Verification of Common Interprocedural Compiler Optimizations Using Visibly Pushdown Kleene Algebra[1]

Claude Bolduc and Béchir Ktari

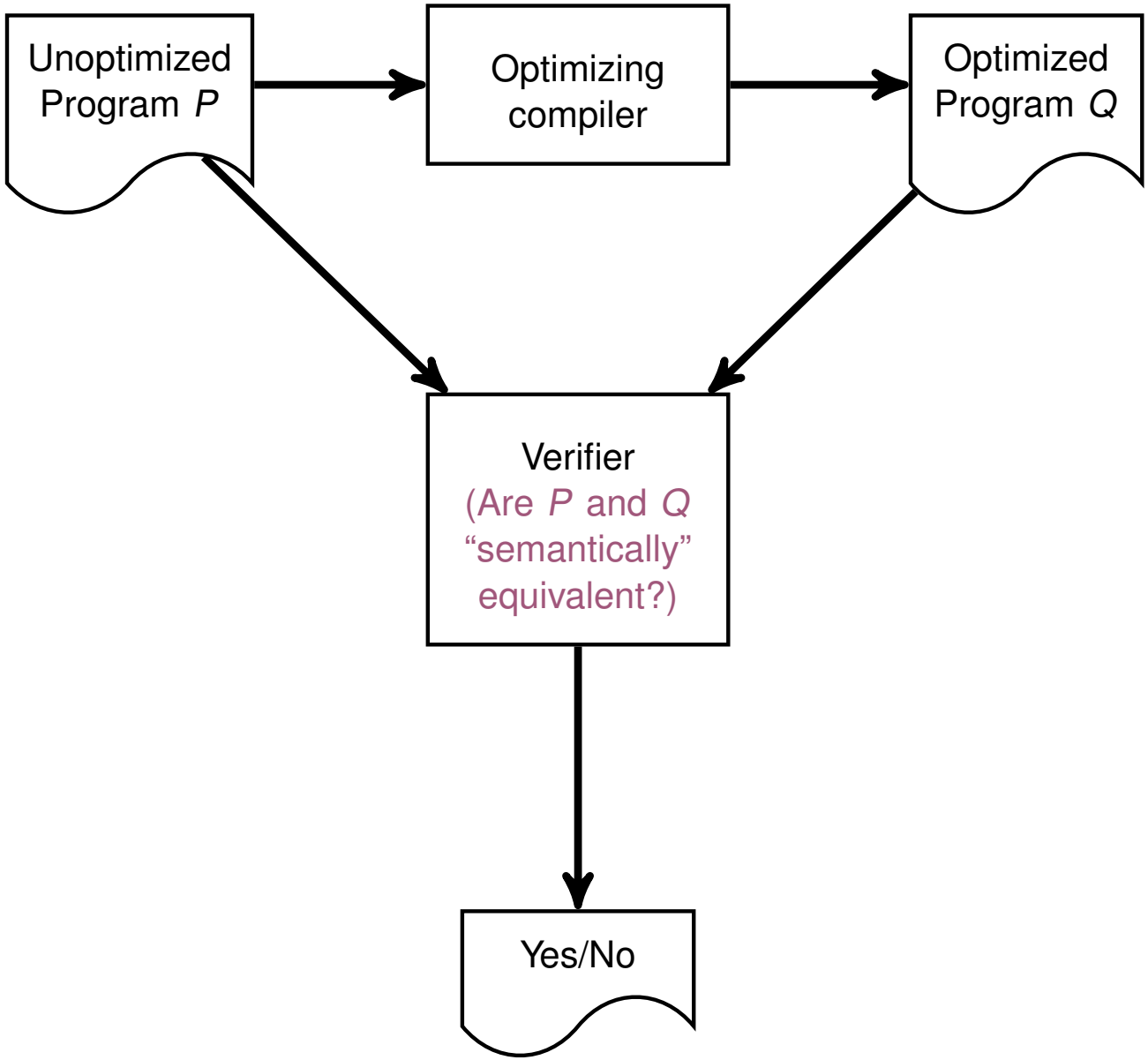
