Polynomialization of Ordinary Differential Equations given by Straight-line programs

Arnaud Minondo

Joint work with Grégoire LECERF and Joris van der HOEVEN

Computer Science Laboratory of the École polytechnique

CNRS, École polytechnique, Institut Polytechnique de Paris

ISSAC, Oldenburg

July 16, 2026

Polynomialization example

Initial Value Problem (IVP):

$$\begin{cases} y' = y^{-1} \\ y(0) = \sqrt{2} \end{cases}$$

Polynomialization example

Initial Value Problem (IVP):

$$\begin{cases} y' = y^{-1} \\ y(0) = \sqrt{2} \end{cases}$$

Euler step

$$\begin{cases} y_0 = \sqrt{2} \\ y_{n+1} = y_n + h y_n^{-1} \text{ for all } n \in \mathbb{N}, \end{cases}$$

Polynomialization example

Initial Value Problem (IVP):

$$\begin{cases} y' = y^{-1} \\ y(0) = \sqrt{2} \end{cases}$$

Euler step

$$\begin{cases} y_0 = \sqrt{2} \\ y_{n+1} = y_n + h y_n^{-1} \text{ for all } n \in \mathbb{N}, \end{cases}$$

uses an inversion
$\approx$8 CPU cycles

Polynomialization example

Initial Value Problem (IVP):

$$\begin{cases} y' = y^{-1} \\ y(0) = \sqrt{2} \end{cases}$$

Euler step

$$\begin{cases} y_0 = \sqrt{2} \\ y_{n+1} = y_n + h y_n^{-1} \text{ for all } n \in \mathbb{N}, \end{cases}$$

uses an inversion
≈ 8 CPU cycles

Polynomialize, let $z = y^{-1}$ then $z' = -y' y^{-2} = -y^{-3} = -z^3$.

Polynomialization example

Initial Value Problem (IVP):

$$\begin{cases} y' = y^{-1} \\ y(0) = \sqrt{2} \end{cases}$$

Euler step

$$\begin{cases} y_0 = \sqrt{2} \\ y_{n+1} = y_n + h y_n^{-1} \text{ for all } n \in \mathbb{N}, \end{cases}$$

uses an inversion
≈8 CPU cycles

Polynomialize, let $z = y^{-1}$ then $z' = -y' y^{-2} = -y^{-3} = -z^3$.

Polynomial IVP:

$$\begin{cases} y(0) &= \sqrt{2} \\ z(0) &= \sqrt{2}^{-1} \\ y' &= z \\ z' &= -z^3 \end{cases}$$

Polynomialization example

Initial Value Problem (IVP):

$$\begin{cases} y' = y^{-1} \\ y(0) = \sqrt{2} \end{cases}$$

Euler step

$$\begin{cases} y_0 = \sqrt{2} \\ y_{n+1} = y_n + h y_n^{-1} \text{ for all } n \in \mathbb{N}, \end{cases}$$

uses an inversion
≈8 CPU cycles

Polynomialize, let $z = y^{-1}$ then $z' = -y' y^{-2} = -y^{-3} = -z^3$.

Polynomial IVP:

$$\begin{cases} y(0) &= \sqrt{2} \\ z(0) &= \sqrt{2}^{-1} \\ y' &= z \\ z' &= -z^3 \end{cases}$$

Euler step:

$$\begin{cases} y_0 &= \sqrt{2} \\ z_0 &= \sqrt{2}^{-1} \\ y_{n+1} &= y_n + h z_n \\ z_{n+1} &= z_n - h z_n^3 \end{cases}$$

Polynomialization example

Initial Value Problem (IVP):

$$\begin{cases} y' = y^{-1} \\ y(0) = \sqrt{2} \end{cases}$$

Euler step

$$\begin{cases} y_0 = \sqrt{2} \\ y_{n+1} = y_n + h y_n^{-1} \text{ for all } n \in \mathbb{N}, \end{cases}$$

uses an inversion
≈8 CPU cycles

Polynomialize, let $z = y^{-1}$ then $z' = -y' y^{-2} = -y^{-3} = -z^3$.

Polynomial IVP:

$$\begin{cases} y(0) &= \sqrt{2} \\ z(0) &= \sqrt{2}^{-1} \\ y' &= z \\ z' &= -z^3 \end{cases}$$

Euler step:

$$\begin{cases} y_0 &= \sqrt{2} \\ z_0 &= \sqrt{2}^{-1} \\ y_{n+1} &= y_n + h z_n \\ z_{n+1} &= z_n - h z_n^3 \end{cases}$$

use only ring instructions ≈.5 CPU cycles

Polynomialization examples

Pendulum, mass m, length l:

$$-\frac{g}{l}\sin(\theta) = \theta'' \iff \begin{cases} \theta' &= u, \\ u' &= -\frac{g}{l}\sin(\theta). \end{cases}$$

Polynomialization examples

Pendulum, mass m, length l:

$$-\frac{g}{l}\sin(\theta) = \theta'' \iff \begin{cases} \theta' &= u, \\ u' &= -\frac{g}{l}\sin(\theta). \end{cases}$$

sin hard to compute ≈34 CPU cycles.

