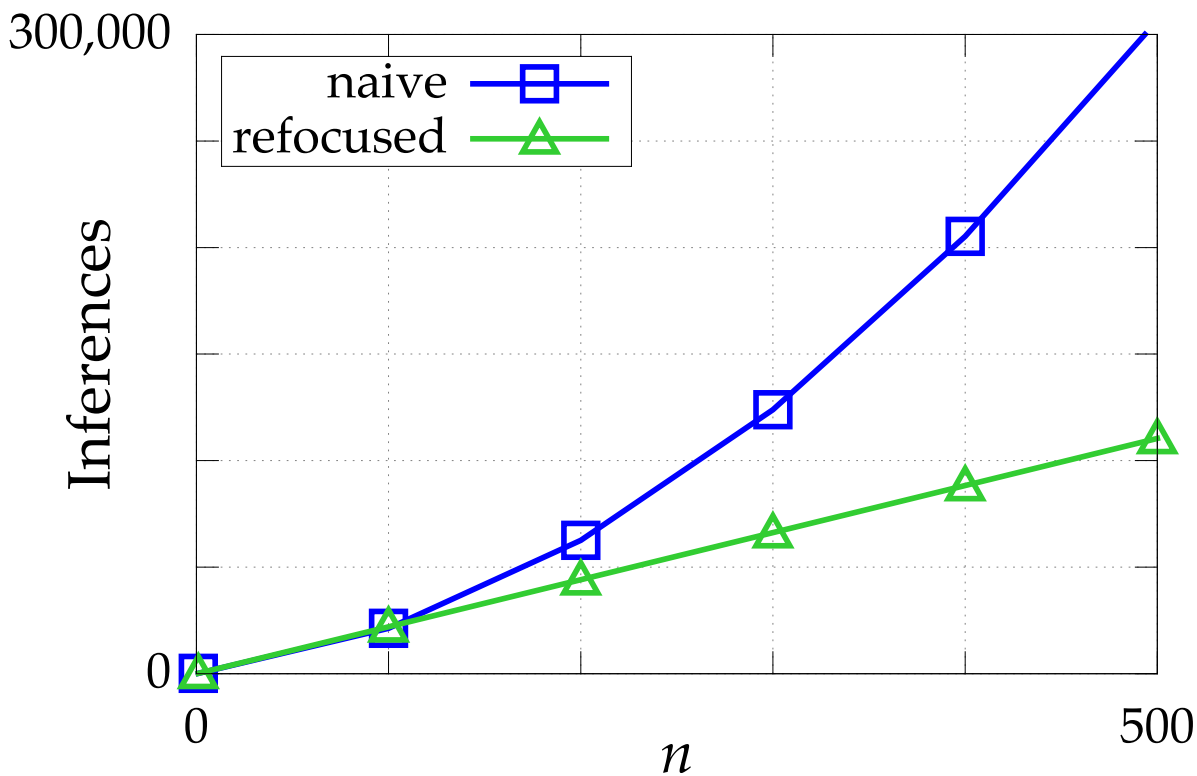
$$\frac{\dfrac{e \xrightarrow{L_1} e'}{\mathsf{assign}(ref, e) \xrightarrow{L_1} \mathsf{assign}(ref, e')}\ [\textsc{asn1}] \qquad \mathsf{assign}(ref, e') \xrightarrow{L_2}{}^{*}\ v}{\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} v}\ [\textsc{Refocus}]$$

$$\Longrightarrow \quad \frac{e \xrightarrow{L_1} e' \qquad \mathsf{assign}(ref, e') \xrightarrow{L_2}{}^{*}\ v}{\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} v}\ [\textsc{asn1-Refocus}]$$

Refocusing improves efficiency

$$\underbrace{\mathsf{assign}(\mathsf{x}_1, \mathsf{assign}(\mathsf{x}_2 \cdots \mathsf{assign}}_{n}(\mathsf{x}_n, 42) \cdots))$$

Refocused Prolog interpreter uses $O(n)$ inferences:



Refocused rules are big-step

$$\boxed{e \xrightarrow{L} v} \qquad \boxed{e \xrightarrow{L}{}^{*} v}$$

But refocused proofs require more inferences to reduce than classic big-step proofs

$$\dfrac{\dfrac{\dfrac{}{42 \xrightarrow{\{-\}} 42}\,[\textsc{Refl-v}] \qquad \sigma' = \sigma[ref \mapsto 42]}{\mathsf{assign}(\mathsf{x}, 42) \xrightarrow{\{\mathbf{sto}=\sigma, \mathbf{sto}'=\sigma', -\}} 42}\,[\textsc{Asn1-Big}] \qquad \sigma'' = \sigma'[ref \mapsto 42]}{\mathsf{assign}(\mathsf{x}, \mathsf{assign}(\mathsf{x}, 42)) \xrightarrow{\{\mathbf{sto}=\sigma, \mathbf{sto}'=\sigma'', -\}} 42}\,[\textsc{Asn1-Big}]$$

vs.

$$\dfrac{\dfrac{\dfrac{\sigma' = \sigma[ref \mapsto 42]}{\mathsf{assign}(\mathsf{x}, 42) \xrightarrow{\{\mathbf{sto}=\sigma, \mathbf{sto}'=\sigma', -\}}{}^{*} 42}\,[\textsc{Asn2-Refocus}] \qquad \dfrac{}{42 \xrightarrow{\{-\}}{}^{*} 42}\,[\textsc{Refl-v}]}{\mathsf{assign}(\mathsf{x}, 42) \xrightarrow{\{\mathbf{sto}=\sigma, \mathbf{sto}'=\sigma', -\}} 42}\,[\textsc{Asn1-Refocus}] \qquad \dfrac{}{42 \xrightarrow{\{-\}}{}^{*} 42}\,[\textsc{Refl-v}]}{\mathsf{assign}(\mathsf{x}, \mathsf{assign}(\mathsf{x}, 42)) \xrightarrow{\{\mathbf{sto}=\sigma, \mathbf{sto}'=\sigma'', -\}} 42}\,[\textsc{Asn1-Refocus}]$$

