Typable Fragments of Polynomial Automatic Amortized Resource Analysis

Long Pham and Jan Hoffmann

Carnegie Mellon University

January 28th, 2021

AARA is a type-based resource analysis

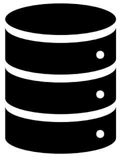
Automatic Amortized Resource Analysis

Programs

Type inference of AARA

Polynomial costs bounds

Time



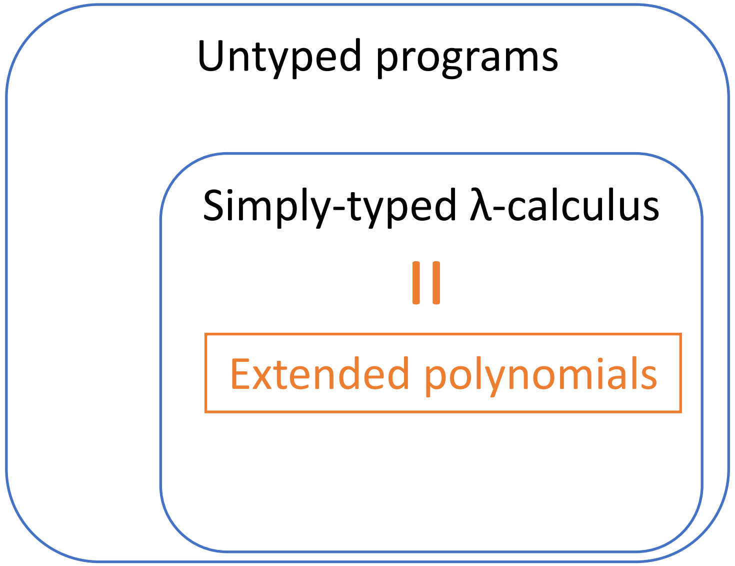
Memory

Power

Two questions about any type systems

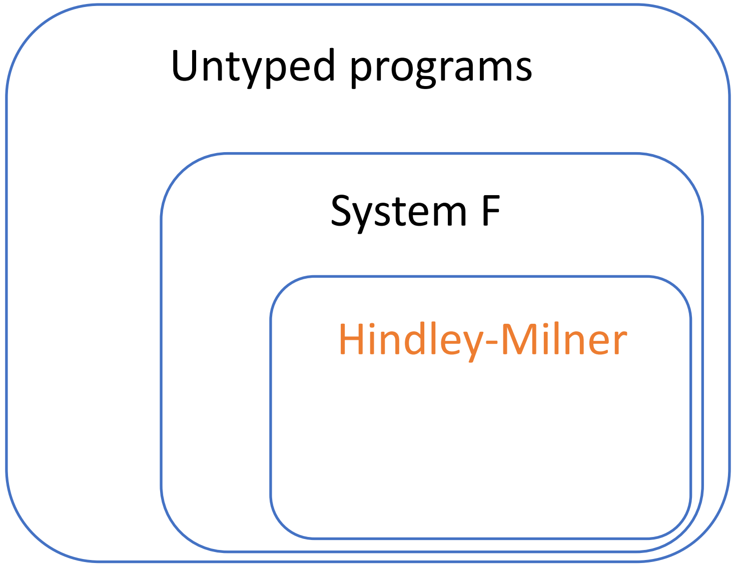
1. Semantic characterization of typable programs?

Example:

