Developing Disciplined Programs

Seminar at the James M. Hull College of Business

Clément Aubert

Appalachian State University

Augusta University

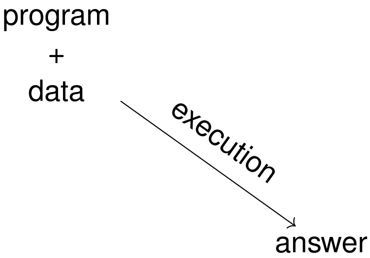
30th January 2017

program

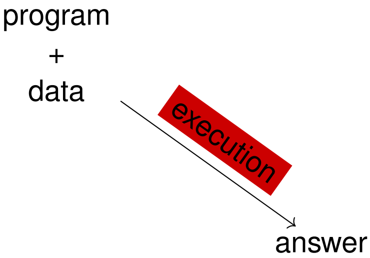
program

+

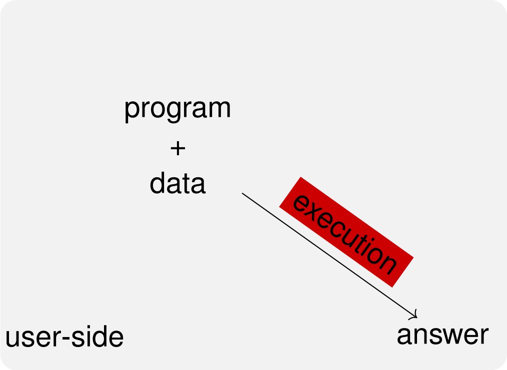
data

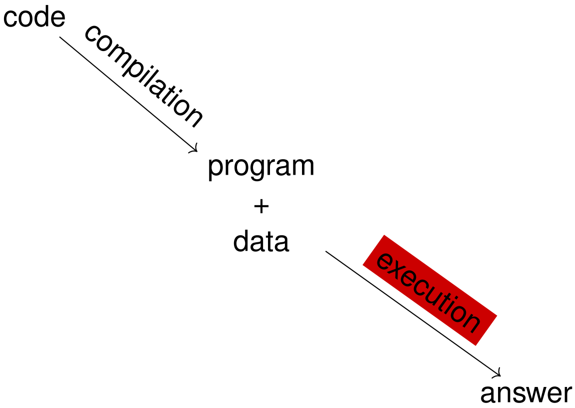

program
+
data
execution
answer

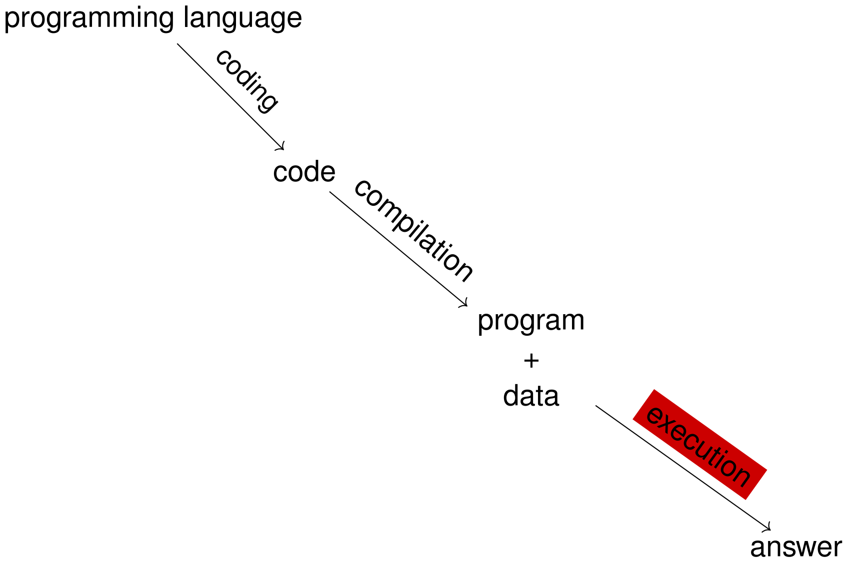
Introduction: What is the problem with my program?