SOS vs. MSOS

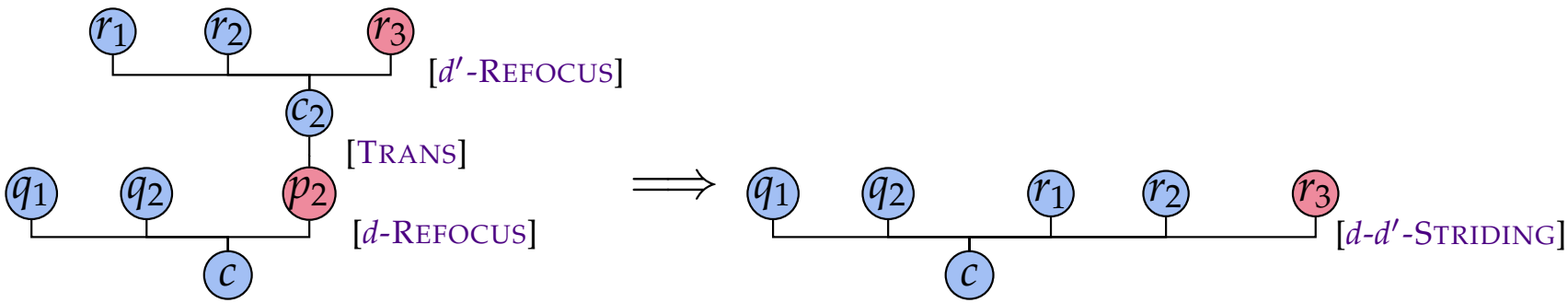
MSOS in Prolog

Refocusing in MSOS

Striding

Striding specialization

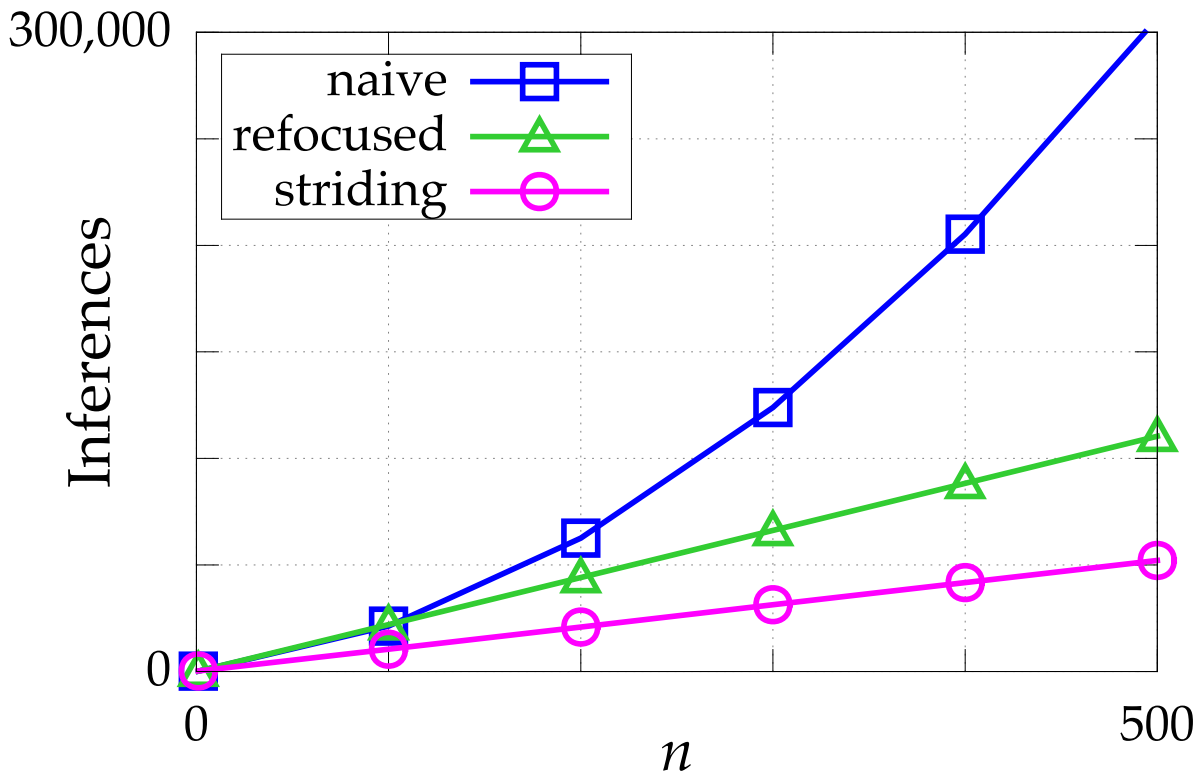
Transforming refocused rules into big-step rules



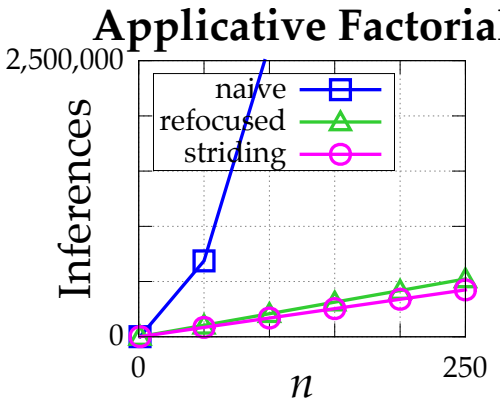
Striding gives big-step efficiency

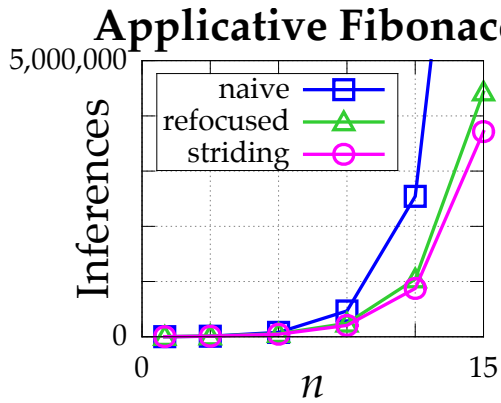
$$\underbrace{\mathsf{assign}(x_1, \mathsf{assign}(x_2 \cdots \mathsf{assign}}_{n}(x_n, 42) \cdots))$$

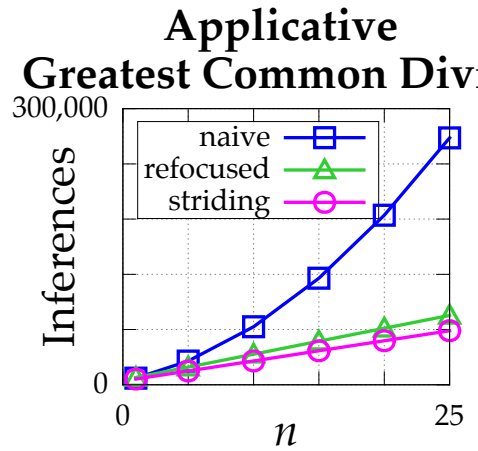
Big-step Prolog interpreter uses $O(n)$ inferences with reduced constant factor:

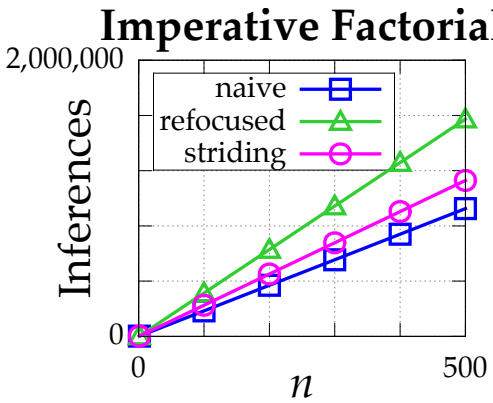


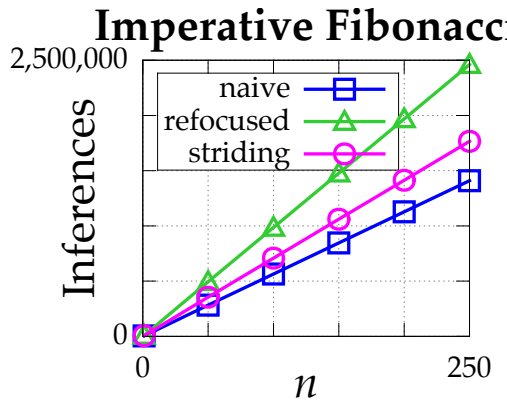
More benchmark experiments

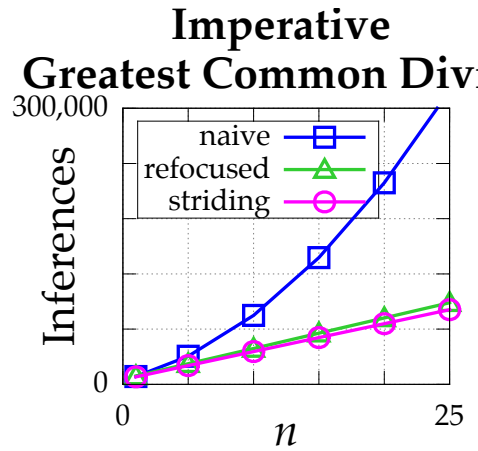












Works well for deeply nested terms

Extra label composition operations give slight overhead for shallowly nested terms

Further work

Case study: Caml Light

Proof of correctness in Coq

Relation to pretty-big-step semantics [Charguéraud, ESOP'13]

Conclusion

Specializing small-step MSOS specifications improves generated interpreters for MSOS

- Refocusing specialization reduces overhead of evaluating deeply nested terms
- Striding specialization gives classic big-step rules

Extra slides

Refocusing and abrupt termination

$$\frac{e \xrightarrow{\{\mathbf{blocked}'=1,\ldots\}} e'}{\mathsf{blocking}(e) \xrightarrow{\{\mathbf{blocked}'=0,\ldots\}} \mathsf{skip}} \qquad \frac{}{\mathsf{blocking}(v) \xrightarrow{\{-\}} \mathsf{skip}}$$

$$\frac{e \xrightarrow{\{\mathbf{blocked}'=0,\ldots\}} e'}{\mathsf{blocking}(e) \xrightarrow{\{\mathbf{blocked}'=0,\ldots\}} \mathsf{blocking}(e')} \qquad \frac{}{\mathsf{block} \xrightarrow{\{\mathbf{blocked}'=1,-\}} \mathsf{stuck}}$$

- add evaluates sub-terms to values in interleaved order; and
- loop diverges

What are possible outcomes of evaluating

$$\mathsf{blocking}(\mathsf{add}(\mathsf{block}, \mathsf{loop}))$$

under ordinary vs. refocused evaluation?

Forcing abrupt termination

Add label component ε such that abrupt termination changes the component from 0 to 1

$$\frac{e \xrightarrow{\{\varepsilon=0,\varepsilon'=1,\mathbf{blocked}'=1,\ldots\}} e'}{\mathsf{blocking}(e) \xrightarrow{\{\varepsilon=0,\varepsilon'=0,\mathbf{blocked}'=0,\ldots\}} \mathsf{skip}} \qquad \frac{}{\mathsf{blocking}(v) \xrightarrow{\{-\}} \mathsf{skip}}$$