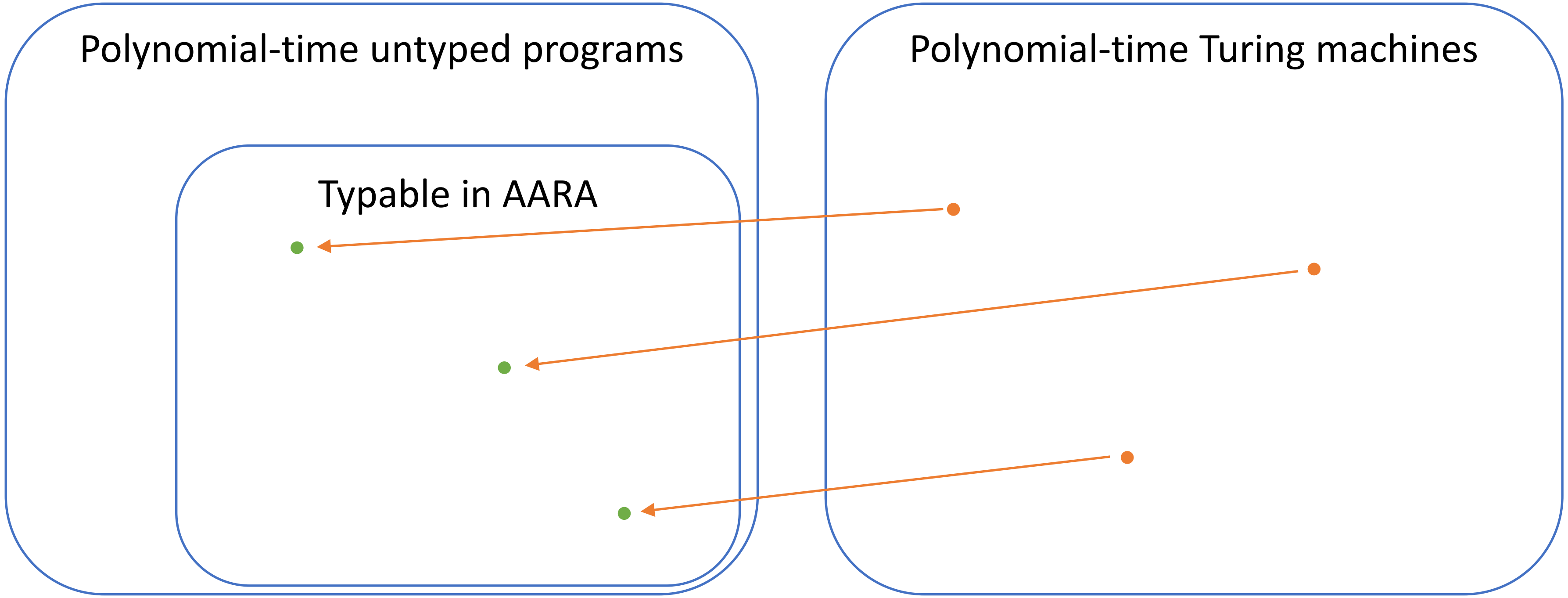


2. Sufficient condition for typability?

Example:

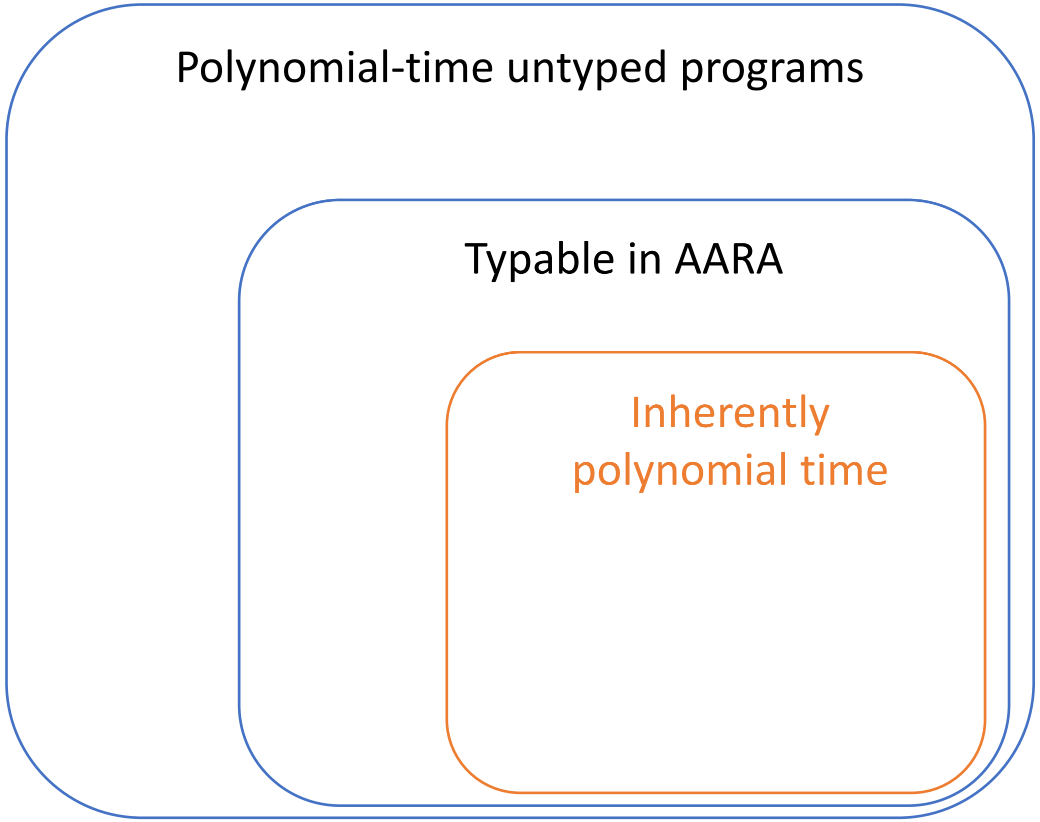


First contribution: semantic characterization of AARA



1. Input-output relations remain identical.
2. Cost bound is larger than or equal to the original running time.

Second contribution: sufficient condition for AARA's typability



The sufficient condition should be

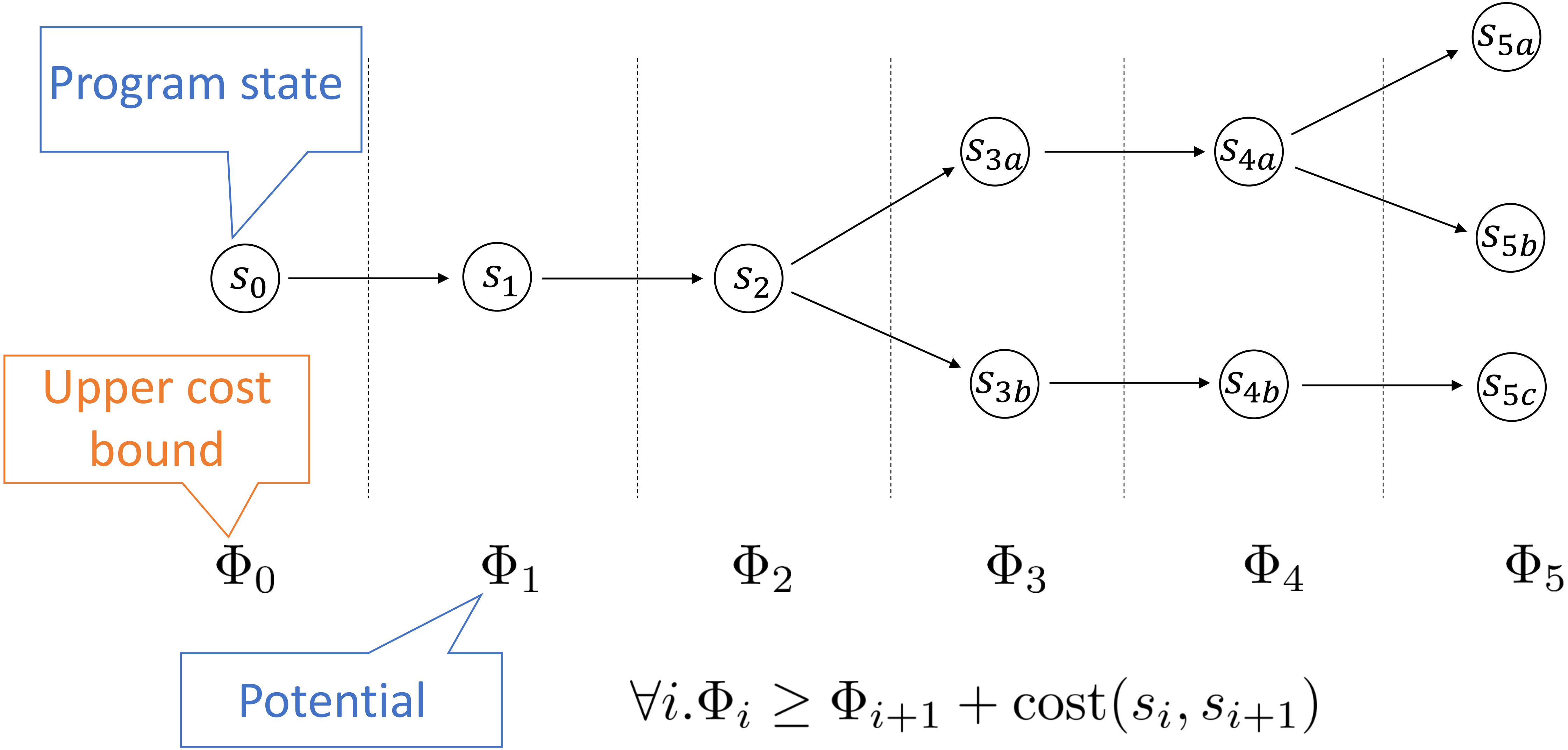
1. not identical to AARA
2. easy to understand

This talk

- ☑ Motivation
- ☐ Basic idea of AARA
- ☐ Sufficient condition for AARA's typability
- ☐ Challenges in the typability proof