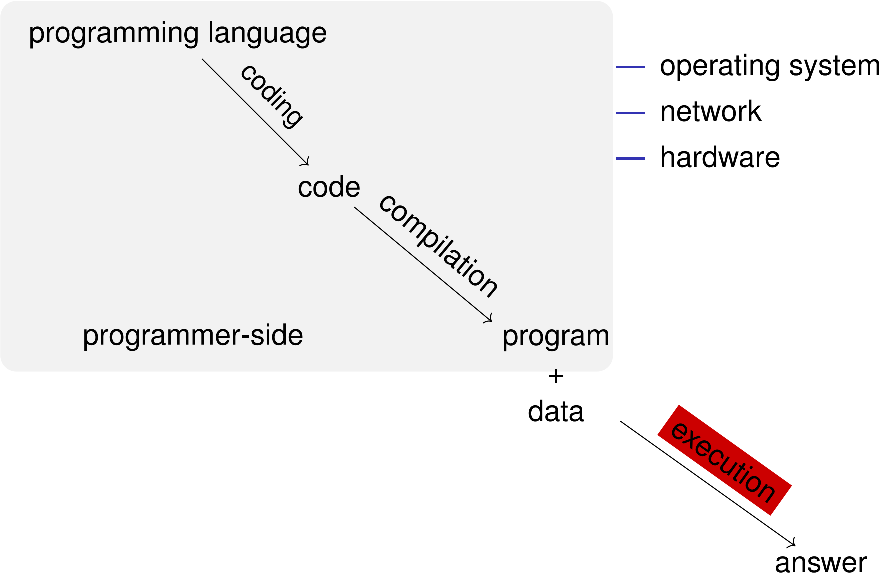


- operating system
- network
- hardware

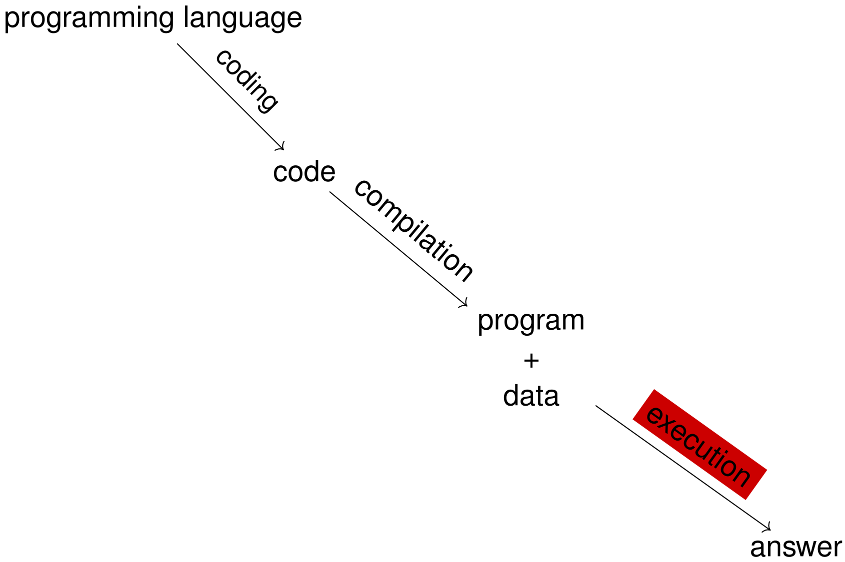



code
compilation
program
+
data
execution
answer


programming language
coding
code
compilation
program
+
data
execution
answer


programming language
coding
code
compilation
programmer-side
program
+
data
execution
answer
operating system
network
hardware

Introduction: What is the problem with my program?



Developing Disciplined Programs

Seminar at the James M. Hull College of Business

Clément Aubert

Appalachian State University

Augusta University

30th January 2017

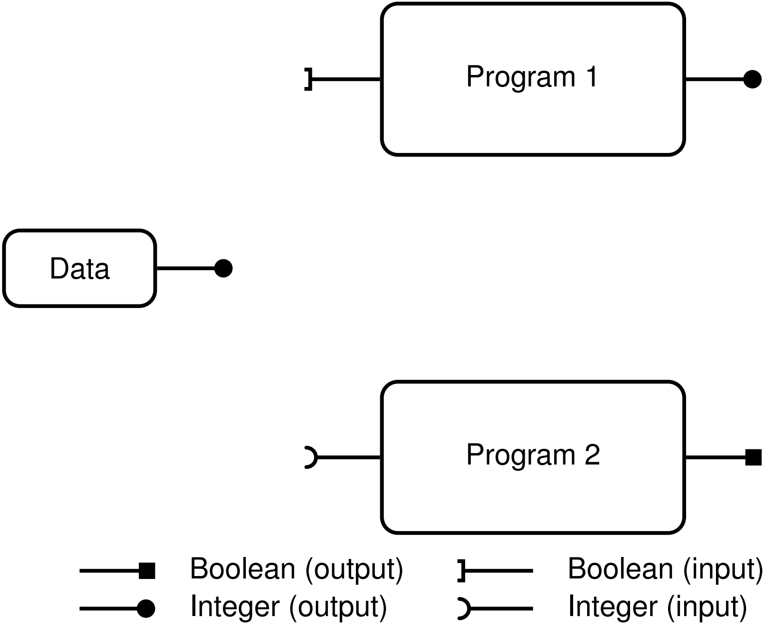
Developing Disciplined *Programing Languages*

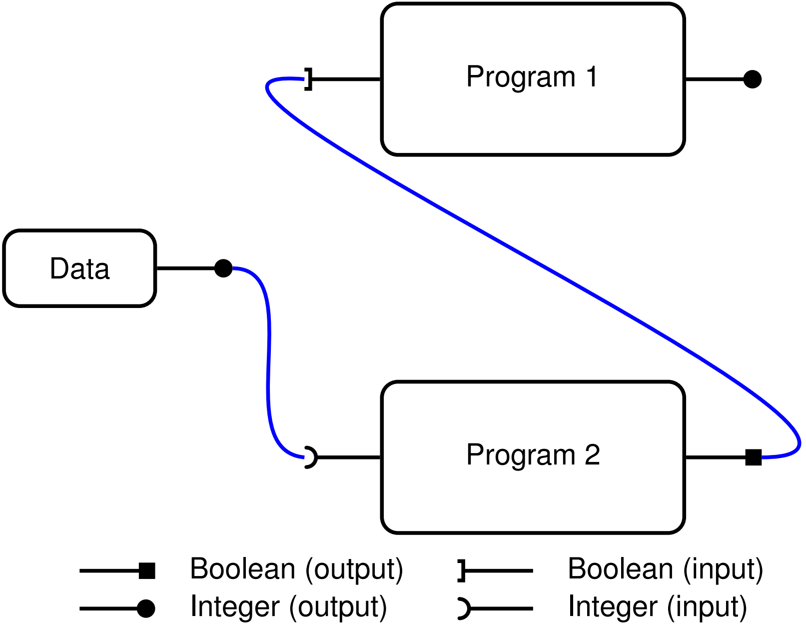
Seminar at the James M. Hull College of Business