Polynomialization examples

Pendulum, mass m, length l:

$$-\frac{g}{l}\sin(\theta) = \theta'' \iff \begin{cases} \theta' &= u, \\ u' &= -\frac{g}{l}\sin(\theta). \end{cases}$$

sin hard to compute $\approx$34 CPU cycles.

Polynomialize, let $v = \sin(\theta)$ and $w = \cos(\theta)$:

Polynomialization examples

Pendulum, mass m, length l:

$$-\frac{g}{l}\sin(\theta) = \theta'' \iff \begin{cases} \theta' &= u, \\ u' &= -\frac{g}{l}\sin(\theta). \end{cases}$$

sin hard to compute $\approx$34 CPU cycles.

Polynomialize, let $v = \sin(\theta)$ and $w = \cos(\theta)$:

$$\begin{cases} \theta' &= u \\ u' &= -\frac{g}{l}v \\ v' &= uw \\ w' &= -uv \end{cases}$$

Polynomialization examples

Pendulum, mass m, length l:

$$-\frac{g}{l}\sin(\theta) = \theta'' \iff \begin{cases} \theta' &= u, \\ u' &= -\frac{g}{l}\sin(\theta). \end{cases}$$

sin hard to compute ≈34 CPU cycles.

Polynomialize, let $v = \sin(\theta)$ and $w = \cos(\theta)$:

$$\begin{cases} \theta' &= u \\ u' &= -\frac{g}{l}v \\ v' &= uw \\ w' &= -uv \end{cases}$$

≈2 CPU cycles; easier to compute !

Polynomialization

A **polynomialization** of $(E): y' = \Phi(y)$ is a set of functions $w := w(y)$ such that:

$$(\tilde{E}): \begin{pmatrix} y \\ w \end{pmatrix}' = \Pi\left(\begin{pmatrix} y \\ w \end{pmatrix}\right), \Pi \in \mathbb{K}[y_1, \ldots, y_n, w_1, \ldots, w_m]^{n+m}.$$

Polynomializing

Previous work:

- The canonical form of a system of differentialy algebraic equations. APPELROTH, 1902
- Computability with polynomial differential equations. GRAÇA *et al.* 2008.
- Compiling elementary mathematical functions into finite chemical reaction networks via a polynomialization algorithm for ODEs. HEMERY *et al.* 2021.

Polynomializing

Previous work:

- The canonical form of a system of differentialy algebraic equations. APPELROTH, 1902
- Computability with polynomial differential equations. GRAÇA *et al.* 2008.
- Compiling elementary mathematical functions into finite chemical reaction networks via a polynomialization algorithm for ODEs. HEMERY *et al.* 2021.

Motivations for polynomializing:

- HPC, Numerical integration: Trigonometric functions not available on hardware!
- First step for more complicated problems:
  - Quadratization, BYCHKOV *et al.* 2023, CAI *et al.* 2024, HEMERY *et al.* 2020
  - Computability, BOURNEZ *et al.* 2007.

Straight-Line Program (SLP)

An **SLP Γ** over Σ, is defined by:

- a finite set of variables $\mathcal{V} \subset \mathbb{N}$
- sequence of instructions $\Gamma_1, \ldots, \Gamma_{|\Gamma|}$ of the form:

$$\Gamma_i \equiv X_i := \mathrm{op}_i(X_{i,1}, \ldots, X_{i,|\mathrm{op}_i|}),$$

where $\mathrm{op}_i \in \Sigma$, and $X_i, X_{i,1}, \ldots, X_{i,|\mathrm{op}_i|} \in \mathcal{V}$.

Straight-Line Program (SLP)

An **SLP Γ** over Σ, is defined by:

- a finite set of variables $\mathcal{U} \subset \mathbb{N}$
- sequence of instructions $\Gamma_1, \ldots, \Gamma_{|\Gamma|}$ of the form:

$$\Gamma_i \equiv X_i := \mathrm{op}_i(X_{i,1}, \ldots, X_{i,|\mathrm{op}_i|}),$$

where $\mathrm{op}_i \in \Sigma$, and $X_i, X_{i,1}, \ldots, X_{i,|\mathrm{op}_i|} \in \mathcal{U}$.