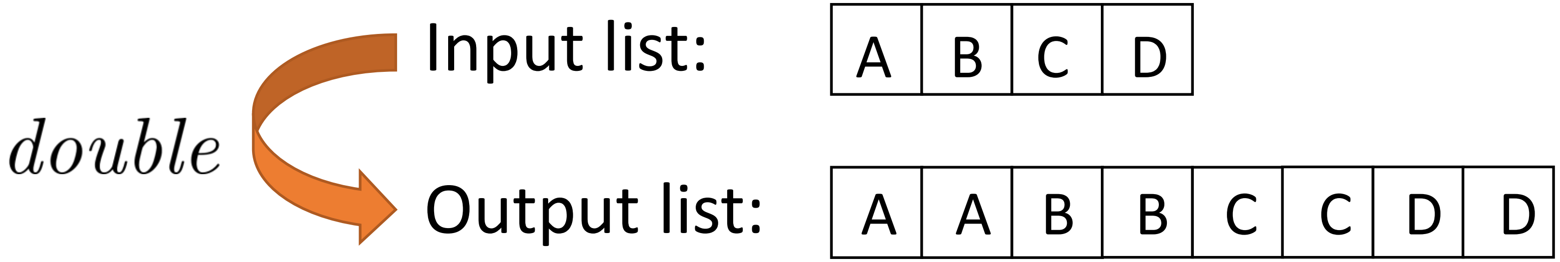
Semantic characterization of AARA is in the paper