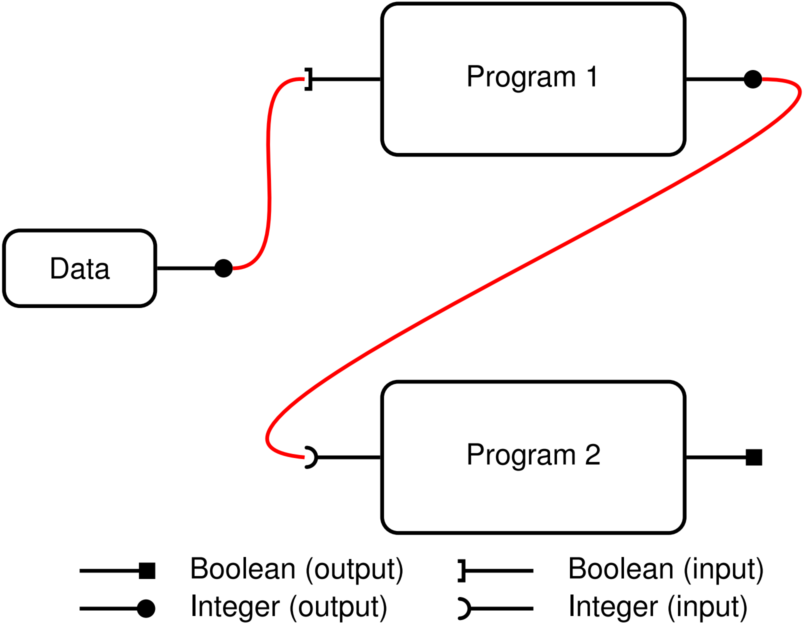
Clément Aubert

Appalachian State University

Augusta University
30th January 2017


Program 1
Data
Program 2
Boolean (output)
Boolean (input)
Integer (output)
Integer (input)


Program 1
Data
Program 2
Boolean (output)
Integer (output)
Boolean (input)
Integer (input)


Program 1
Data
Program 2
Boolean (output)
Integer (output)
Boolean (input)
Integer (input)

$$\dfrac{\vdash \text{Program 1} : \text{Bool} \to \text{Int} \qquad \dfrac{\vdash \text{Program 2} : \text{Int} \to \text{Bool} \qquad \vdash \text{data} : \text{Int}}{\vdash \text{Program 2 (data)} : \text{Bool}}}{\vdash \text{Program1 (Program 2 (data))} : \text{Int}}$$

$$\dfrac{\vdash \text{Program 1 : Bool} \to \text{Int} \qquad \dfrac{\vdash \text{Program 2 : Int} \to \text{Bool} \qquad \vdash \text{data : Int}}{\vdash \text{Program 2 (data) : Bool}}}{\vdash \text{Program1 (Program 2 (data)) : Int}}$$

$$\rightsquigarrow$$

$$\dfrac{\vdash \text{Bool} \to \text{Int} \qquad \dfrac{\vdash \text{Int} \to \text{Bool} \qquad \vdash \text{Int}}{\vdash \text{Bool}}}{\vdash \text{Int}}$$

$$\dfrac{\vdash \text{Program 1} : \text{Bool} \to \text{Int} \qquad \dfrac{\vdash \text{Program 2} : \text{Int} \to \text{Bool} \qquad \vdash \text{data} : \text{Int}}{\vdash \text{Program 2 (data)} : \text{Bool}}}{\vdash \text{Program1 (Program 2 (data))} : \text{Int}}$$

$$\rightsquigarrow$$

$$\dfrac{\vdash \text{Bool} \to \text{Int} \qquad \dfrac{\vdash \text{Int} \to \text{Bool} \qquad \vdash \text{Int}}{\vdash \text{Bool}}}{\vdash \text{Int}}$$

$$\rightsquigarrow$$

$$\dfrac{\vdash_x \text{Bool} \to \text{Int} \qquad \dfrac{\vdash_y \text{Int} \to \text{Bool} \qquad \vdash_z \text{Int}}{\vdash_{y+z+c} \text{Bool}}}{\vdash_{y+z+c+x+c'} \text{Int}}$$

Computational Complexity

— Sort problem by their difficulty

Computational Complexity

- Sort problem by their difficulty
- Order of magnitude

Computational Complexity

- Sort problem by their difficulty
- Order of magnitude
- Benchmark: Turing Machine

Computational Complexity

- Sort problem by their difficulty
- Order of magnitude
- Benchmark: Turing Machine

Complete Problems

Logarithmic Space (**L**): Acyclicity in undirected graph
Non-Deterministic Logarithmic Space (**NL**): Acyclicity in directed graph
Polynomial Time (**Ptime**): Circuit value problem

Explicit Computational Complexity

- Sort problem by their difficulty
- Order of magnitude
- Benchmark: Turing Machine

Complete Problems

Logarithmic Space (**L**): Acyclicity in undirected graph
Non-Deterministic Logarithmic Space (**NL**): Acyclicity in directed graph
Polynomial Time (**Ptime**): Circuit value problem

- Machine-dependent
- “External” clock and “external” measure on the tape

> classes. By implicit, we here mean that classes are not given by constraining the amount of resources a *machine* is allowed to use, but rather by imposing linguistic constraints on the way *algorithms* are formulated. This idea has de-

(Dal Lago, 2011, p. 90)(lacl.fr/~caubert/AU/)

> classes. By implicit, we here mean that classes are not given by constraining the amount of resources a *machine* is allowed to use, but rather by imposing linguistic constraints on the way *algorithms* are formulated. This idea has de-

(Dal Lago, 2011, p. 90)(lacl.fr/~caubert/AU/)

Implicit Computational Complexity (ICC)

- Machine-independent
- Without explicit bounds

> classes. By implicit, we here mean that classes are not given by constraining the amount of resources a *machine* is allowed to use, but rather by imposing linguistic constraints on the way *algorithms* are formulated. This idea has de-

(Dal Lago, 2011, p. 90)(lacl.fr/~caubert/AU/)

Implicit Computational Complexity (ICC)

— Machine-independent

— Without explicit bounds

Some Achievements

— Fine-grained type systems for **Ptime**, **L**, **NL**, **Pspace**, etc.