Fixed Instruction sets:
$\Sigma_{\mathrm{ring}} := \{+, -, \times, \mathrm{fma}, \mathrm{fms}, \mathrm{fnma}, \mathrm{fnms}\}$
$\Sigma_{\mathrm{div}} := \{/, (\cdot)^{-1}\}$
$\Sigma_{\mathrm{special}} := \{\text{Special functions applications}\}$
$\Sigma := \Sigma_{\mathrm{ring}} \cup \Sigma_{\mathrm{div}} \cup \Sigma_{\mathrm{special}}$

Special function
Function solution of a polynomial ODE

Straight-Line Program (SLP)

An **SLP Γ** over Σ, is defined by:

- a finite set of variables $\mathcal{U} \subset \mathbb{N}$
- sequence of instructions $\Gamma_1, \ldots, \Gamma_{|\Gamma|}$ of the form:

$$\Gamma_i \equiv X_i := \mathrm{op}_i(X_{i,1}, \ldots, X_{i,|\mathrm{op}_i|}),$$

where $\mathrm{op}_i \in \Sigma$, and $X_i, X_{i,1}, \ldots, X_{i,|\mathrm{op}_i|} \in \mathcal{U}$.

Fixed Instruction sets:
$\Sigma_{\mathrm{ring}} := \{+, -, \times, \mathrm{fma}, \mathrm{fms}, \mathrm{fnma}, \mathrm{fnms}\}$
$\Sigma_{\mathrm{div}} := \{/, (\cdot)^{-1}\}$
$\Sigma_{\mathrm{special}} := \{\text{Special functions applications}\}$
$\Sigma := \Sigma_{\mathrm{ring}} \cup \Sigma_{\mathrm{div}} \cup \Sigma_{\mathrm{special}}$

Special function
Function solution of a polynomial ODE

Execution Cost $\kappa : \Sigma \to \mathbb{N}$ (in cycles)
$\kappa(*) := .5, * \in \Sigma_{\mathrm{ring}}$
$\kappa(/) = \kappa(\cdot^{-1}) := 8,$
$\kappa(\sigma) \geqslant 16, \sigma \in \Sigma_{\mathrm{special}}.$

Straight-Line Program (SLP)

An **SLP Γ** over Σ, is defined by:

- a finite set of variables $\mathcal{U} \subset \mathbb{N}$
- sequence of instructions $\Gamma_1, \ldots, \Gamma_{|\Gamma|}$ of the form:

$$\Gamma_i \equiv X_i := \mathrm{op}_i(X_{i,1}, \ldots, X_{i,|\mathrm{op}_i|}),$$

where $\mathrm{op}_i \in \Sigma$, and $X_i, X_{i,1}, \ldots, X_{i,|\mathrm{op}_i|} \in \mathcal{U}$.

The **cost of execution** of the SLP is:

$$\kappa(\Gamma) := \sum_{i=1}^{l} \kappa(\mathrm{op}_i).$$

The number of instructions Σ_{kind} in Γ is $|\Gamma|_{\mathbf{kind}}$.

Fixed Instruction sets:
$\Sigma_{\mathrm{ring}} := \{+, -, \times, \mathrm{fma}, \mathrm{fms}, \mathrm{fnma}, \mathrm{fnms}\}$
$\Sigma_{\mathrm{div}} := \{/, (\cdot)^{-1}\}$
$\Sigma_{\mathrm{special}} := \{\text{Special functions applications}\}$
$\Sigma := \Sigma_{\mathrm{ring}} \cup \Sigma_{\mathrm{div}} \cup \Sigma_{\mathrm{special}}$

Special function
Function solution of a polynomial ODE

Execution Cost $\kappa : \Sigma \to \mathbb{N}$ (in cycles)
$\kappa(*) := .5, * \in \Sigma_{\mathrm{ring}}$
$\kappa(/) = \kappa(\cdot^{-1}) := 8,$
$\kappa(\sigma) \geqslant 16, \sigma \in \Sigma_{\mathrm{special}}.$

Polynomialization theorem

Theorem (Differential algebra folklore)
Let Γ be any SLP over $\Sigma = \Sigma_{\text{ring}} \cup \Sigma_{\text{div}} \cup \Sigma_{\text{special}}$ then the ODE $y' = \Gamma(y)$ is polynomializable.

Polynomialization theorem

Theorem (Differential algebra folklore)
Let Γ be any SLP over $\Sigma = \Sigma_{\text{ring}} \cup \Sigma_{\text{div}} \cup \Sigma_{\text{special}}$ then the ODE $y' = \Gamma(y)$ is polynomializable.

Running example:

IVP

$$(E): \begin{cases} y(0) = y_0 \\ z(0) = z_0 \\ y' = 5\frac{y}{z} + z \\ z' = \cos(y) \end{cases}$$

can be polynomialized.