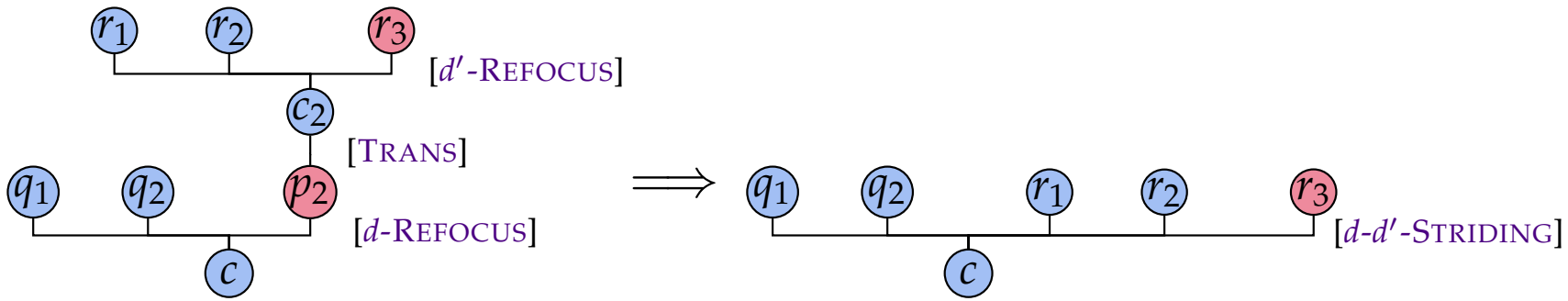
$$\frac{e \xrightarrow{\{\mathbf{blocked}'=0,\ldots\}} e'}{\mathsf{blocking}(e) \xrightarrow{\{\mathbf{blocked}'=0,\ldots\}} \mathsf{blocking}(e')} \qquad \frac{}{\mathsf{block} \xrightarrow{\{\varepsilon'=1,\mathbf{blocked}'=1,-\}} \mathsf{stuck}}$$

Updated evaluation rules:

$$\frac{x \xrightarrow{\{\varepsilon=0,X_1\}} y \qquad y \xrightarrow{L_2}{}^{*} z}{x \xrightarrow{L_2\circ\{\varepsilon=0,X_1\}}{}^{*} z}\ [\textsc{trans-}\varepsilon] \qquad \frac{x \xrightarrow{\{\varepsilon=0,X_1\}} y \qquad y \xrightarrow{L_2}{}^{*} z}{x \xrightarrow{L_2\circ\{\varepsilon=0,X_1\}} z}\ [\textsc{Refocus-}\varepsilon]$$

$$\frac{}{v \xrightarrow{\{\varepsilon=0,-\}}{}^{*} v}\ [\textsc{Refl-v-}\varepsilon] \qquad \frac{}{x \xrightarrow{\{\varepsilon=1,-\}}{}^{*} x}\ [\textsc{refl-}\varepsilon]$$

Striding Specialization

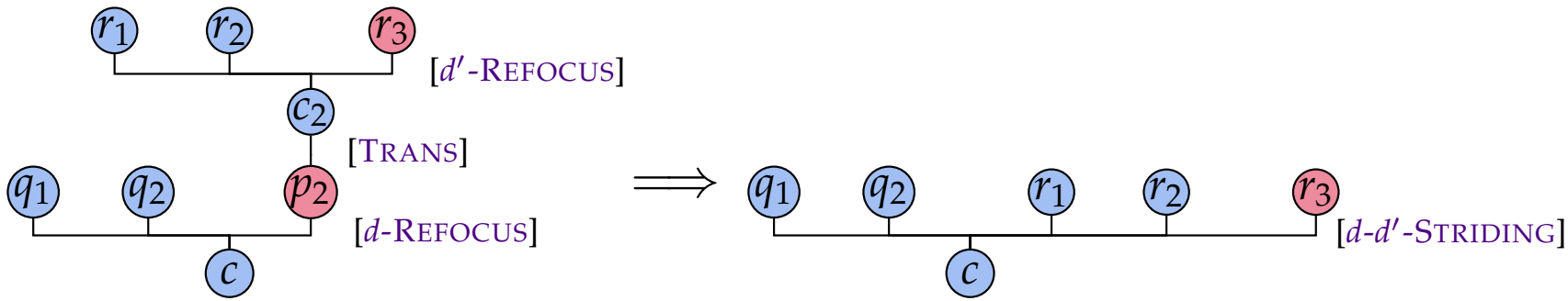


$$\dfrac{\dfrac{\sigma' = \sigma[ref \mapsto v]}{\mathsf{assign}(ref, v) \xrightarrow{\{\mathbf{sto}=\sigma, \mathbf{sto}'=\sigma'', -\}} \mathsf{skip}}\ [\text{ASN2-REFOCUS}]}{e \xrightarrow{L_1} v \qquad \mathsf{assign}(ref, v) \xrightarrow{L_2}{}^{*}\ \mathsf{skip}}\ [\text{TRANS}]$$