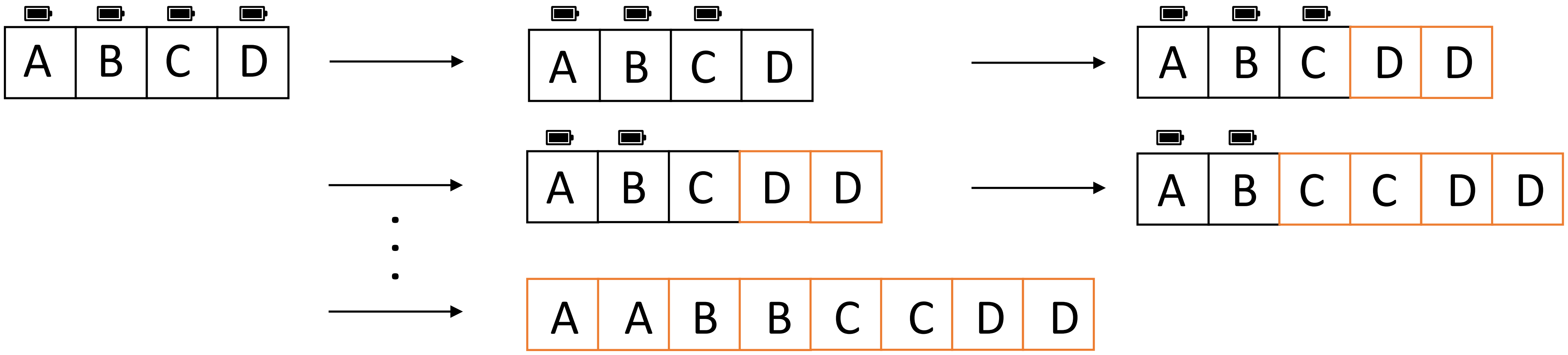
AARA uses the potential method



$$\forall i. \Phi_i \geq \Phi_{i+1} + \mathrm{cost}(s_i, s_{i+1})$$

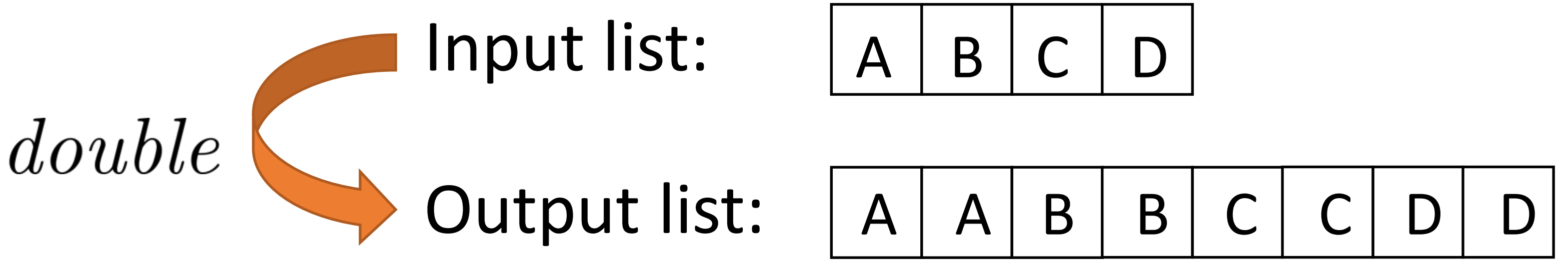
AARA uses the potential method

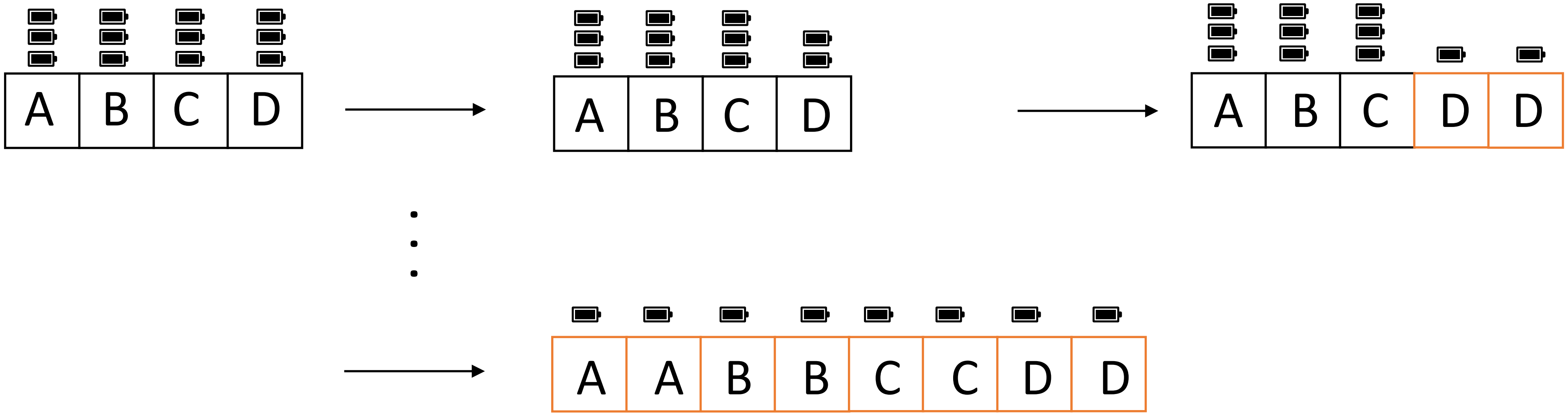




$$double : L^1(b) \to L^0(b)$$

AARA uses the potential method





$$double : L^3(b) \to L^1(b)$$

AARA: benefits and expressive power

Benefits:

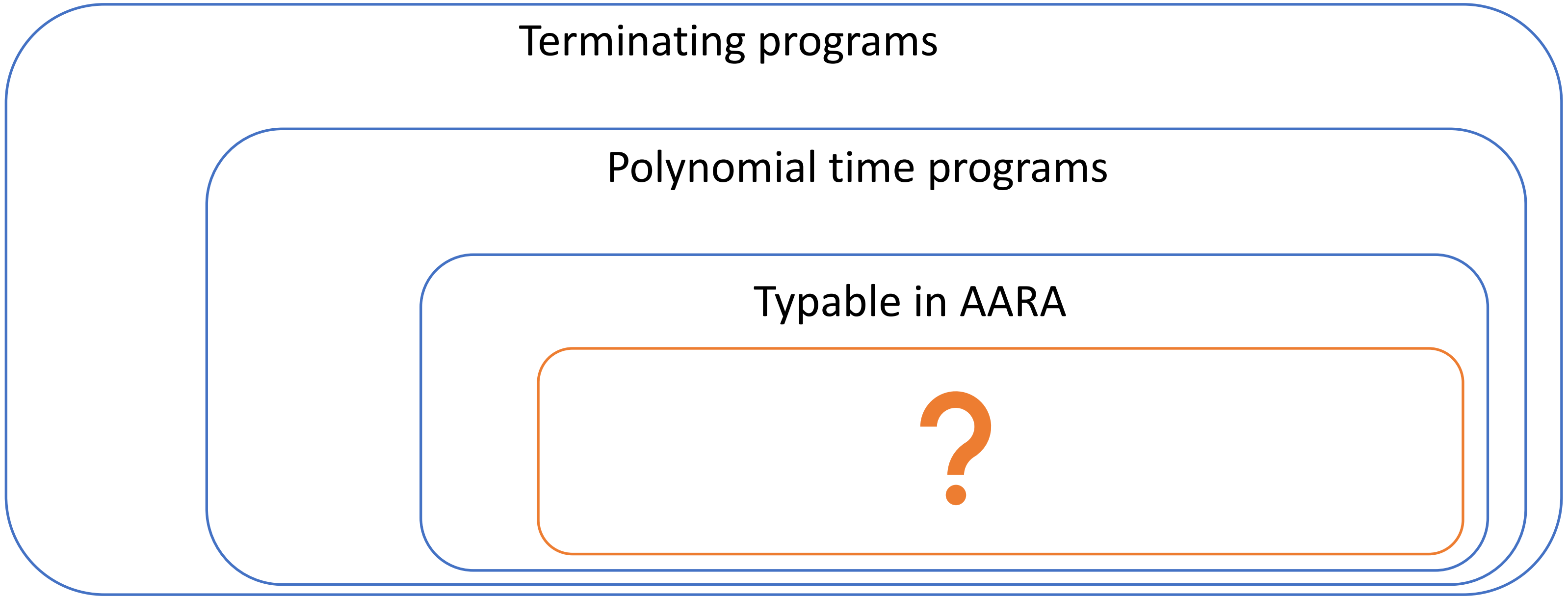
1. Automatic type inference by LP solving
2. Precision by amortized analysis
3. Soundness
4. Certification in the form of type derivations

