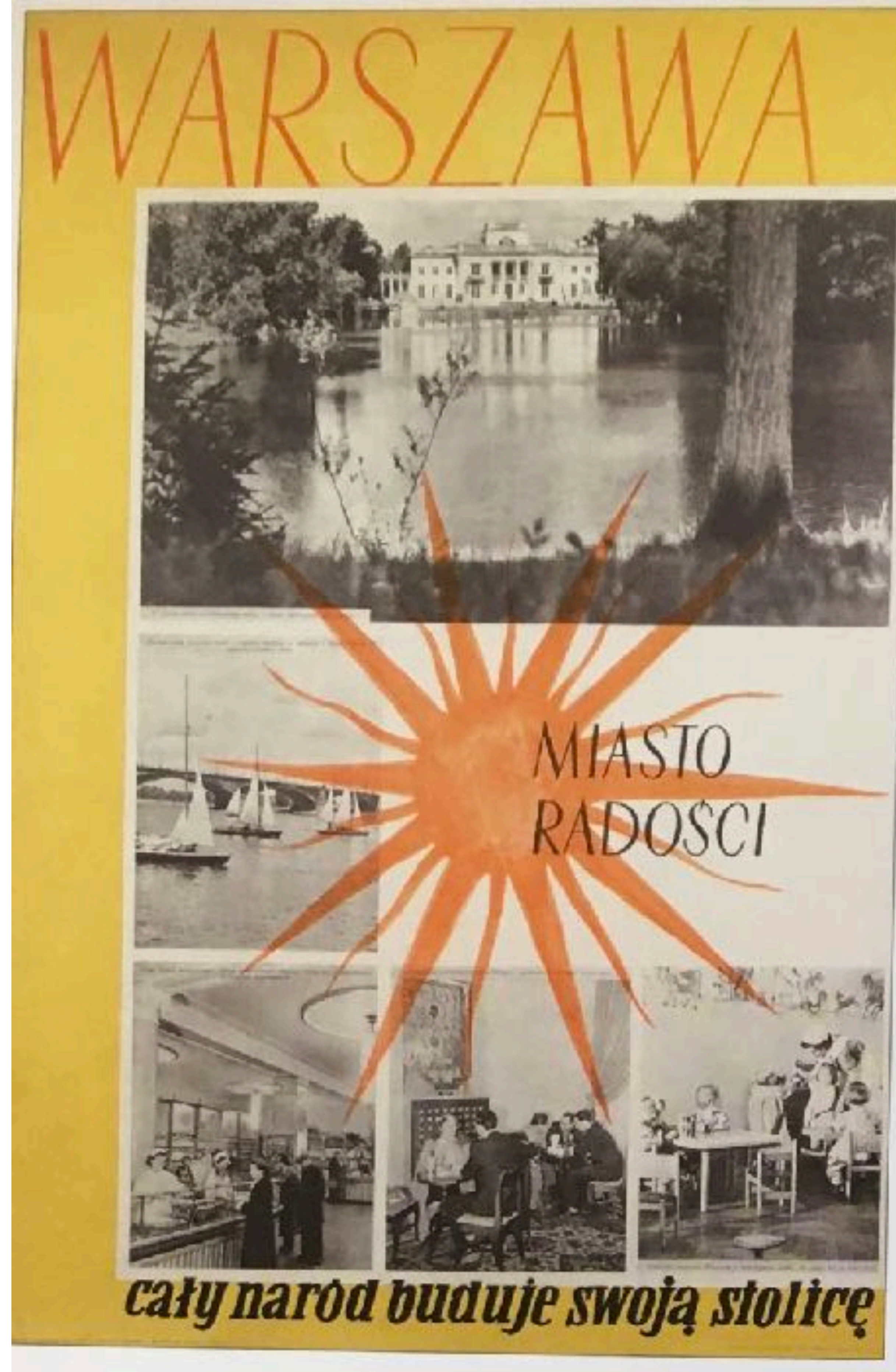
MSOL and higher-order computation

Igor Walukiewicz
CNRS Bordeaux



$$a^n b c^n d \quad + \quad a^w$$

$$a^n b c^n d \quad + \quad a^w$$

$$a^* b c^* d \quad + \quad a^w$$

Where the trees come from?

An undecidable problem:

Given a Turing machine M check if the language accepted by M is empty ($L(M) =^? \emptyset$).