LSFM, Laval University
Québec, Canada

25 June 2010

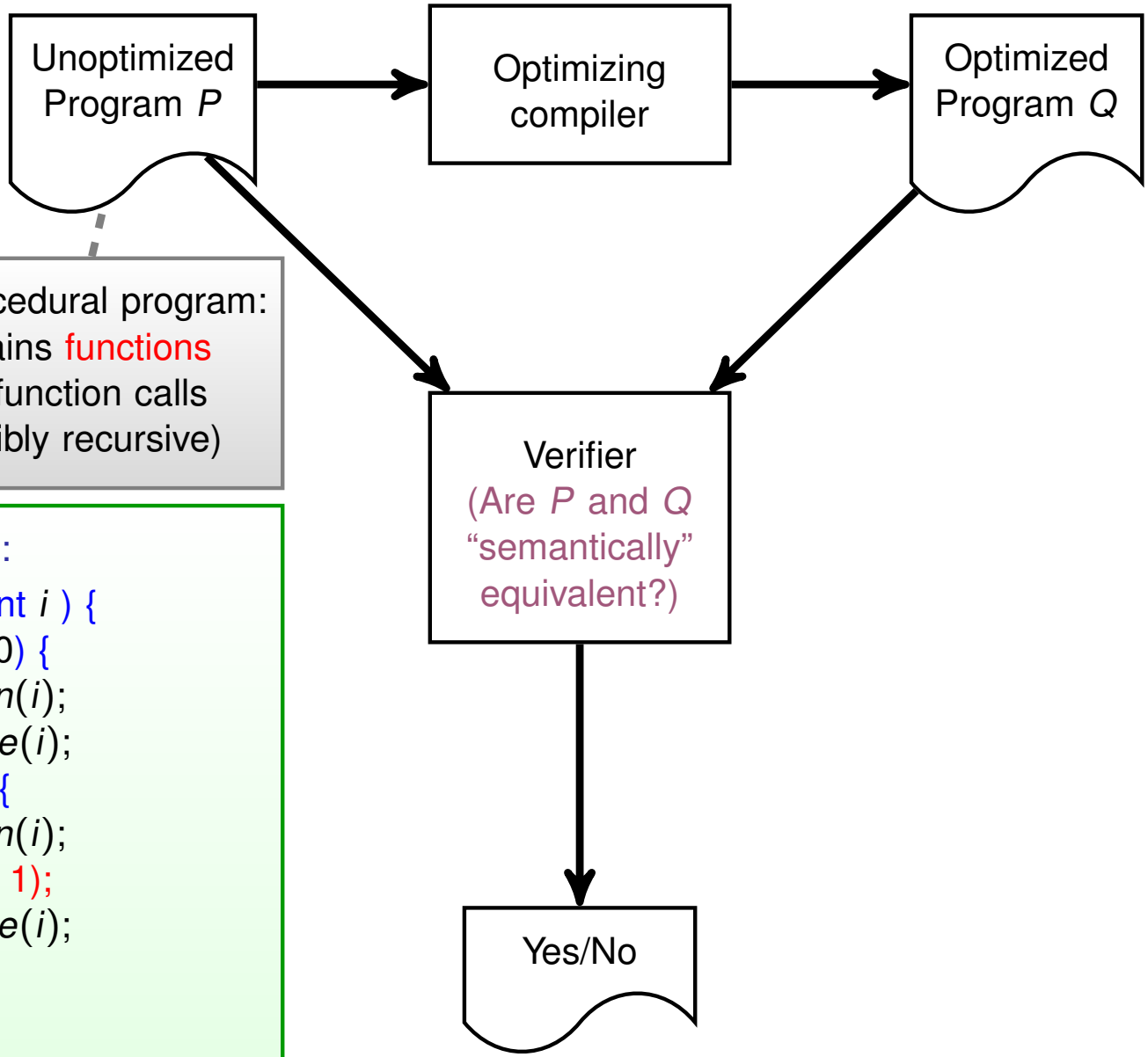
[1] This research is supported by NSERC and FQRNT.