— Differential privacy (Gaboardi et al., 2013)

— Computation over the reals (Férée et al., 2015)

Machine-dependent

Turing machine,
Random access machine,
Counter machine, . . .

Machine-dependent	**Machine-independent**
Turing machine, Random access machine, Counter machine, . . .	Bounded recursion on notation (Cobham, 1965), Bounded linear logic (Girard et al., 1992), Bounded arithmetic (Buss, 1986), . . .

Machine-dependent	**Machine-independent**
Turing machine, Random access machine, Counter machine, . . .	Bounded recursion on notation (Cobham, 1965), Bounded linear logic (Girard et al., 1992), Bounded arithmetic (Buss, 1986), . . .

The rules for storage naturally induce polynomials:

$$\textit{Storage}\quad \frac{!_{\vec{y}}\Gamma \vdash A}{!_{x\vec{y}}\Gamma \vdash !_x A} \qquad \textit{Weakening}\quad \frac{\Gamma \vdash B}{\Gamma, !_0 A \vdash B}$$

$$\textit{Contraction}\quad \frac{\Gamma, !_x A, !_y A \vdash B}{\Gamma, !_{x+y} A \vdash B} \qquad \textit{Dereliction}\quad \frac{\Gamma, A \vdash B}{\Gamma, !_1 A \vdash B}.$$

(Girard et al., 1992, p. 18)

	Machine-dependent	**Machine-independent**
Explicit bounds	Turing machine, Random access machine, Counter machine, . . .	Bounded recursion on notation (Cobham, 1965), Bounded linear logic (Girard et al., 1992), Bounded arithmetic (Buss, 1986), . . .

	Machine-dependent	**Machine-independent**
Explicit bounds	Turing machine, Random access machine, Counter machine, . . .	Bounded recursion on notation (Cobham, 1965), Bounded linear logic (Girard et al., 1992), Bounded arithmetic (Buss, 1986), . . .
Implicit bounds		Descriptive complexity (Fagin, 1973), Recursion on notation (Bellantoni and Cook, 1992), Tiered recurrence (Leivant, 1993), . . .

	Machine-dependent	**Machine-independent**
Explicit bounds	Turing machine, Random access machine, Counter machine, . . .	Bounded recursion on notation (Cobham, 1965), Bounded linear logic (Girard et al., 1992), Bounded arithmetic (Buss, 1986), . . .
Implicit bounds	Automaton, Auxiliary pushdown machine,. . .	Descriptive complexity (Fagin, 1973), Recursion on notation (Bellantoni and Cook, 1992), Tiered recurrence (Leivant, 1993), . . .

	Machine-dependent	**Machine-independent**
Explicit bounds	Turing machine, Random access machine, Counter machine, ...	Bounded recursion on notation (Cobham, 1965), Bounded linear logic (Girard et al., 1992), Bounded arithmetic (Buss, 1986), ...
...ounds	Automaton, Auxiliary pushdown machine,	Descriptive complexity (Fagin, 1973), Recursion on notation (Bellantoni and Cook, 1992),

related to the foregoing question. More specifically, we have attempted to characterize several tape and time complexity classes of Turing machines in terms of devices whose definitions involve only ways in which their infinite memory may be manipulated and no restrictions are imposed on the amount of memory that they use. The basic model

(Ibarra, 1971, p. 88)

$\mathrm{2NFA(k,p)}$

For $k \geqslant 1$, $p \geqslant 0$, a 2-*way non-deterministic finite automaton with k-heads and p pushdown stacks* is a tuple $M = \{\mathbf{S}, A, B, \triangleright, \triangleleft, \boxdot, \sigma\}$ where:

- $\mathbf{S}$ is the finite set of *states*;
- A is the *input alphabet*, B is the *stack alphabet*;
- $\triangleright$ and $\triangleleft$ are the *left* and *right endmarkers*, $\triangleright, \triangleleft \notin A$;
- $\boxdot$ is the *bottom symbol of the stack*, $\boxdot \notin B$;
- $\sigma \subseteq (\mathbf{S} \times (A \cup \{\triangleright, \triangleleft\})^k \times (B \cup \{\boxdot\})^p) \times (\mathbf{S} \times \{-1, 0, +1\}^k \times \{\mathrm{pop}, \mathrm{peek}, \mathrm{push}(b)\}^p)$

$\mathbf{2NFA}(\mathbf{k}, \mathbf{p}) = \{\mathcal{L}(M) \mid M \text{ a } \mathrm{2NFA(k,p)}\}$
$\mathbf{2NFA}(*, \mathbf{p}) = \cup_{k \geqslant 1} \mathbf{2NFA}(\mathbf{k}, \mathbf{p})$

Main characterizations

Automata	**Language / Predicate**
2NFA$(\mathbf{1}, \mathbf{2})$	Computable
2NFA$(*, \mathbf{1})$	Polynomial time (**Ptime**)
2NFA$(*, \mathbf{0})$	Non-Deterministic Logarithmic space (**NL**)
2NFA$(\mathbf{1}, \mathbf{1})$	Context-free
2NFA$(\mathbf{1}, \mathbf{0})$	Regular

Main characterizations

Automata	**Language / Predicate**
2NFA$(\mathbf{1}, \mathbf{2})$	Computable
2NFA$(*, \mathbf{1})$	Polynomial time (**Ptime**)
2NFA$(*, \mathbf{0})$	Non-Deterministic Logarithmic space (**NL**)
2NFA$(\mathbf{1}, \mathbf{1})$	Context-free
2NFA$(\mathbf{1}, \mathbf{0})$	Regular

Question

Can we use those results to develop disciplined programing languages?