Problem P(φ):

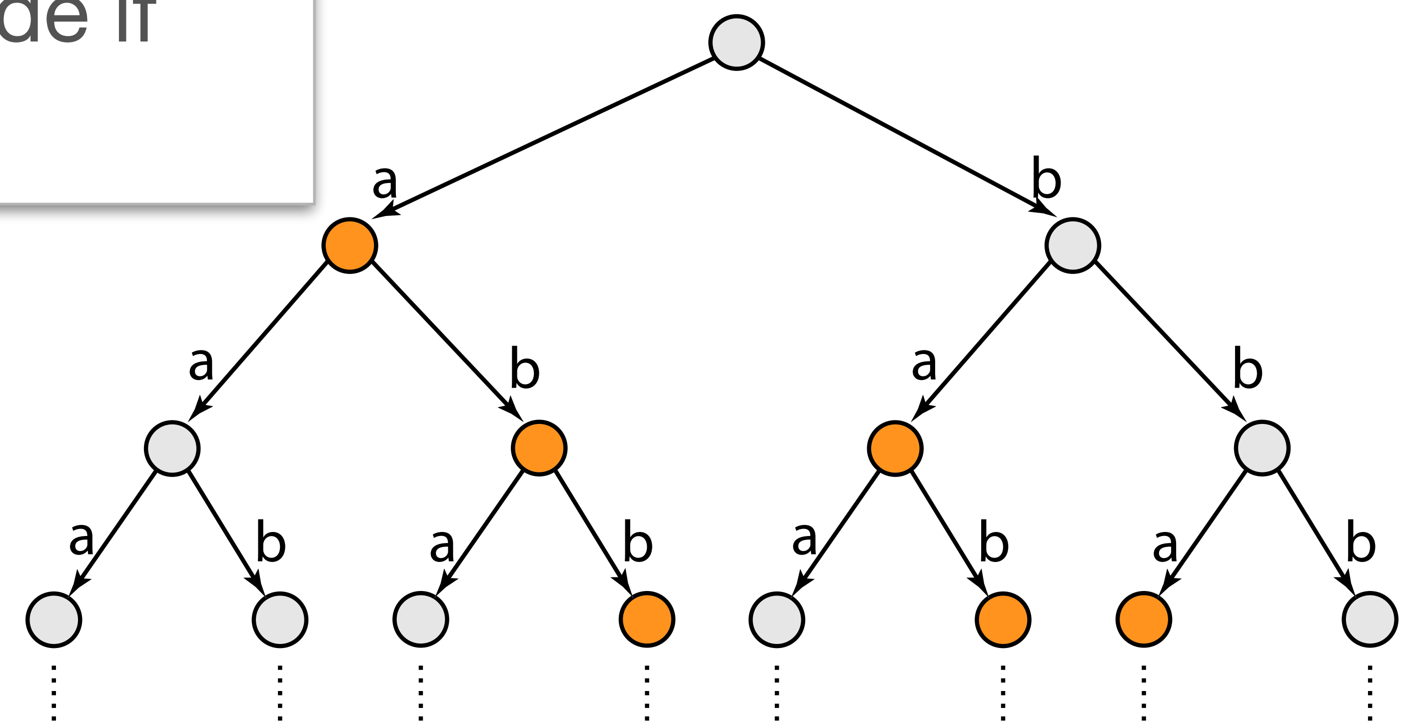
Given a Turing machine M decide if L(M) has the property φ.

Thm [Rice'53]:

For every nontrivial property φ, the problem P(φ) is undecidable.

The case of finite automata

Problem P'(φ):
Given a **finite automaton** A decide if L(A) has the property φ.



If Σ the alphabet of A, then L(A) is a **regular subset** of the full Σ-tree.