Programs	Multivariate polynomial cost bounds
$double\ x$	$\lvert x \rvert$
$multiply\ x\ y$	$\lvert x \rvert \cdot \lvert y \rvert$

This talk

- ☑ Motivation
- ☑ Basic idea of AARA
- ☐ Sufficient condition for AARA’s typability
- ☐ Challenges in the typability proof

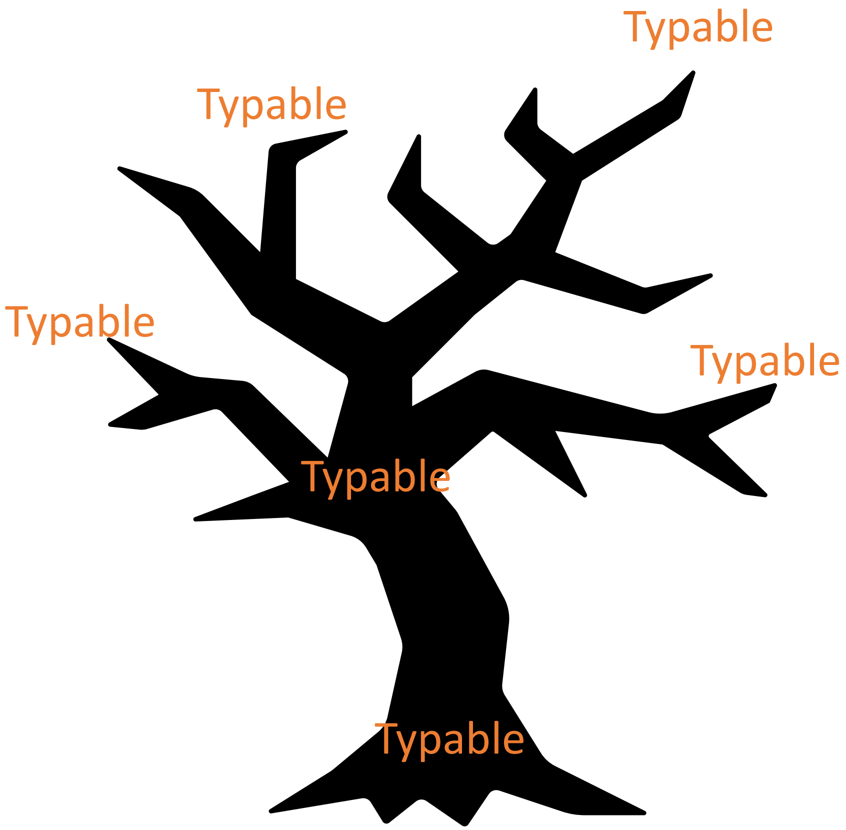
Inherently polynomial time: termination



1. Primitive recursion instead of general recursion.

2a. The program must be polynomial-time.

Inherently polynomial time: compositionality



2b. Every subexpression must be polynomial time.

Inherently polynomial time: primitive recursion

$\mathtt{FV}(e_1) \subseteq \{y, ys, z\}$

Primitive recursion has the form

$$e := \textsf{rec}\ x\ \{[\,] \hookrightarrow e_0 \mid (y :: ys)\ \textsf{with}\ z \hookrightarrow e_1\}$$

Recursive result

Question

If e_0 and e_1 are polynomial time, is e always also polynomial time?