Logic Programming

- A programming paradigm
- Computation = unification
- Turing-complete

Logic Programming

- A programming paradigm
- Computation = unification
- Turing-complete

Used in …

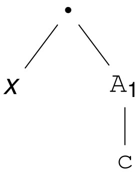
- `Prolog`, `Datalog`
- Type-inference in `Haskell` and `ML`
- Models of Linear Logic (Baillot and Pedicini, 2001; Girard, 2013)

First-order terms

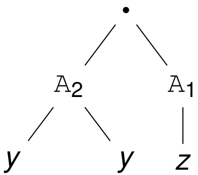
$$
\begin{array}{llll}
t, u & := & \mathtt{c}, \mathtt{d}, \ldots & \in \mathrm{C} \\
 & \mid & x, y, z, \ldots & \in \mathrm{V} \\
 & \mid & \mathtt{A}_n(t_1, \ldots, t_n) & n \in \mathbb{N}^* \\
 & \mid & t \bullet u & \text{with } t \bullet u \bullet v := t \bullet (u \bullet v)
\end{array}
$$

Example

$x \bullet \mathtt{A}_1(\mathtt{c})$



$\mathtt{A}_2(y, y) \bullet \mathtt{A}_1(z)$

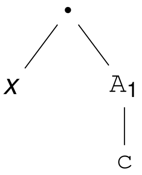


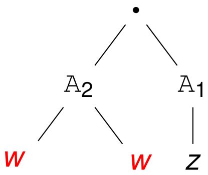
First-order terms

$$
\begin{array}{llll}
t, u & := & \mathtt{c}, \mathtt{d}, \ldots & \in \mathrm{C} \\
 & \mid & x, y, z, \ldots & \in \mathrm{V} \\
 & \mid & \mathtt{A}_n(t_1, \ldots, t_n) & n \in \mathbb{N}^* \\
 & \mid & t \bullet u & \text{with } t \bullet u \bullet v := t \bullet (u \bullet v)
\end{array}
$$

Example

$x \bullet \mathtt{A}_1(\mathtt{c})$ $\mathtt{A}_2(w, w) \bullet \mathtt{A}_1(z)$



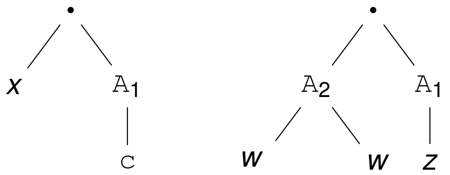


First-order terms

$$
\begin{array}{llll}
t, u & := & \mathtt{c}, \mathtt{d}, \ldots & \in \mathrm{C} \\
 & \mid & x, y, z, \ldots & \in \mathrm{V} \\
 & \mid & \mathtt{A}_n(t_1, \ldots, t_n) & n \in \mathbb{N}^* \\
 & \mid & t \bullet u & \text{with } t \bullet u \bullet v := t \bullet (u \bullet v)
\end{array}
$$

Example

$x \bullet \mathtt{A}_1(\mathtt{c})$ $\mathtt{A}_2(w, w) \bullet \mathtt{A}_1(z)$ Unifiable?

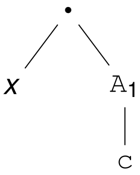


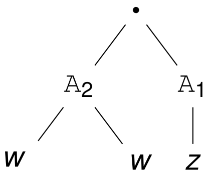
First-order terms

t, u	:=	$\mathtt{c}, \mathtt{d}, \ldots$	$\in \mathrm{C}$
	|	$x, y, z, \ldots$	$\in \mathrm{V}$
	|	$\mathtt{A}_n(t_1, \ldots, t_n)$	$n \in \mathbb{N}^*$
	|	$t \bullet u$	with $t \bullet u \bullet v := t \bullet (u \bullet v)$

Example

$x \bullet \mathtt{A}_1(\mathtt{c})$ $\quad$ $\mathtt{A}_2(w, w) \bullet \mathtt{A}_1(z)$ $\quad$ Unifiable?





✓

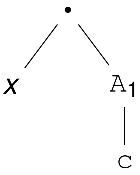
$$\theta = [x \leftarrow \mathtt{A}_2(w, w); z \leftarrow c]$$

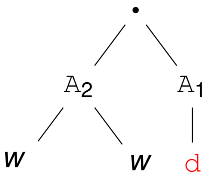
First-order terms

$$
\begin{array}{llll}
t, u & := & \mathtt{c}, \mathtt{d}, \ldots & \in \mathrm{C} \\
 & \mid & x, y, z, \ldots & \in \mathrm{V} \\
 & \mid & \mathtt{A}_n(t_1, \ldots, t_n) & n \in \mathbb{N}^* \\
 & \mid & t \bullet u & \text{with } t \bullet u \bullet v := t \bullet (u \bullet v)
\end{array}
$$

Example

$x \bullet \mathtt{A}_1(\mathtt{c})$ $\quad$ $\mathtt{A}_2(w, w) \bullet \mathtt{A}_1(\mathtt{d})$ $\quad$ Unifiable?





✗

$\mathtt{c} \neq \mathtt{d}$

Flows and Wirings

A *flow* is a pair of terms $t \leftharpoondown u$ with $\mathtt{Var}(t) \subseteq \mathtt{Var}(u)$.
A *wiring* is a finite set of flows.

Flows and Wirings

A *flow* is a pair of terms $t \leftharpoonup u$ with $\texttt{Var}(t) \subseteq \texttt{Var}(u)$.
A *wiring* is a finite set of flows.