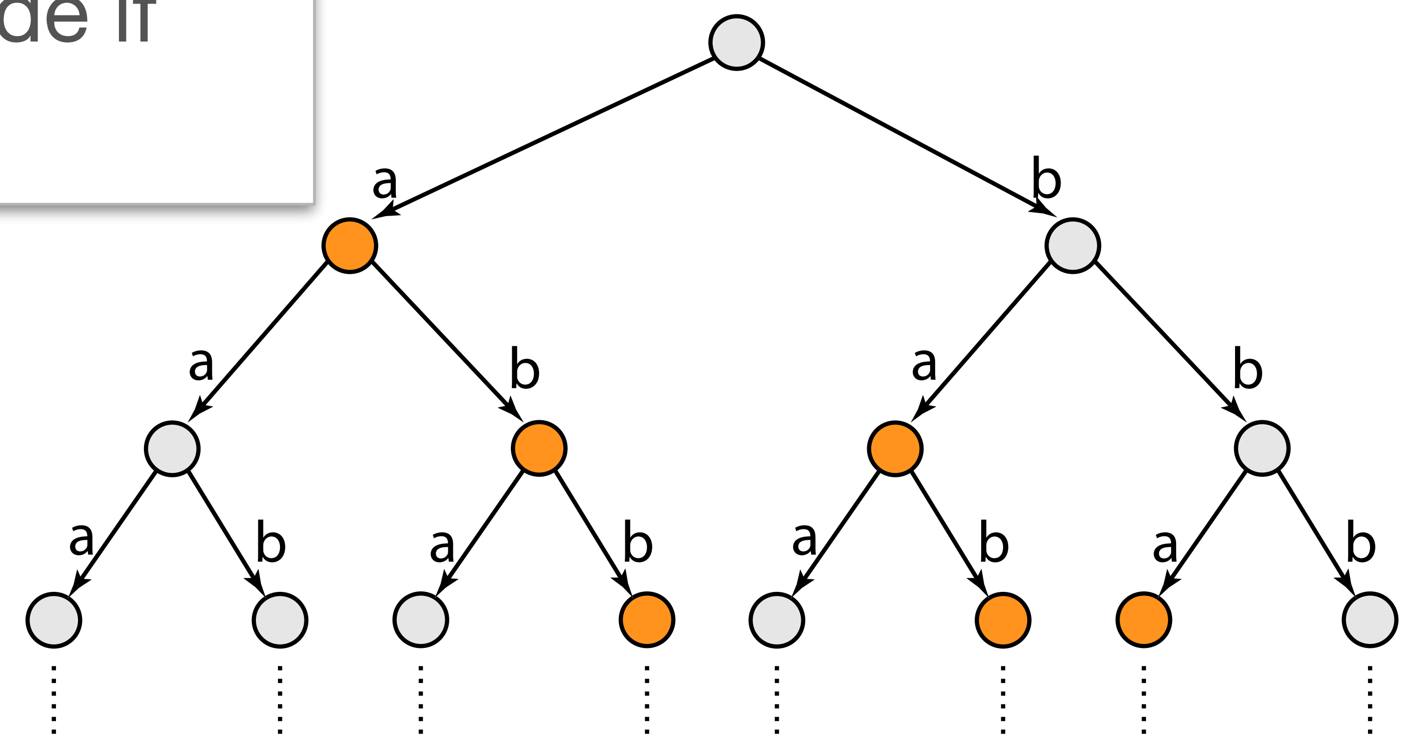
Regular subset:
Definable by an automaton, or equivalently, by **monadic second-order logic.**

MSOL: $P(x) \mid Z(x) \mid E(x,y) \mid \varphi \vee \psi \mid \neg\varphi \mid \exists x.\varphi \mid \exists Z.\varphi$

The case of finite automata

Problem P'(φ):
Given a **finite automaton** A decide if L(A) has the property φ.