SLP Γ

Input I_1, I_2
Output O_1, O_2
$X_1 := I_1 / I_2$
$O_1 := \text{fma}(5, X_1, I_2)$
$O_2 := \cos(I_1)$

$\xrightarrow{?}$

IVP SLPs Π polynomial, Φ

$$(\bar{E}): \begin{cases} (y,z,w)(0) = \Phi(y_0, z_0) \\ (y,z,w)' = \Pi(y,z,w) \end{cases}$$

Polynomialization theorem

Theorem (Differential algebra folklore)
Let Γ be any SLP over $\Sigma = \Sigma_{\text{ring}} \cup \Sigma_{\text{div}} \cup \Sigma_{\text{special}}$ then the ODE $y' = \Gamma(y)$ is polynomializable.

Running example:

IVP

$$(E): \begin{cases} y(0) = y_0 \\ z(0) = z_0 \\ y' = 5\frac{y}{z} + z \\ z' = \cos(y) \end{cases}$$

can be polynomialized.

SLP Γ

Input I_1, I_2
Output O_1, O_2
$X_1 := I_1 / I_2$
$O_1 := \text{fma}(5, X_1, I_2)$
$O_2 := \cos(I_1)$

$\xrightarrow{?}$

IVP SLPs Π polynomial, Φ

$$(\bar{E}): \begin{cases} (y,z,w)(0) = \Phi(y_0, z_0) \\ (y,z,w)' = \Pi(y,z,w) \end{cases}$$

From Γ, how to obtain Π, Φ such that $(E) \iff (\bar{E})$?

Our contribution: the algorithm

Assumption: For each instruction $\sigma \in \Sigma_{\text{special}}$, there exist $\mu_\sigma \in \mathbb{N}$ and Q_σ SLP such that:

$$(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t))' = Q_\sigma(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t), t)$$

Our contribution: the algorithm

Assumption: For each instruction $\sigma \in \Sigma_{\text{special}}$, there exist $\mu_\sigma \in \mathbb{N}$ and Q_σ SLP such that:

$$(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t))' = Q_\sigma(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t), t)$$

Lift: a mapping that associates to an instruction another sequence of instructions

Our contribution: the algorithm

Assumption: For each instruction $\sigma \in \Sigma_{\text{special}}$, there exist $\mu_\sigma \in \mathbb{N}$ and Q_σ SLP such that:

$$(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t))' = Q_\sigma(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t), t)$$

Lift: a mapping that associates to an instruction another sequence of instructions

Algorithm:

Our contribution: the algorithm

Assumption: For each instruction $\sigma \in \Sigma_{\text{special}}$, there exist $\mu_\sigma \in \mathbb{N}$ and Q_σ SLP such that:

$$(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t))' = Q_\sigma(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t), t)$$

Lift: a mapping that associates to an instruction another sequence of instructions

Algorithm:

Γ

Input $I_1, \ldots, I_n$
Output $O_1, \ldots, O_n$

for $i = 1, \ldots, |\Gamma|$:
$X_i := \text{op}_i(X_{i,1}, \ldots, X_{i,|\text{op}_i|})$

Our contribution: the algorithm

Assumption: For each instruction $\sigma \in \Sigma_{\text{special}}$, there exist $\mu_\sigma \in \mathbb{N}$ and Q_σ SLP such that:

$$(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t))' = Q_\sigma(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t), t)$$

Lift: a mapping that associates to an instruction another sequence of instructions

Algorithm:

Γ	Π
Input $I_1, \ldots, I_n$	Input $i_1, \bar{i_1}, \ldots, i_n, \bar{i_n}, i_{n+1}, \ldots, i_{n+\theta}$, $\theta := \sum_i \mu_{\text{op}_i}$
Output $O_1, \ldots, O_n$	Ouptut $o_1, \bar{o_1}, \ldots, o_n, \bar{o_n}, o_{n+1}, \ldots, o_{n+\theta}$
for $i = 1, \ldots, \lvert\Gamma\rvert$:	Assume all $x_{i,j}$ and $\bar{x_{i,j}}$ were computed
$X_i := \text{op}_i(X_{i,1}, \ldots, X_{i,\lvert\text{op}_i\rvert})$	Lift($X_i := \text{op}_i(X_{i,1}, \ldots, X_{i,\lvert\text{op}_i\rvert})$)
	ensures $x_i, \bar{x}_i$ are computed.