Inherently polynomial time: primitive recursion

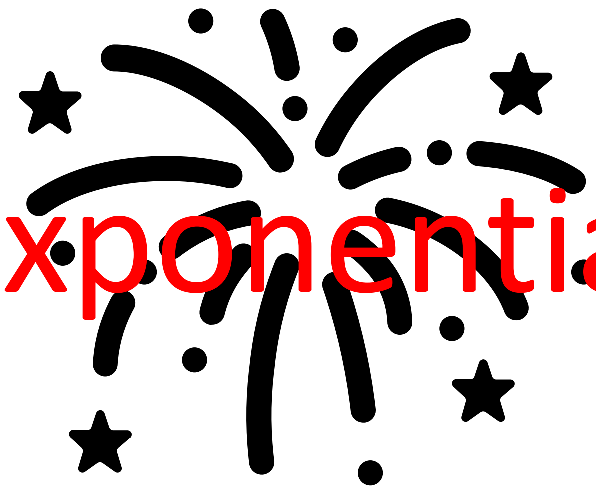
Recursive result

$$e := \mathsf{rec}\ x\ \{[\,] \hookrightarrow e_0 \mid (y :: ys)\ \mathsf{with}\ z \hookrightarrow append\ \langle z, z \rangle\}$$

×2

Stepping function

e_0 ×2 ×2 ×2



Exponential

Inherently polynomial time: primitive recursion

Recursive result

$$e := \mathsf{rec}\ x\ \{[\,] \hookrightarrow e_0 \mid (y :: ys)\ \mathsf{with}\ z \hookrightarrow append\ \langle z, z \rangle\}$$

×2

Stepping function

3. In a primitive recursion, the stepping function's running time is constant in $|z|$.

Inherently polynomial time: summary

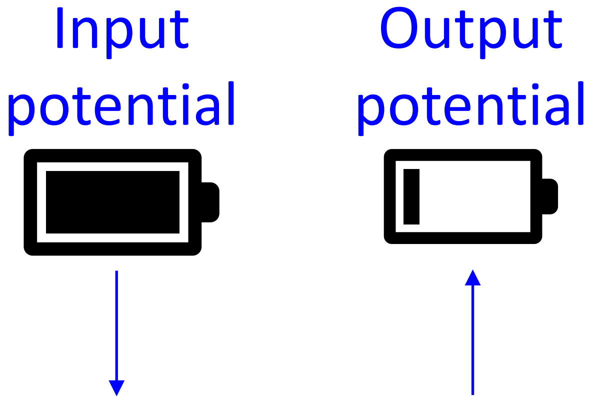
1. Primitive recursion instead of general recursion.
2b. Every subexpression must be polynomial time.
3. In a primitive recursion, the stepping function's running time is constant in $|z|$.

This talk

- ☑ Motivation
- ☑ Basic idea of AARA
- ☑ Sufficient condition for AARA's typability
- ☐ Challenges in the typability proof