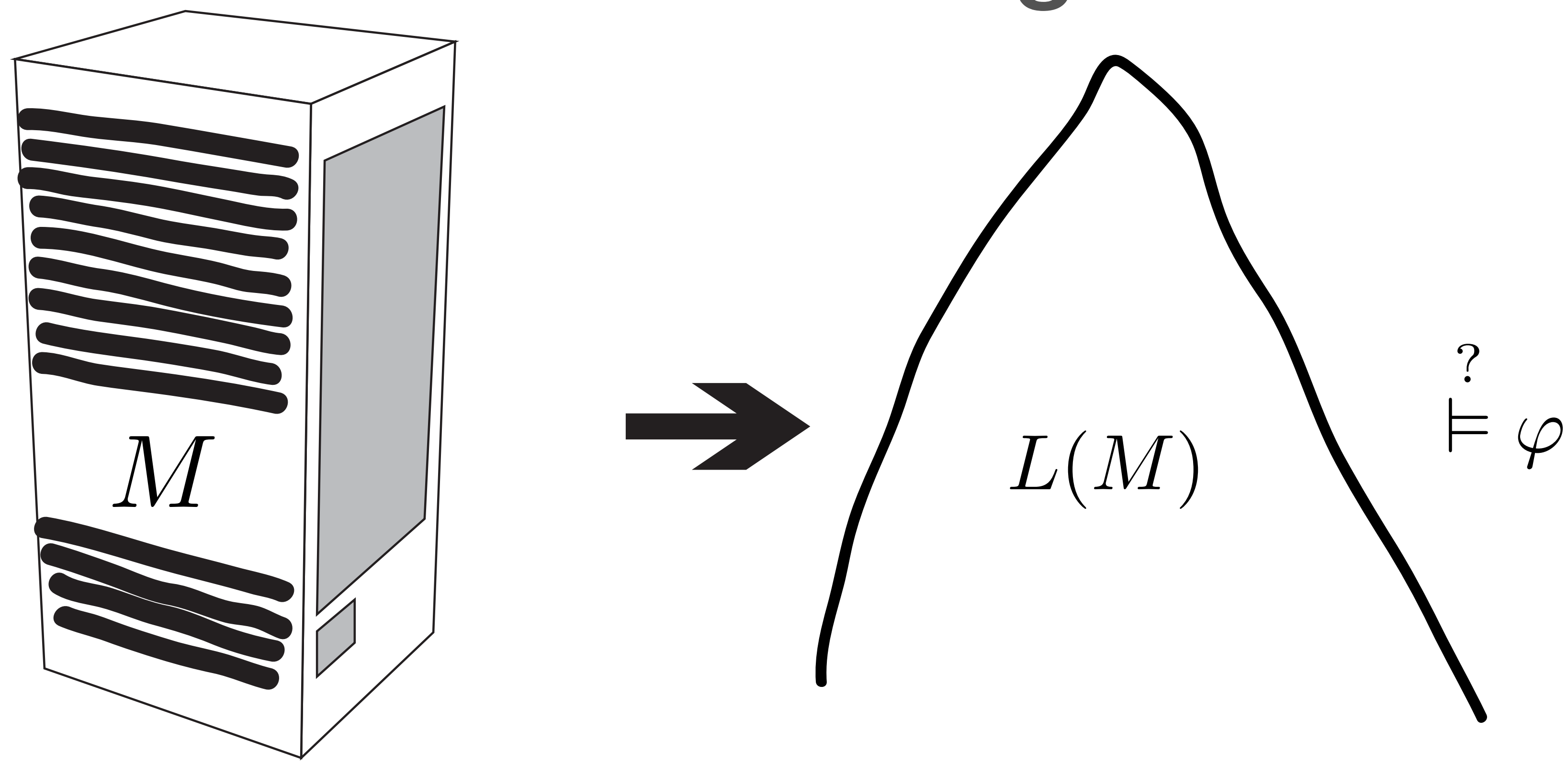
If Σ the alphabet of A, then L(A) is a **regular subset** of the full Σ-tree



Thm [Rabin'68]**:**
The **monadic second-order** theory of Σ-tree is decidable.

Corollary:
For every MSOL property φ the problem P'(φ) is decidable.

What about other means to generate languages?



Pushdown automata:

Nondeterministic: It is undecidable if L(M) contains all the words.

Deterministic: we will see later.

Uninterpreted programs (program schemes):

$$Fct(x) \equiv \textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } Fct(x-1) \cdot x \ .$$

$$\boxed{Fct(x) \;\equiv\; \textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } Fct(x-1)\cdot x\ .}$$