Our contribution: the algorithm

Assumption: For each instruction $\sigma \in \Sigma_{\text{special}}$, there exist $\mu_\sigma \in \mathbb{N}$ and Q_σ SLP such that:

$$(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t))' = Q_\sigma(\sigma(t), s_2(t), \ldots, s_{\mu_\sigma}(t), t)$$

Lift: a mapping that associates to an instruction another sequence of instructions

Algorithm:

Γ	Π	Φ
Input $I_1, \ldots, I_n$ Output $O_1, \ldots, O_n$	Input $i_1, \bar{i_1}, \ldots, i_n, \bar{i_n}, i_{n+1}, \ldots, i_{n+\theta}$, $\theta := \sum_i \mu_{\text{op}_i}$ Ouptut $o_1, \bar{o_1}, \ldots, o_n, \bar{o_n}, o_{n+1}, \ldots, o_{n+\theta}$	Same as Γ input Same as Π input
for $i = 1, \ldots, \lvert\Gamma\rvert$: $X_i := \text{op}_i(X_{i,1}, \ldots, X_{i,\lvert\text{op}_i\rvert})$	Assume all $x_{i,j}$ and $\overline{x_{i,j}}$ were computed $\text{Lift}(X_i := \text{op}_i(X_{i,1}, \ldots, X_{i,\lvert\text{op}_i\rvert}))$ ensures $x_i, \bar{x}_i$ are computed.	Copy Γ instruction, if special, division or inversion, output value.

Running example

SLP Γ
Input I_1, I_2
Output O_1, O_2

$X_1 := I_1 / I_2$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$O_2 := \cos(I_1)$

Running example

SLP Γ
Input I_1, I_2
Output O_1, O_2
$X_1 := I_1 / I_2$

SLP Π
Input $i_1, \overline{i_1}, i_2, \overline{i_2}, i_3, i_4, i_5$
Output $o_1, \overline{o_1}, o_2, \overline{o_2}, o_3, o_4, o_5$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$O_2 := \cos(I_1)$

Running example

SLP Γ
Input I_1, I_2
Output O_1, O_2
$X_1 := I_1 / I_2$

SLP Π
Input $i_1, \overline{i_1}, i_2, \overline{i_2}, i_3, i_4, i_5$
Output $o_1, \overline{o_1}, o_2, \overline{o_2}, o_3, o_4, o_5$

SLP Φ
Input I_1, I_2
Output $I_1, O_1, I_2, O_2, i_3, i_4, i_5$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$O_2 := \cos(I_1)$

Running example

SLP Γ
Input I_1, I_2
Output O_1, O_2

$X_1 := I_1 / I_2$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$O_2 := \cos(I_1)$

SLP Π
Input $i_1, \overline{i_1}, i_2, \overline{i_2}, i_3, i_4, i_5$
Output $o_1, \overline{o_1}, o_2, \overline{o_2}, o_3, o_4, o_5$

$x_1 := i_1 \times i_3 \quad (= i_1 / i_2)$
$\overline{x_1} := \overline{i_1} \times i_3 \quad (= \overline{i_1} / i_2)$
$o_3 := i_3 \times \overline{i_2} \quad (= \overline{i_2} / i_2)$
$o_3 := \mathrm{nmul}(i_3, o_3) \quad (= -\overline{i_2} / i_2^2)$
$\overline{x_1} := \mathrm{fma}(i_1, o_3, \overline{x_1}) \quad (= \overline{i_1} / i_2 - i_1 \overline{i_2} / i_2^2)$

SLP Φ
Input I_1, I_2
Output $I_1, O_1, I_2, O_2, i_3, i_4, i_5$

Running example

SLP Γ
Input I_1, I_2
Output O_1, O_2

$X_1 := I_1 / I_2$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$O_2 := \cos(I_1)$

SLP Π
Input $i_1, \overline{i_1}, i_2, \overline{i_2}, i_3, i_4, i_5$
Output $o_1, \overline{o_1}, o_2, \overline{o_2}, o_3, o_4, o_5$

$x_1 := i_1 \times i_3 \quad (= i_1 / i_2)$
$\overline{x_1} := \overline{i_1} \times i_3 \quad (= \overline{i_1} / i_2)$
$o_3 := i_3 \times \overline{i_2} \quad (= \overline{i_2} / i_2)$
$o_3 := \mathrm{nmul}(i_3, o_3) \quad (= -\overline{i_2} / i_2^2)$
$\overline{x_1} := \mathrm{fma}(i_1, o_3, \overline{x_1}) \quad (= \overline{i_1} / i_2 - i_1 \overline{i_2} / i_2^2)$

SLP Φ
Input I_1, I_2
Output $I_1, O_1, I_2, O_2, i_3, i_4, i_5$

$i_3 := 1 / i_2$
$X_1 := i_1 \times i_3$

Running example

SLP Γ
Input I_1, I_2
Output O_1, O_2

$X_1 := I_1 / I_2$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$O_2 := \cos(I_1)$