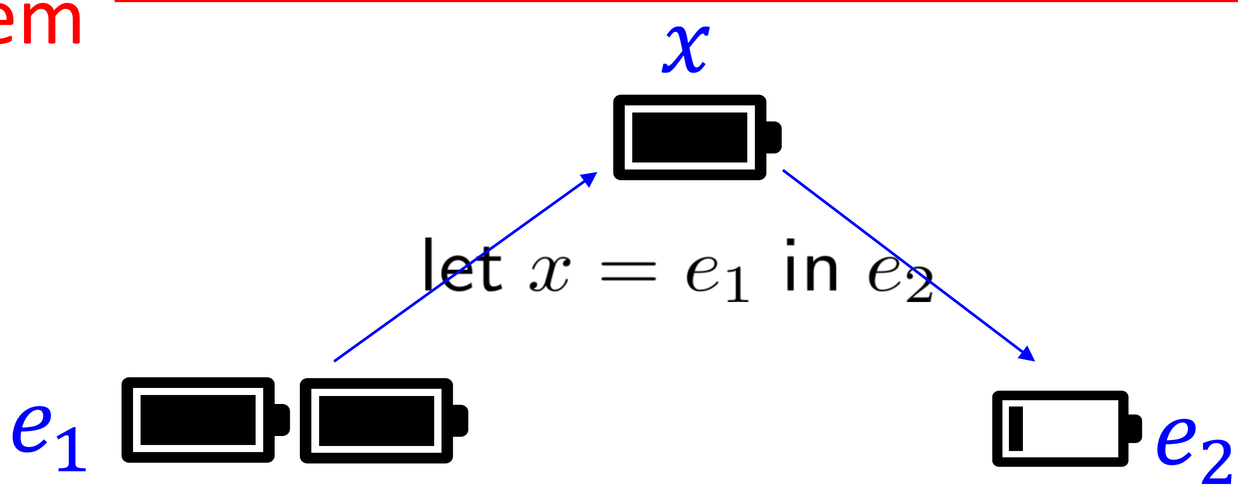
Naïve theorem is not compositional

Goal



e is inherently polynomial time → $\exists P, Q.\, (\Gamma; P \vdash e : \langle b, Q \rangle)$

Problem



We need a stronger theorem

New Goal

e is inherently polynomial time $\Rightarrow$ $\forall Q. \exists P. (\Gamma; P \vdash e : \langle b, Q \rangle)$

How do we find a loop invariant?

New Goal

Input potential

Output potential

e is inherently polynomial time

$$\forall Q . \exists P . \, (\Gamma ; P \vdash e : \langle b, Q \rangle)$$

Problem

Recursive result

$$\text{rec } x \ \{[\,] \hookrightarrow e_0 \mid (y :: ys) \text{ with } z \hookrightarrow e_1\}$$

Stepping function

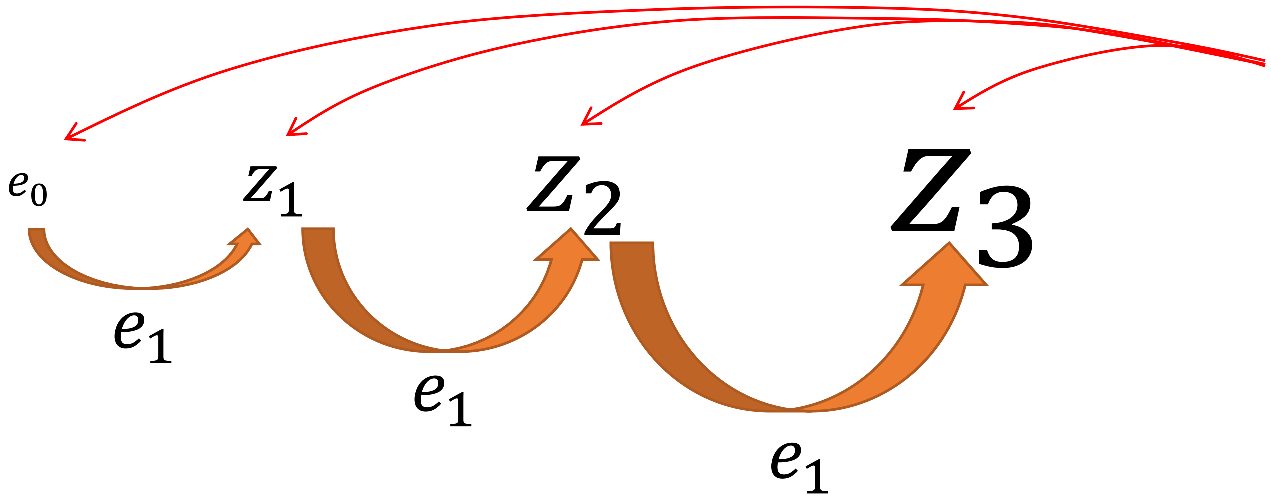
How do we find a loop invariant?

Problem

Recursive result

$$\text{rec } x \ \{[\,] \hookrightarrow e_0 \mid (y :: ys) \text{ with } z \hookrightarrow e_1\}$$

Stepping function



Identical resource-annotated types

=

Loop invariant

We need uniform resource annotations

$$\text{rec } x \ \{[\,] \hookrightarrow e_0 \mid (y :: ys) \text{ with } z \hookrightarrow e_1\}$$

Special case

$y :$  $, ys :$  $, z :$  $\vdash e_1 :$ 

e_1 runs in constant time in $|z|$.

Uniform resource-annotated types of YELLOW with respect to $v_1, \ldots, v_n$.

General case

$x_1 :$  $, \ldots, x_m :$  $, v_1 :$  $, \ldots, v_n :$  $\vdash e :$ 

e runs in constant time in $|v_i|$.

Contributions

1. Embedding of polynomial-time Turing machines in AARA
2. Definition of inherently polynomial time
3. Typability proof under two restrictions:
   - Variable sharing
   - Nested lists.