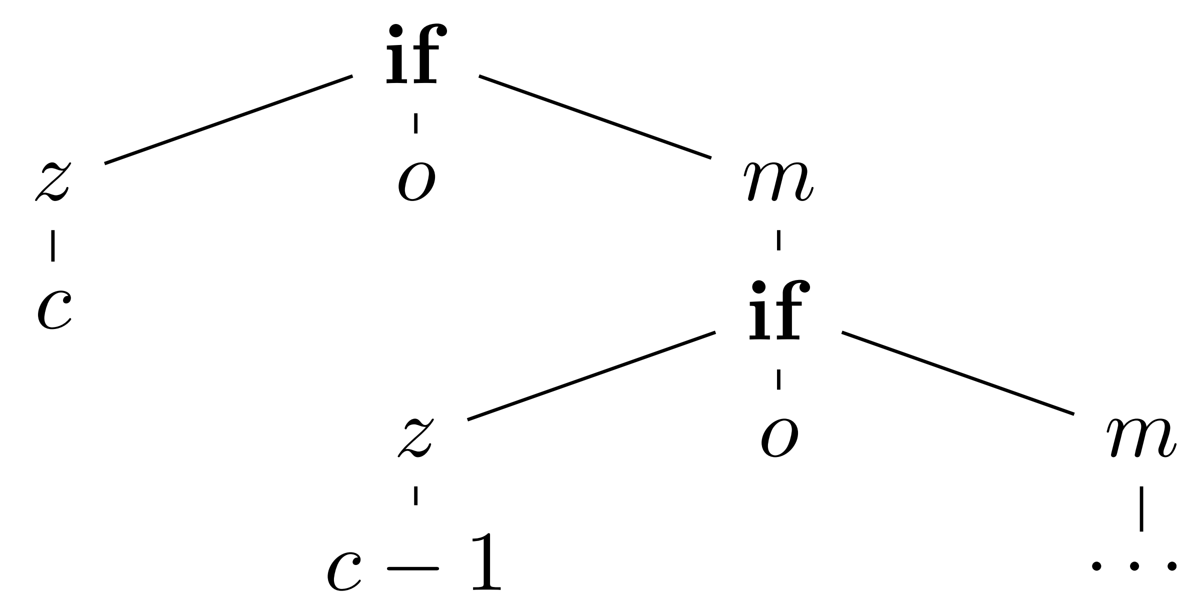
$$Fct(x) \;\equiv \textbf{if} - \textbf{then} - \textbf{else}\big(z(x), o, m(Fct(x-1), x)\big)$$

$$Fct(x) \;\equiv\; \textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } Fct(x-1)\cdot x\ .$$

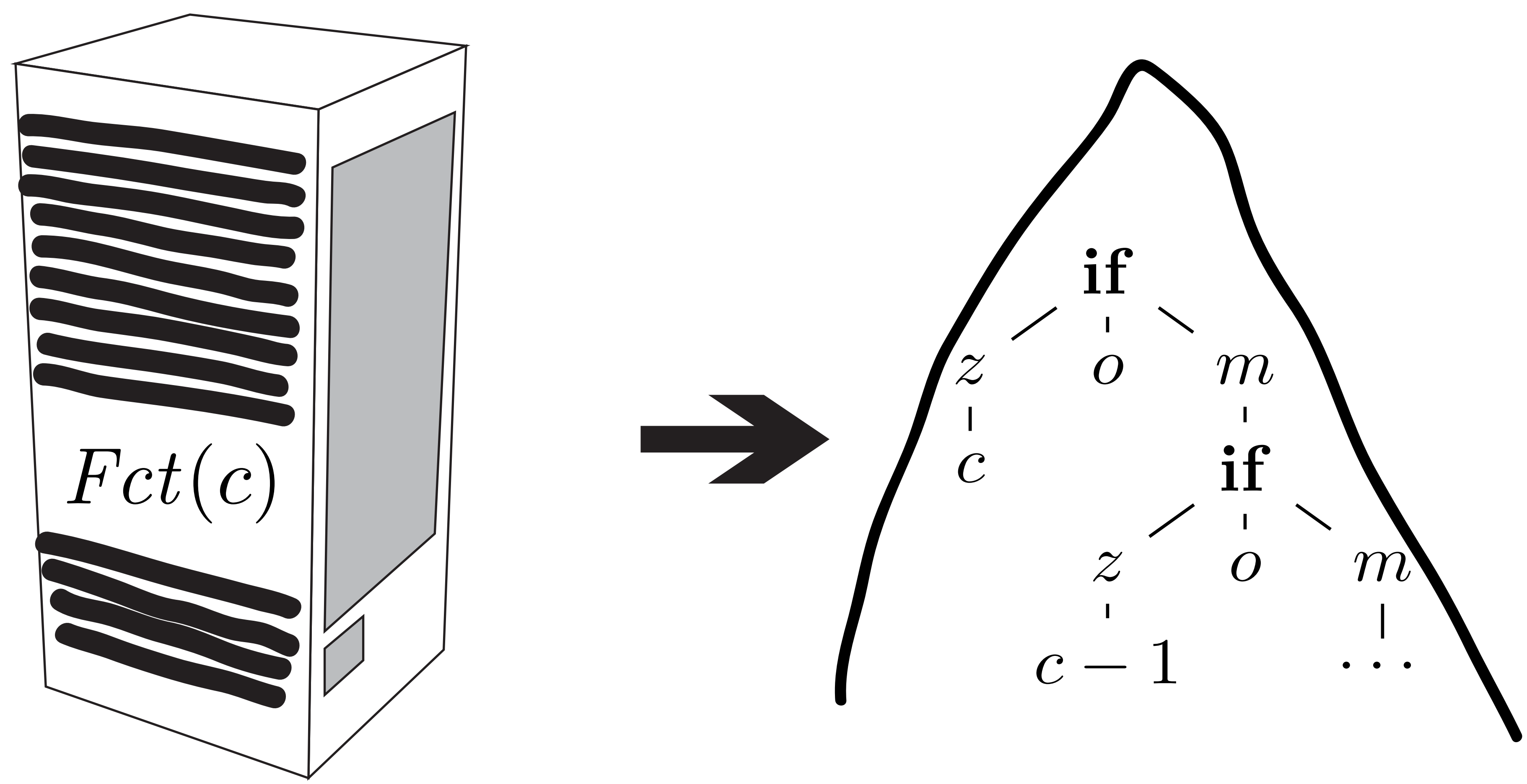
$$Fct(x) \;\equiv \textbf{if} - \textbf{then} - \textbf{else}\big(z(x), o, m(Fct(x-1), x)\big)$$