SLP Π
Input $i_1, \overline{i_1}, i_2, \overline{i_2}, i_3, i_4, i_5$
Output $o_1, \overline{o_1}, o_2, \overline{o_2}, o_3, o_4, o_5$

$x_1 := i_1 \times i_3 \quad (= i_1 / i_2)$
$\overline{x_1} := \overline{i_1} \times i_3 \quad (= \overline{i_1} / i_2)$
$o_3 := i_3 \times \overline{i_2} \quad (= \overline{i_2} / i_2)$
$o_3 := \mathrm{nmul}(i_3, o_3) \quad (= -\overline{i_2} / i_2^2)$
$\overline{x_1} := \mathrm{fma}(i_1, o_3, \overline{x_1}) \quad (= \overline{i_1} / i_2 - i_1 \overline{i_2} / i_2^2)$

$o_1 := \mathrm{fma}(5, x_1, i_2) \quad (= 5 x_1 + i_2)$
$\overline{o_1} := \mathrm{fma}(5, \overline{x_1}, \overline{i_2}) \quad (= 5 \overline{x_1} + \overline{i_2})$

SLP Φ
Input I_1, I_2
Output $I_1, O_1, I_2, O_2, i_3, i_4, i_5$

$i_3 := 1 / i_2$
$X_1 := i_1 \times i_3$

Running example

SLP Γ
Input I_1, I_2
Output O_1, O_2

$X_1 := I_1 / I_2$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$O_2 := \cos(I_1)$

SLP Π
Input $i_1, \overline{i_1}, i_2, \overline{i_2}, i_3, i_4, i_5$
Output $o_1, \overline{o_1}, o_2, \overline{o_2}, o_3, o_4, o_5$

$x_1 := i_1 \times i_3 \quad (= i_1 / i_2)$
$\overline{x_1} := \overline{i_1} \times i_3 \quad (= \overline{i_1} / i_2)$
$o_3 := i_3 \times \overline{i_2} \quad (= \overline{i_2} / i_2)$
$o_3 := \mathrm{nmul}(i_3, o_3) \quad (= -\overline{i_2} / i_2^2)$
$\overline{x_1} := \mathrm{fma}(i_1, o_3, \overline{x_1}) \quad (= \overline{i_1} / i_2 - i_1 \overline{i_2} / i_2^2)$

$o_1 := \mathrm{fma}(5, x_1, i_2) \quad (= 5 x_1 + i_2)$
$\overline{o_1} := \mathrm{fma}(5, \overline{x_1}, \overline{i_2}) \quad (= 5 \overline{x_1} + \overline{i_2})$

SLP Φ
Input I_1, I_2
Output $I_1, O_1, I_2, O_2, i_3, i_4, i_5$

$i_3 := 1 / i_2$
$X_1 := i_1 \times i_3$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

Running example

SLP Γ
Input I_1, I_2
Output O_1, O_2

$X_1 := I_1 / I_2$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$O_2 := \cos(I_1)$

SLP Π
Input $i_1, \overline{i_1}, i_2, \overline{i_2}, i_3, i_4, i_5$
Output $o_1, \overline{o_1}, o_2, \overline{o_2}, o_3, o_4, o_5$

$x_1 := i_1 \times i_3 \quad (= i_1 / i_2)$
$\overline{x_1} := \overline{i_1} \times i_3 \quad (= \overline{i_1} / i_2)$
$o_3 := i_3 \times \overline{i_2} \quad (= \overline{i_2} / i_2)$
$o_3 := \mathrm{nmul}(i_3, o_3) \quad (= -\overline{i_2} / i_2^2)$
$\overline{x_1} := \mathrm{fma}(i_1, o_3, \overline{x_1}) \quad (= \overline{i_1} / i_2 - i_1 \overline{i_2} / i_2^2)$

$o_1 := \mathrm{fma}(5, x_1, i_2) \quad (= 5 x_1 + i_2)$
$\overline{o_1} := \mathrm{fma}(5, \overline{x_1}, \overline{i_2}) \quad (= 5 \overline{x_1} + \overline{i_2})$

$o_2 := i_4 \quad (= \cos(i_1))$
$o_5 := \overline{i_1} \times i_4 \quad (= \overline{i_1} \cos(i_1))$
$o_4 := \mathrm{nmul}(\overline{i_1}, i_5) \quad (= -\overline{i_1} \sin(i_1))$
$\overline{o_2} := o_4 \quad (= \cos(i_1))$

SLP Φ
Input I_1, I_2
Output $I_1, O_1, I_2, O_2, i_3, i_4, i_5$

$i_3 := 1 / i_2$
$X_1 := i_1 \times i_3$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