Composition of Flows

Let $u \leftharpoonup v$ and $t \leftharpoonup w$ be two flows, $\texttt{Var}(v) \cap \texttt{Var}(w) = \varnothing$,

$$(u \leftharpoonup v)(t \leftharpoonup w) := \begin{cases} u\theta \leftharpoonup w\theta & \text{if } v\theta = t\theta \\ \text{undefined} & \text{otherwise} \end{cases}$$

Flows and Wirings

A *flow* is a pair of terms $t \leftharpoonup u$ with $\texttt{Var}(t) \subseteq \texttt{Var}(u)$.
A *wiring* is a finite set of flows.

Composition of Flows

Let $u \leftharpoonup v$ and $t \leftharpoonup w$ be two flows, $\texttt{Var}(v) \cap \texttt{Var}(w) = \emptyset$,

$$(u \leftharpoonup v)(t \leftharpoonup w) := \begin{cases} u\theta \leftharpoonup w\theta & \text{if } v\theta = t\theta \\ \text{undefined} & \text{otherwise} \end{cases}$$

Examples

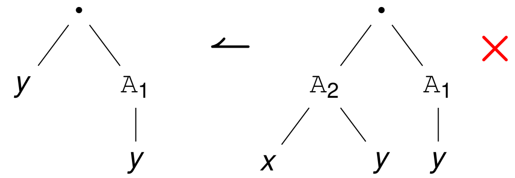
$$(\texttt{f}(x) \leftharpoonup x)(\texttt{f}(y) \leftharpoonup \texttt{g}(y)) = \texttt{f}(\texttt{f}(y)) \leftharpoonup \texttt{g}(y)$$

$$(x \bullet \texttt{c} \leftharpoonup (y \bullet y) \bullet x)((\texttt{c} \bullet \texttt{c}) \bullet x \leftharpoonup y \bullet x) = x \bullet \texttt{c} \leftharpoonup \texttt{c} \bullet \texttt{x}$$

Balanced

A flow $f = t \leftharpoondown u$ is *balanced* if for any $x \in \mathtt{Var}(t) \cup \mathtt{Var}(u)$, all occurrences of x in both t and u have the same height.

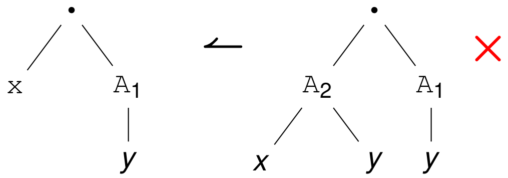
Examples



Balanced

A flow $f = t \leftharpoonup u$ is *balanced* if for any $x \in \mathtt{Var}(t) \cup \mathtt{Var}(u)$, all occurrences of x in both t and u have the same height.

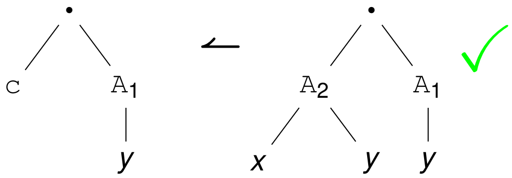
Examples



Balanced

A flow $f = t \leftharpoonup u$ is *balanced* if for any $x \in \mathtt{Var}(t) \cup \mathtt{Var}(u)$, all occurrences of x in both t and u have the same height.

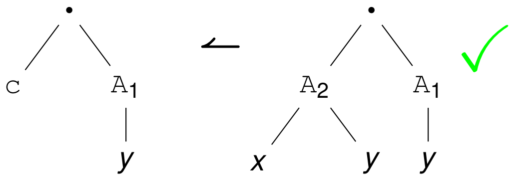
Examples



Balanced

A flow $f = t \leftharpoonup u$ is *balanced* if for any $x \in \mathtt{Var}(t) \cup \mathtt{Var}(u)$, all occurrences of x in both t and u have the same height.

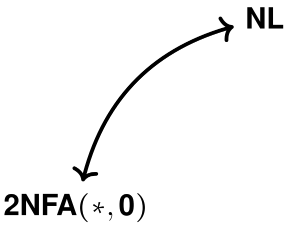
Examples



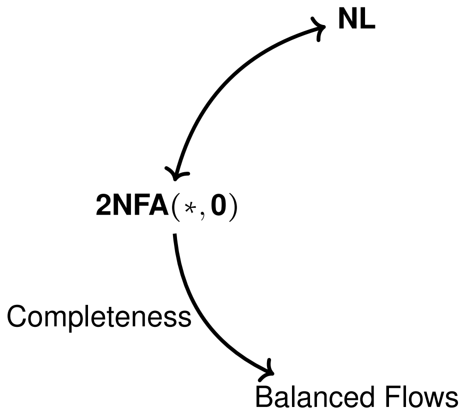
Unary

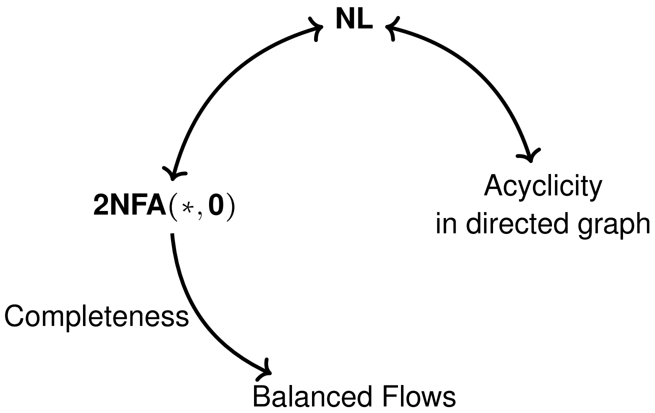
A flow is *unary* if it is built using only unary function symbols and a variable.