Böhm tree

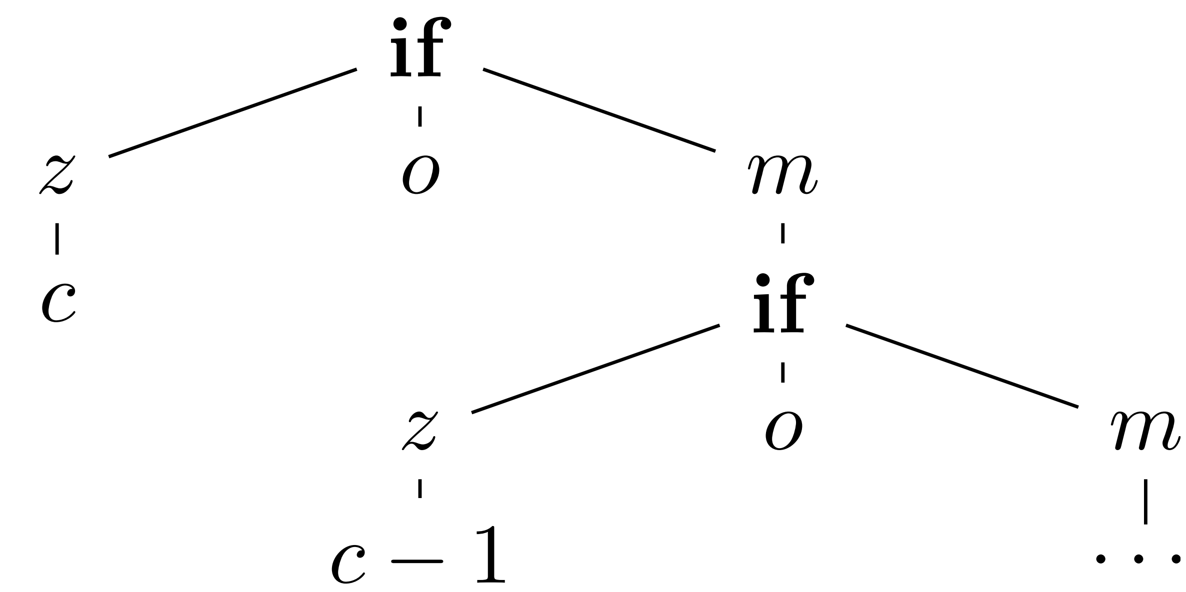
$BT(Fct(c))$ is



$$Fct(x) \equiv \mathbf{if-then-else}\big(z(x), o, m(Fct(x-1), x)\big)$$



$BT(Fct(c))$ is



Böhm-trees are interesting because:

- They reflect a part of the semantics of the program
- They have decidable monadic second order theory (MSOL)
- Interesting properties can be expressed in MSOL
  e.g. usage patterns