$$\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} \mathsf{skip}\ [\text{ASN1-REFOCUS}]$$

$$\Longrightarrow \dfrac{e \xrightarrow{L_1} v \qquad \sigma' = \sigma[ref \mapsto v]}{\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} \mathsf{skip}} \simeq \dfrac{e \xrightarrow{L_1}{}^{*}\ v \qquad \sigma' = \sigma[ref \mapsto v]}{\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} \mathsf{skip}}\ [\text{ASN1-ASN2-STRIDING}]$$

Striding Specialization



$$\dfrac{\dfrac{\sigma' = \sigma[ref \mapsto v]}{\mathsf{assign}(ref, v) \xrightarrow{\{\mathbf{sto}=\sigma, \mathbf{sto}'=\sigma'', -\}} \mathsf{skip}}\ [\text{ASN2-REFOCUS}]}{e \xrightarrow{L_1} v \qquad \mathsf{assign}(ref, v) \xrightarrow{L_2}{}^{*}\ \mathsf{skip}}\ [\text{TRANS}]$$

$$\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} \mathsf{skip}\ [\text{ASN1-REFOCUS}]$$

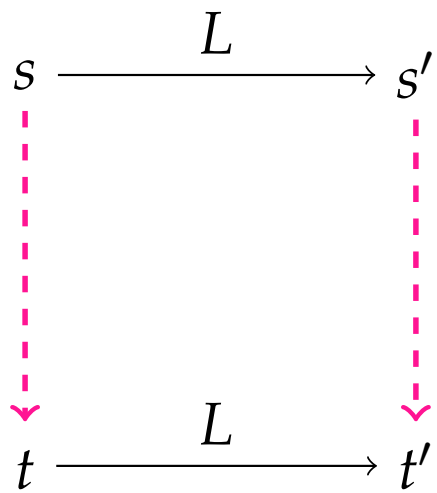
$$\Longrightarrow \dfrac{e \xrightarrow{L_1} v \qquad \sigma' = \sigma[ref \mapsto v]}{\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} \mathsf{skip}} \simeq \dfrac{e \xrightarrow{L_1}{}^{*}\ v \qquad \sigma' = \sigma[ref \mapsto v]}{\mathsf{assign}(ref, e) \xrightarrow{L_2 \circ L_1} \mathsf{skip}}\ [\text{ASN1-ASN2-STRIDING}]$$

Filtering Rules

Problem with Striding: Explosion in number of rules

E.g., [ASN1-ASN1-STRIDING], [ASN1-ASN2-STRIDING],
[ASN1-ASN1-ASN2-STRIDING], . . .

Solution: Filter rules using simulation



Prolog proof search using big-step rules

$$\frac{e_1 \xrightarrow{L_1}{}^{*} v_1 \qquad e_2 \xrightarrow{\{\mathbf{env}=\rho[id\mapsto v],\ldots\}}{}^{*} v_2}{\mathsf{let}(id, e_1, e_2) \xrightarrow{\{\mathbf{env}=\rho,\ldots\}\circ L_1} v_2}\ [\textsc{let3}]$$

$$\frac{e_1 \xrightarrow{L_1}{}^{*} v_1 \qquad e_2 \xrightarrow{\{\mathbf{env}=\rho[id\mapsto v],\ldots\}} e_2' \qquad \mathsf{let}(id, v_1, e_2') \xrightarrow{L_3}{}^{*} z}{\mathsf{let}(id, e_1, e_2) \xrightarrow{L_3\circ\{\mathbf{env}=\rho,\ldots\}\circ L_1} z}\ [\textsc{let2}]$$

What happens when e_2 yields a stuck term in [LET3]?

Minimizing back-tracking in big-step interpreters

Prolog interpreters generated from big-step style rules result in excessive back-tracking.

Solution: Left-factoring

$$\begin{array}{l} H \leftarrow A \wedge B \\ H \leftarrow A \wedge C \end{array} \quad \Longrightarrow \quad H \leftarrow A \wedge (B \vee C)$$