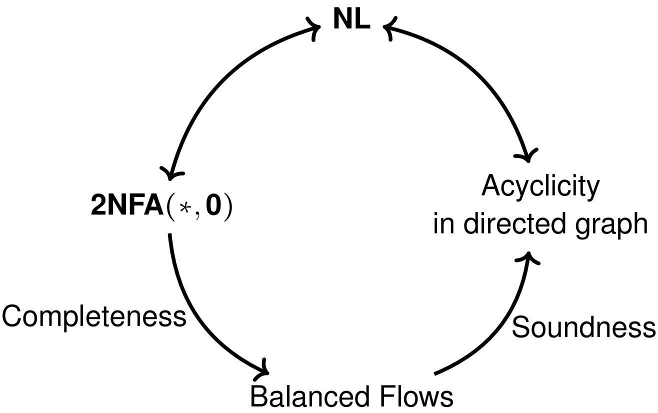
1. Introduction
2. Automata and ICC
3. Logic Programming
4. A New Correspondence
   - New Results
   - New Connexions
5. Perspectives

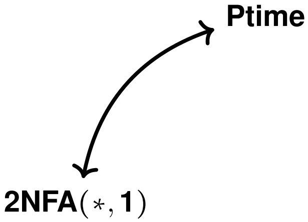


Balanced Flows

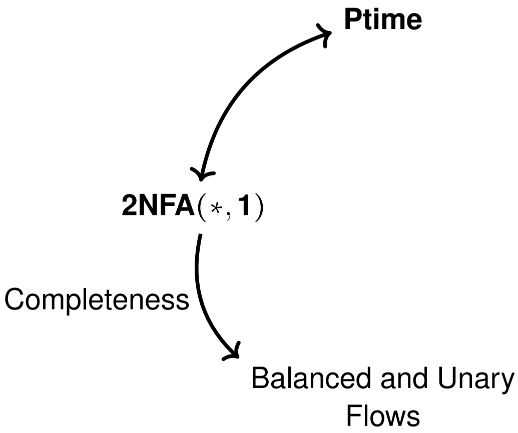

NL
2NFA$(*, \mathbf{0})$
Completeness
Balanced Flows

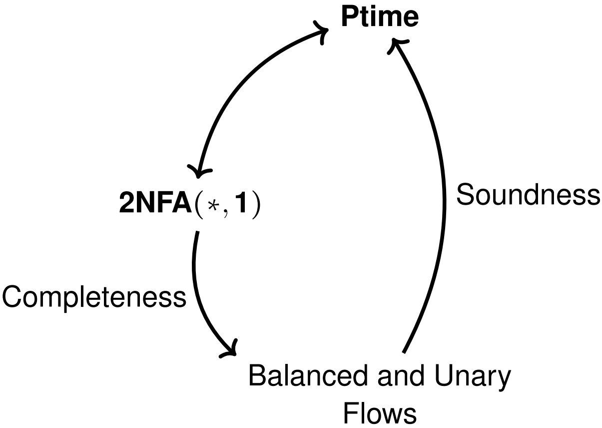

NL
2NFA$(*, \mathbf{0})$
Acyclicity
in directed graph
Completeness
Balanced Flows

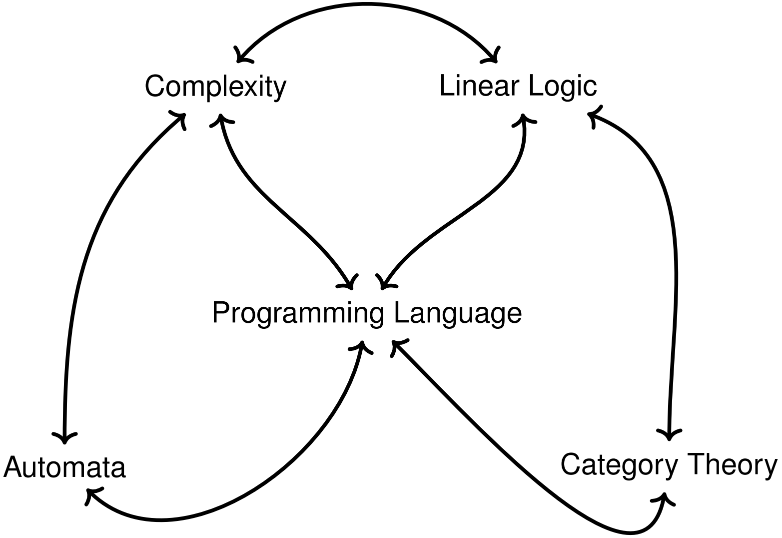

NL
2NFA(∗, 0)
Acyclicity
in directed graph
Completeness
Soundness
Balanced Flows



Balanced and Unary Flows


Ptime
2NFA$(*, \mathbf{1})$
Completeness
Balanced and Unary
Flows


Ptime
2NFA$(*, \mathbf{1})$
Soundness
Completeness
Balanced and Unary Flows


Complexity
Linear Logic
Programming Language
Automata
Category Theory

Results of a series of works (Aubert, 2015; Aubert and Bagnol, 2014; Aubert, Bagnol, and Seiller, 2016; Aubert and Seiller, 2016a,b; Aubert et al., 2014) whose story remains to be told.

— From Proof Theory to simulations

Results of a series of works (Aubert, 2015; Aubert and Bagnol, 2014; Aubert, Bagnol, and Seiller, 2016; Aubert and Seiller, 2016a,b; Aubert et al., 2014) whose story remains to be told.

- From Proof Theory to simulations
- Algebraic techniques

Results of a series of works (Aubert, 2015; Aubert and Bagnol, 2014; Aubert, Bagnol, and Seiller, 2016; Aubert and Seiller, 2016a,b; Aubert et al., 2014) whose story remains to be told.

- From Proof Theory to simulations
- Algebraic techniques
- Pushdown Systems (PDS)?

Results of a series of works (Aubert, 2015; Aubert and Bagnol, 2014; Aubert, Bagnol, and Seiller, 2016; Aubert and Seiller, 2016a,b; Aubert et al., 2014) whose story remains to be told.