While programs

$$x := e \mid \texttt{if}\ x = 0\ \texttt{then}\ I_1\ \texttt{else}\ I_2 \mid \texttt{while}\ \ x > 0\ \texttt{do}\ I$$

variables range over $\mathbb{N}$ and e are arithmetic expressions

While programs

$$x := e \mid \texttt{if } x = 0 \texttt{ then } I_1 \texttt{ else } I_2 \mid \texttt{while } \ x > 0 \texttt{ do } I$$

variables range over $\mathbb{N}$ and e are arithmetic expressions

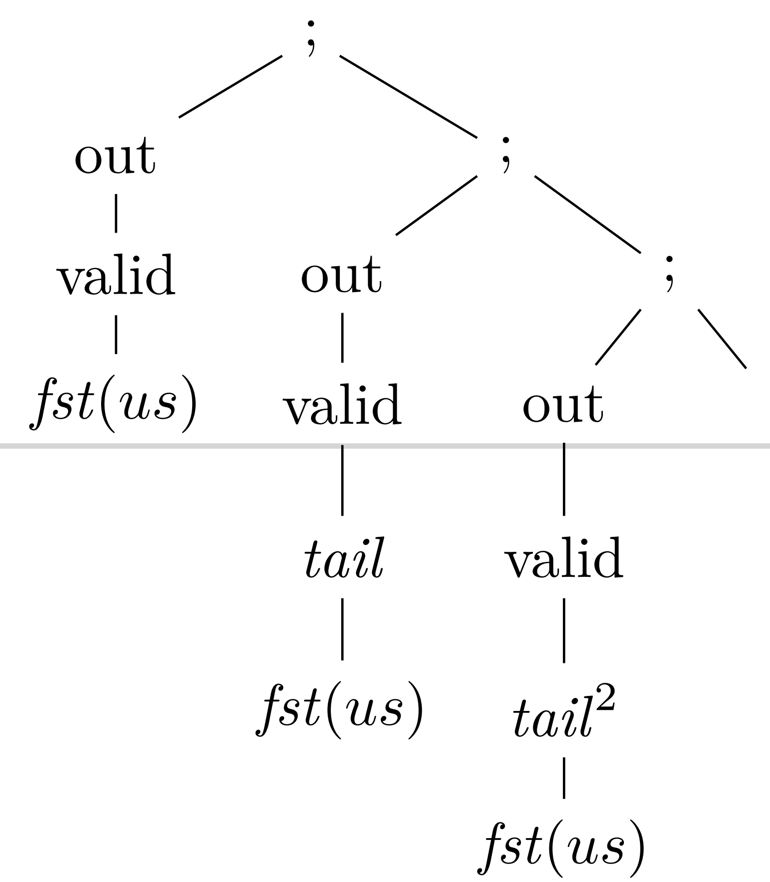
- While-programs are Turing powerful.
- Does it mean that all other programming concepts are obsolete?
- Program schemes give a way to show that they are not:

  There is a recursive program whose Böhm tree cannot be generated by any while program

uninterpreted programs ≡ λY-terms ≃ complicated control, no data

Another example

```
let validate(x)=.. in
let rec iterate(f,s)=
        let x=first(s) in
            output(f(x));
            iterate(f,tail(s));
in
    iterate(validate,untrusted_stream);
```



On every path, between an occurrence of `out` and `us` there should always be an occurrence of `valid`.

Every call to `valid` uses `us`.

Productivity: If input stream is accessed infinitely often then output is produced infinitely often.

The input stream is accessed infinitely deep: unbounded number of `tail` before `fst`.

Examples of properties

- reachability
  fail constant is reachable

- resource usage
  every open file is eventually closed

- method invocation patterns
  m.init should appear before m.usage

- fairness properties
  if access is asked infinitely often then it is granted infinitely often

1. Program → λ-term
P → M

2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \vDash \varphi$

- We would like to consider programs with: semicolon, let, and evaluation by value.
- We use λY-calculus: simply typed λ-calculus with fix point operator as our target language
- To translate programs to λY-calculus we can use some sort of CPS translation.

1. Program → λ-term

P → M

(P and M have similar Böhm trees)

2. Property → MSOL-formula

'no fail' → φ

3. Verification

$BT(M) \models \varphi$

Why Böhm trees are interesting:

- Giving denotational semantics for the full language is difficult.
- Standard denotational semantics talks about reachability/safety properties.
- A Böhm tree gives full interpretation of the control-flow,
  but does not interpret commands operating on data.