UNIVERSITÉ LAVAL

Motivation


Unoptimized
Program *P*
Optimizing
compiler
Optimized
Program *Q*
Verifier
(Are *P* and *Q*
"semantically"
equivalent?)
Yes/No

Motivation

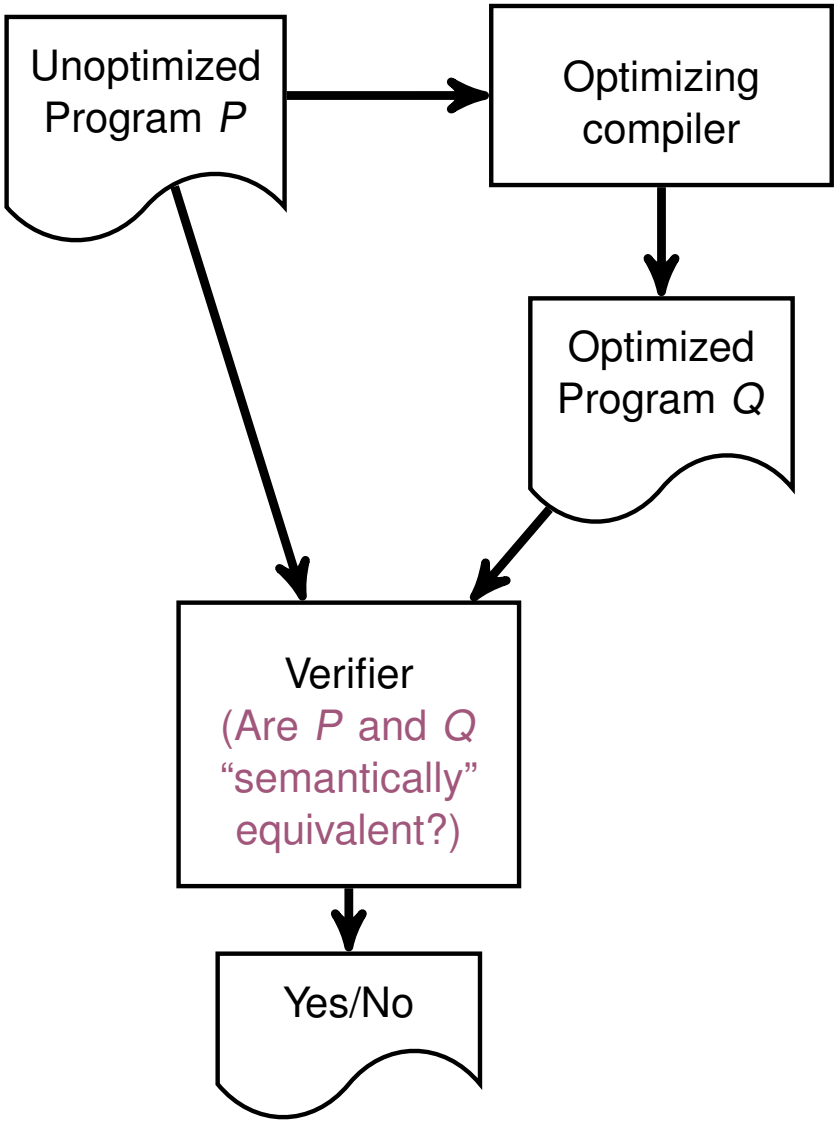


Interprocedural program:
contains functions
and function calls
(possibly recursive)

Example:

```
void f ( int i ) {
   if (i ⩽ 0) {
      open(i);
      close(i);
   } else {
      open(i);
      f(i – 1);
      close(i);
   }
}
```

Certification of *Intraprocedural* Compiler Optimizations Using KAT [Kozen and Patron, 2000]



Formal verification of some compiler optimizations including:

- dead code elimination;
- common subexpression elimination;
- copy propagation;
- loop hoisting;
- array bounds check elimination.

Main limitation:

Restricted to intraprocedural programs (no functions).

UNIVERSITÉ LAVAL

An Example of Interprocedural Dead Code Elimination

```
int main(void) {
  int x = 2;

  increment(&x);
  increment(&x);

  return 0; /* Success. */
}

void increment(int* n) {
  if (n == NULL)
    printf (stderr, "Error");
  else
    *n += 1;
}
```

The pointer n cannot be null.