Running example

SLP Γ
Input I_1, I_2
Output O_1, O_2

$X_1 := I_1 / I_2$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$O_2 := \cos(I_1)$

SLP Π
Input $i_1, \overline{i_1}, i_2, \overline{i_2}, i_3, i_4, i_5$
Output $o_1, \overline{o_1}, o_2, \overline{o_2}, o_3, o_4, o_5$

$x_1 := i_1 \times i_3 \quad (= i_1 / i_2)$
$\overline{x_1} := \overline{i_1} \times i_3 \quad (= \overline{i_1} / i_2)$
$o_3 := i_3 \times \overline{i_2} \quad (= \overline{i_2} / i_2)$
$o_3 := \mathrm{nmul}(i_3, o_3) \quad (= -\overline{i_2} / i_2^2)$
$\overline{x_1} := \mathrm{fma}(i_1, o_3, \overline{x_1}) \quad (= \overline{i_1} / i_2 - i_1 \overline{i_2} / i_2^2)$

$o_1 := \mathrm{fma}(5, x_1, i_2) \quad (= 5 x_1 + i_2)$
$\overline{o_1} := \mathrm{fma}(5, \overline{x_1}, \overline{i_2}) \quad (= 5 \overline{x_1} + \overline{i_2})$

$o_2 := i_4 \quad (= \cos(i_1))$
$o_5 := \overline{i_1} \times i_4 \quad (= \overline{i_1} \cos(i_1))$
$o_4 := \mathrm{nmul}(\overline{i_1}, i_5) \quad (= -\overline{i_1} \sin(i_1))$
$\overline{o_2} := o_4 \quad (= \cos(i_1))$

SLP Φ
Input I_1, I_2
Output $I_1, O_1, I_2, O_2, i_3, i_4, i_5$

$i_3 := 1 / i_2$
$X_1 := i_1 \times i_3$

$O_1 := \mathrm{fma}(5, X_1, I_2)$

$i_4 := \cos(i_1)$
$O_2 := I_4$
$i_5 := \sin(i_1)$

Our contribution: theoritical result

Instruction set	Lift number of instructions
Σ_{ring}	3
Σ_{div}	5
Σ_{special}	$\lvert Q_\sigma \rvert + \mu_\sigma + 2$

Our contribution: theoritical result

Instruction set	**Lift number of instructions**
Σ_{ring}	3
Σ_{div}	5
Σ_{special}	$\lvert Q_\sigma\rvert + \mu_\sigma + 2$

Theorem

Let $\Gamma\colon \mathbb{K}^n \to \mathbb{K}^n$ be an SLP over Σ. Then we compute both Π, Φ SLPs in time $O(|\Gamma| + n)$ such that:

$$\begin{cases} y' = \Gamma(y) \\ y(0) = y_0 \end{cases} \iff \begin{cases} (y,w)' = \Pi(y,w) \\ (y,w)(0) = \Phi(y_0) \end{cases}$$

Moreover Π is polynomial and:

$$|\Pi| \leqslant 3|\Gamma|_{\text{ring}} + 5\,|\Gamma|_{\text{div}} + \sum_{1 \leqslant k \leqslant l} (|Q_{\text{op}_k}| + \mu_{\text{op}_k} + 2)$$

Optimizations

Simplify Π:

- Γ represents a polynomial ODE, $\bar{i}$ are output of Γ.
- Apply usual SLP simplification techniques: common subexpression elimination, use fma when possible

Optimizations

Simplify Π:

- Γ represents a polynomial ODE, $\bar{i}$ are output of Γ.
- Apply usual SLP simplification techniques: common subexpression elimination, use fma when possible

For the running example we obtain the final SLP:

$$
\begin{array}{l}
\Pi \\
\text{Input} \quad i_1, i_2, i_3, i_4, i_5 \\
\text{Output} \quad o_1, o_2, o_3, o_4, o_5
\end{array}
$$

$$
\begin{array}{llll}
\Pi_1 \equiv x_1 := i_1 \times i_3 & & \Pi_4 \equiv o_1 := \mathrm{fma}(5, x_1, i_2) & \\
\Pi_2 \equiv o_3 := \mathrm{nmul}(i_3, i_3) & & \Pi_5 \equiv o_4 := \mathrm{nmul}(o_1, i_5) & \\
\Pi_3 \equiv o_3 := o_3 \times i_4 & & \Pi_6 \equiv o_5 := o_1 \times i_4 & \text{Cost} \approx 3 \text{ cycles}
\end{array}
$$