1. Program → λ-term
P → M

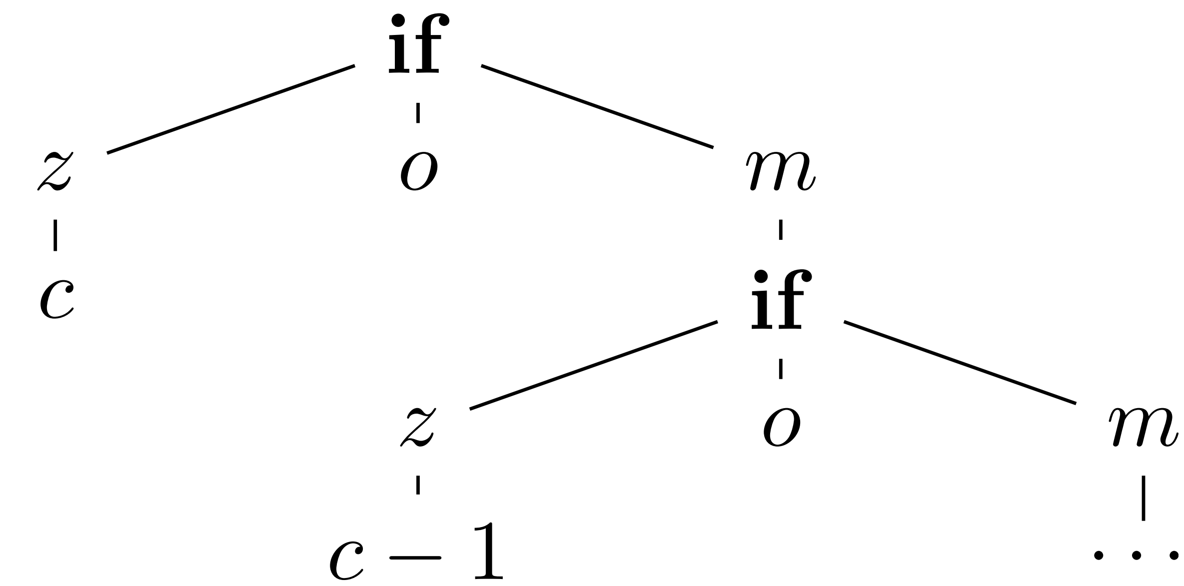
2. Property → MSOL-formula
'no fail' → φ

3. Verification
$BT(M) \models \varphi$

Why MSOL:

- Standard logic for tree properties (regular tree properties).
- Can express many interesting properties.
- The MSOL theory of a Böhm tree of a λY-term is decidable (Ong's Theorem).

$Fct(c)$ is



Two main algorithmic problems

Deciding equality of schemes:
Do two given schemes have the same trees

Decidable for order 1 schemes
[Senizergues]

Deciding MSOL theory of schemes:
Does a given MSOL formula hold in the eval-tree of a given scheme.

Decidable
[Ong]

Schemes

+ Ianov ’58
  The logical schemes of algorithms
+ Park ’68 Recursive schemes
+ Scott Elgot ’70
  Semantics via free interpretation
+ Milner ’73, Plotkin ’77 PCF

Automata

+ Aho ’68 indexed languages
+ Maslov ’74 ’76
  higher-order indexed languages
  and higher-order pushdown automata

+ Courcelle ’76 for trees: 1st order schemes = DCFL
+ Engelfreit,Schmidt ’77 IO/OI
+ Damm ’82 for languages: rec schemes = higher-order pushdowns
+ Muller, Schupp ’85: MSOL theory of pushdown trees
+ Senizergues ’97 Equivalence of 1st order schemes is decidable
+ Loader ’01 Lambda-definability is undecidable
+ Knapik, Niwinski, Urzyczyn ’02 Safe schemes = higher order-pushdowns
+ Ong ’06 MSOL theory of eval-trees of schemes is decidable

- Early beginning with Frege (1893) and Schönfinkel (1924).
- Conceived by Church (1932-1933) as part of a general theory of functions and logic.
- General theory shown inconsistent by Kleene & Rosner (1936), but the functional part has become successful.
- All computable functions are representable in lambda-calculus Kleene & Rosner (1936), Turing (1937).
  Equivalence of two lambda-terms is the first known undecidable problem.
- Typed version has been introduced by Curry (1936), and Church (1940).
- In the 60-ties Scott gives mathematical semantics to the calculus.
- Applications to functional languages, and to linguistics start.