An Example of Interprocedural Dead Code Elimination

```
int main(void) {
  int x = 2;

  increment(&x);
  increment(&x);

  return 0; /* Success. */
}

void increment(int* n) {



  *n += 1;
}
```

An Example of Interprocedural Dead Code Elimination

```
int main(void) {
  int x = 2;

  increment(&x);
  increment(&x);

  return 0; /* Success. */
}

void increment(int* n) {


  *n += 1;
}
```

N.B. We can also handle recursive programs.

Outline

Outline

Kleene Algebra and Kleene Algebra with Tests (KAT)

Definition (Kleene Algebra [Kozen, 1994])

A Kleene algebra is a structure $(K, +, \cdot, ^*, 0, 1)$ satisfying the axioms:

$$p + (q + r) = (p + q) + r \qquad p \cdot (q \cdot r) = (p \cdot q) \cdot r \qquad p + 0 = p$$
$$p \cdot (q + r) = p \cdot q + p \cdot r \qquad p + q = q + p \qquad p \cdot 0 = 0 = 0 \cdot p$$
$$(p + q) \cdot r = p \cdot r + q \cdot r \qquad p + p = p \qquad p \cdot 1 = p = 1 \cdot p$$

$$q \cdot p + r \leqslant p \to q^* \cdot r \leqslant p \qquad 1 + p^* \cdot p \leqslant p^*$$
$$p \cdot q + r \leqslant p \to r \cdot q^* \leqslant p \qquad 1 + p \cdot p^* \leqslant p^*$$

$$p \leqslant q \leftrightarrow p + q = q$$

It is common to add a set of tests to Kleene algebra with operators

$$\text{OR } (+), \qquad \text{AND } (\cdot), \qquad \text{NOT } (\overline{\ }).$$

Intuitive Meaning of the Operators in Program Semantics

Each atomic element of the algebra is considered to be a simple instruction (assigment statement, . . .) of a program.

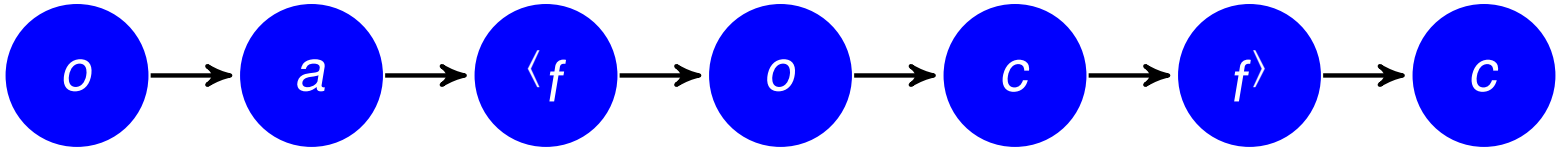
Representation of the control flow of a sequential program without procedures:

$p; q$	is represented by	$p \cdot q$
if b then p else q	is represented by	$b \cdot p + \overline{b} \cdot q$
while b do p	is represented by	$(b \cdot p)^* \cdot \overline{b}$

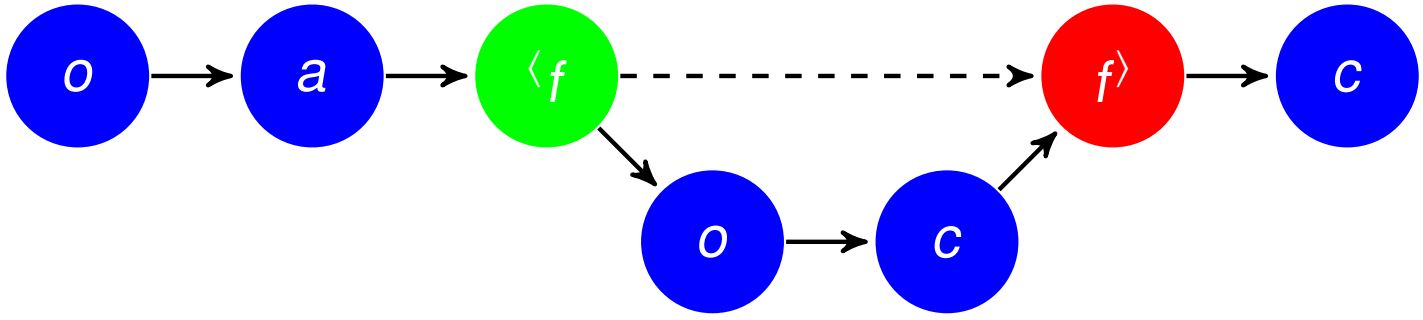
Outline

A Word Represents an Execution of an Interprocedural Program

Idea: Do not look at executions of an interprocedural program with a linear view:



Instead, see them with a nested view:



To this end, partition the set of atomic actions of a word into three types:

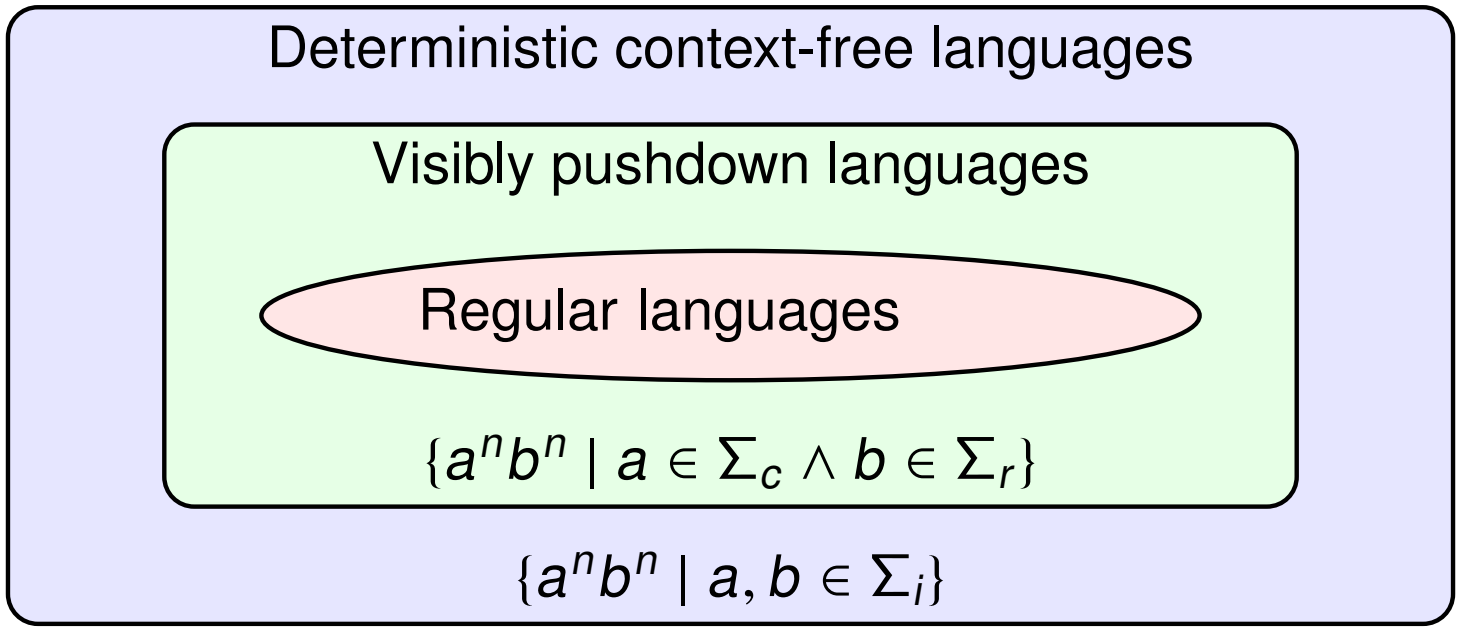
- internal actions;
- call actions;
- return actions.

Visibly Pushdown Automata [Alur *et al.*, 2004]

Intuition: Pushdown automata in which the stack manipulations are driven (made "visible") by the input word.

Idea: When a visibly pushdown automaton reads

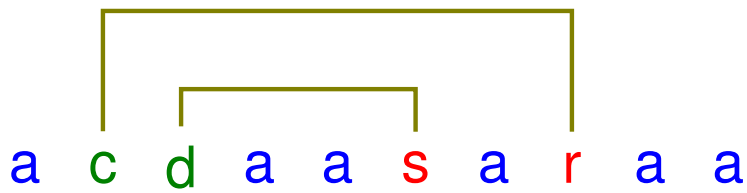
- an internal action, it cannot modify the stack;
- a call action, it must push a symbol on the stack;
- a return action, it must read the symbol on the top of the stack and pop it (unless it is the bottom-of-stack symbol).