— From Proof Theory to simulations
— Algebraic techniques
— Pushdown Systems (PDS)?
— Functional complexity?

In increasing order of complexity:

— Write an intepreter for Automata (Chakraborty, Saxena, and Katti, 2011)

In increasing order of complexity:

- Write an intepreter for Automata (Chakraborty, Saxena, and Katti, 2011)
- The odd status of input in logic programming: can we have non-deterministic data?

In increasing order of complexity:

- Write an intepreter for Automata (Chakraborty, Saxena, and Katti, 2011)
- The odd status of input in logic programming: can we have non-deterministic data?
- Transfer results from automata to logic programming!

In increasing order of complexity:

- Write an intepreter for Automata (Chakraborty, Saxena, and Katti, 2011)
- The odd status of input in logic programming: can we have non-deterministic data?
- Transfer results from automata to logic programming!
- Encode other variations of automata

In increasing order of complexity:

- Write an intepreter for Automata (Chakraborty, Saxena, and Katti, 2011)
- The odd status of input in logic programming: can we have non-deterministic data?
- Transfer results from automata to logic programming!
- Encode other variations of automata
- Go back to the type system

In increasing order of complexity:

- Write an intepreter for Automata (Chakraborty, Saxena, and Katti, 2011)
- The odd status of input in logic programming: can we have non-deterministic data?
- Transfer results from automata to logic programming!
- Encode other variations of automata
- Go back to the type system

Thanks!

Aubert, Clément (2015). *An in-between "implicit" and "explicit" complexity: Automata*. Research Report. 5 pp. arXiv: `1502.00145 [cs.LO]`. Communication at DICE 2015.

Aubert, Clément and Marc Bagnol (2014). "Unification and Logarithmic Space". In: *RTA-TLCA*. Ed. by Gilles Dowek. Vol. 8650. LNCS. Springer, pp. 77–92. arXiv: `1402.4327 [cs.LO]`.

Aubert, Clément, Marc Bagnol, and Thomas Seiller (2016). "Unary Resolution: Characterizing Ptime". In: *FoSSaCS*. Ed. by Bart Jacobs and Christof Löding. Vol. 9634. LNCS. Springer, pp. 373–389.

Aubert, Clément and Thomas Seiller (2016a). "Characterizing co-NL by a group action". In: *MSCS* 26 (04), pp. 606–638.

Aubert, Clément and Thomas Seiller (2016b). "Logarithmic Space and Permutations". In: *Inf. Comput.* 248. Ed. by Simona Ronchi Della Rocca and Virgile Mogbil. Development on Implicit Computational Complexity (DICE 2013), pp. 2–21.

Aubert, Clément, Marc Bagnol, Paolo Pistone, and Thomas Seiller (2014). "Logic Programming and Logarithmic Space". In: *APLAS*. Ed. by Jacques Garrigue. Vol. 8858. LNCS. Springer, pp. 39–57. arXiv: `1406.2110 [cs.LO]`.

Baillot, Patrick and Marco Pedicini (2001). "Elementary Complexity and Geometry of Interaction". In: *Fund. Inform.* 45.1–2, pp. 1–31.

Bellantoni, Stephen J. and Stephen Arthur Cook (1992). “A New Recursion-Theoretic Characterization of the Polytime Functions (Extended Abstract)”. In: *STOC*. Ed. by S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis. ACM, pp. 283–93.

Buss, Samuel R. (1986). *Bounded Arithmetic*. Vol. 3. Studies in Proof Theory. Lecture Notes. Bibliopolis.

Chakraborty, Pinaki, Prem Chandra Saxena, and Chittaranjan Padmanabha Katti (2011). “Fifty years of automata simulation: a review”. In: *Inroads* 2.4, pp. 59–70.

- Cobham, Alan (1965). “The intrinsic computational difficulty of functions”. In: *Logic, methodology and philosophy of science: Proceedings of the 1964 international congress held at the Hebrew university of Jerusalem, Israel, from August 26 to September 2, 1964*. Ed. by Yehoshua Bar-Hillel. Studies in Logic and the foundations of mathematics. North-Holland Publishing Company, pp. 24–30.
- Dal Lago, Ugo (2011). “A Short Introduction to Implicit Computational Complexity”. In: *ESSLLI*. Ed. by Nick Bezhanishvili and Valentin Goranko. Vol. 7388. LNCS. Springer, pp. 89–109.
- Fagin, Ronald (1973). “Contributions to the Model Theory of Finite Structures”. PhD thesis. University of California, Berkeley.

- Férée, Hugo, Emmanuel Hainry, Mathieu Hoyrup, and Romain Péchoux (2015). "Characterizing polynomial time complexity of stream programs using interpretations". In: *Theoret. Comput. Sci.* 585, pp. 41–54.
- Gaboardi, Marco, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce (2013). "Linear dependent types for differential privacy". In: *POPL*. Ed. by Roberto Giacobazzi and Radhia Cousot. ACM, pp. 357–370.
- Girard, Jean-Yves (2013). "Three lightings of logic". In: *CSL*. Ed. by Simona Ronchi Della Rocca. Vol. 23. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 11–23.
- Girard, Jean-Yves, Andre Scedrov, and Philip J. Scott (1992). "Bounded linear logic: a modular approach to polynomial-time computability". In: *Theoret. Comput. Sci.* 97.1, pp. 1–66.

- Ibarra, Oscar H. (1971). "Characterizations of Some Tape and Time Complexity Classes of Turing Machines in Terms of Multihead and Auxiliary Stack Automata". In: *J. Comput. Syst. Sci.* 5.2, pp. 88–117.
- Leivant, Daniel (1993). "Stratified Functional Programs and Computational Complexity". In: *POPL*. Ed. by Mary S. Van Deusen and Bernard Lang. ACM Press, pp. 325–333.