Our contribution: Practical result

Implementation within the JustInLine (JIL) open-source C++ library:
git clone https://git.renater.fr/anonscm/git/jil/jil.git

```cpp
#include <jil/ode/slp_ode_polynomialize.hpp>
#include <jil/slp/slp_recorder.hpp>

using namespace jil;

template <typename T>
void ode1 (T* out, const T* in) {
  out[0] = 5 * in[0] / in[1] + in[1];
  out[1] = cos (in[0]);
}

int
main (int argc, char** argv) {
  domain tp = r32_domain;
  slp q = as_slp (2, 2, tp, ode1);
  jout << "Original SLP = " << q << "\n";
  slp p = ode_polynomialize (q);
  jout << "Polynomialization = " << p << "\n";
}
```

Our contribution: Practical result

Implementation within the JustInLine (JIL) open-source C++ library:
git clone https://git.renater.fr/anonscm/git/jil/jil.git

```cpp
#include <jil/ode/slp_ode_polynomialize.hpp>
#include <jil/slp/slp_recorder.hpp>

using namespace jil;

template <typename T>
void ode1 (T* out, const T* in) {
  out[0] = 5 * in[0] / in[1] + in[1];
  out[1] = cos (in[0]);
}

int
main (int argc, char** argv) {
  domain tp = r32_domain;
  slp q = as_slp (2, 2, tp, ode1);
  jout << "Original SLP = " << q << "\n";
  slp p = ode_polynomialize (q);
  jout << "Polynomialization = " << p << "\n";
}
```

```
Original SLP = input    0, 0
input   1, 1
mul     2, 0, 5
div     3, 2, 1
add     4, 1, 3
cos     2, 0
output  0, 4
output  1, 2
data    5, 5.000000e+00

Polynomialization = input       0, 0
input   1, 1
input   2, 2
input   3, 3
input   4, 4
mul     5, 0, 10
fma     6, 2, 5, 1
mul     7, 2, 3
nmul    7, 2, 7
nmul    8, 4, 6
mul     9, 6, 8
output  0, 6
output  1, 3
output  2, 7
output  3, 8
output  4, 9
data    10, 5.000000e+00
```

Our contribution: Practical result

Implementation within the JustInLine (JIL) open-source C++ library:
git clone https://git.renater.fr/anonscm/git/jil/jil.git

```cpp
#include <jil/ode/slp_ode_polynomialize.hpp>
#include <jil/slp/slp_recorder.hpp>

using namespace jil;

template <typename T>
void ode1 (T* out, const T* in) {
  out[0] = 5 * in[0] / in[1] + in[1];
  out[1] = cos (in[0]);
}

int
main (int argc, char** argv) {
  domain tp = r32_domain;
  slp q = as_slp (2, 2, tp, ode1);
  jout << "Original SLP = " << q << "\n";
  slp p = ode_polynomialize (q);
  jout << "Polynomialization = " << p << "\n";
}
```

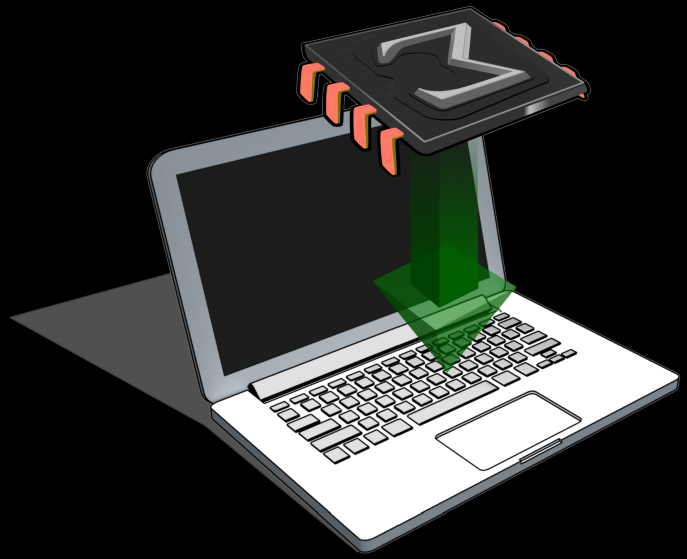
```
Original SLP = input    0, 0
input   1, 1
mul     2, 0, 5
div     3, 2, 1
add     4, 1, 3
cos     2, 0
output  0, 4
output  1, 2
data    5, 5.000000e+00

Polynomialization = input       0, 0
input   1, 1
input   2, 2
input   3, 3
input   4, 4
mul     5, 0, 10
fma     6, 2, 5, 1
mul     7, 2, 3
nmul    7, 2, 7
nmul    8, 4, 6
mul     9, 6, 8
output  0, 6
output  1, 3
output  2, 7
output  3, 8
output  4, 9
data    10, 5.000000e+00
```

a gcc -O3 -mavx2 later…

	Pratical costs
Gen	253 cycles
Poly	12 cycles

Thank you!



http://www.TeXmacs.org