Interesting Results on Visibly Pushdown Languages (VPLs)

- VPLs are closed under union, concatenation, Kleene star, intersection, complementation and prefix-closure.
- Their language equivalence problem is EXPTIME-complete.
- VPLs differ from regular languages mostly for their well-matched words.

Well-matched word: A word is said to be well matched if every call action in the word has a matching return action in it and vice versa.
Example:



Outline

Goal: To Have an Algebra That Axiomatises the Equational Theory of VPLs

First, we need “expressions” for the algebra.

Regular expressions alone are not sufficient.

(Atomic elements, constants 0 and 1, and operators $\cdot$, $+$, *.)

How to Represent Well-Matched Words?

Personal thought: These words are better described via context-free grammars

Idea: Extend regular expressions with a restricted set of context-free grammars.

There are different (but equivalent) ways of restricting context-free grammars to generate well-matched VPLs.

Well-Matched Visibly Pushdown Grammar (WMVPG)

Our preferred way of restricting context-free grammars

Given:

- disjoint finite sets (of atomic elements) Σ_i, Σ_c and Σ_r;
- a finite set of symbols V including symbols s and t.

Define WMVPG over Σ_i, Σ_c and Σ_r to be a tuple

$$G := (\{P_{(x,y)} \mid x, y \in V\}, P_{(s,t)}, \rightarrow)$$

where $\rightarrow$ is a finite set of explicit rewrite rules of the form

- $P_{(x,y)} \rightarrow \varepsilon$, where $x, y \in V$;
- $P_{(x,y)} \rightarrow a$, where $a \in \Sigma_i$ and $x, y \in V$;
- $P_{(x,y)} \rightarrow c\, P_{(z,w)}\, r$, where $c \in \Sigma_c$, $r \in \Sigma_r$ and $w, x, y, z \in V$

and implicit rewrite rules

- $P_{(x,y)} \rightarrow P_{(x,z)}\, P_{(z,y)}$ for each $x, y, z \in V$.

Well-Matched Visibly Pushdown Grammar (WMVPG)

Our preferred way of restricting context-free grammars

Given:

- disjoint finite sets (of atomic elements) Σ_i, Σ_c and Σ_r;
- a finite set of symbols V including symbols s and t.

Define WMVPG over Σ_i, Σ_c and Σ_r to be a tuple

$$G := (\{P_{(x,y)} \mid x, y \in V\}, P_{(s,t)}, \rightarrow)$$

Enforce the definition of a current element x and an ending element y.

where $\rightarrow$ is a finite set of explicit rewrite rules of the form

- $P_{(x,y)} \rightarrow \varepsilon$, where $x, y \in V$;
- $P_{(x,y)} \rightarrow a$, where $a \in \Sigma_i$ and $x, y \in V$;
- $P_{(x,y)} \rightarrow c\, P_{(z,w)}\, r$, where $c \in \Sigma_c$, $r \in \Sigma_r$ and $w, x, y, z \in V$

and implicit rewrite rules

- $P_{(x,y)} \rightarrow P_{(x,z)}\, P_{(z,y)}$ for each $x, y, z \in V$.

Example of a WMVPG

Let $\Sigma_i := \{a\}$, $\Sigma_c := \{c\}$, $\Sigma_r := \{r\}$ and $V := \{x, y, z\}$.

Consider a WMVPG G having starting nonterminal $P_{(x,z)}$ and explicit rewrite rules:

- $P_{(x,y)} \to \varepsilon$;
- $P_{(x,y)} \to c\, P_{(x,y)}\, r$;
- $P_{(y,z)} \to a$;
- $P_{(z,z)} \to a$.

The language generated by G is

$$\{c^n\, r^n\, a^m \mid m \geqslant 1 \wedge n \geqslant 0\} \ .$$

Example of a WMVPG

Let $\Sigma_i := \{a\}$, $\Sigma_c := \{c\}$, $\Sigma_r := \{r\}$ and $V := \{x, y, z\}$.

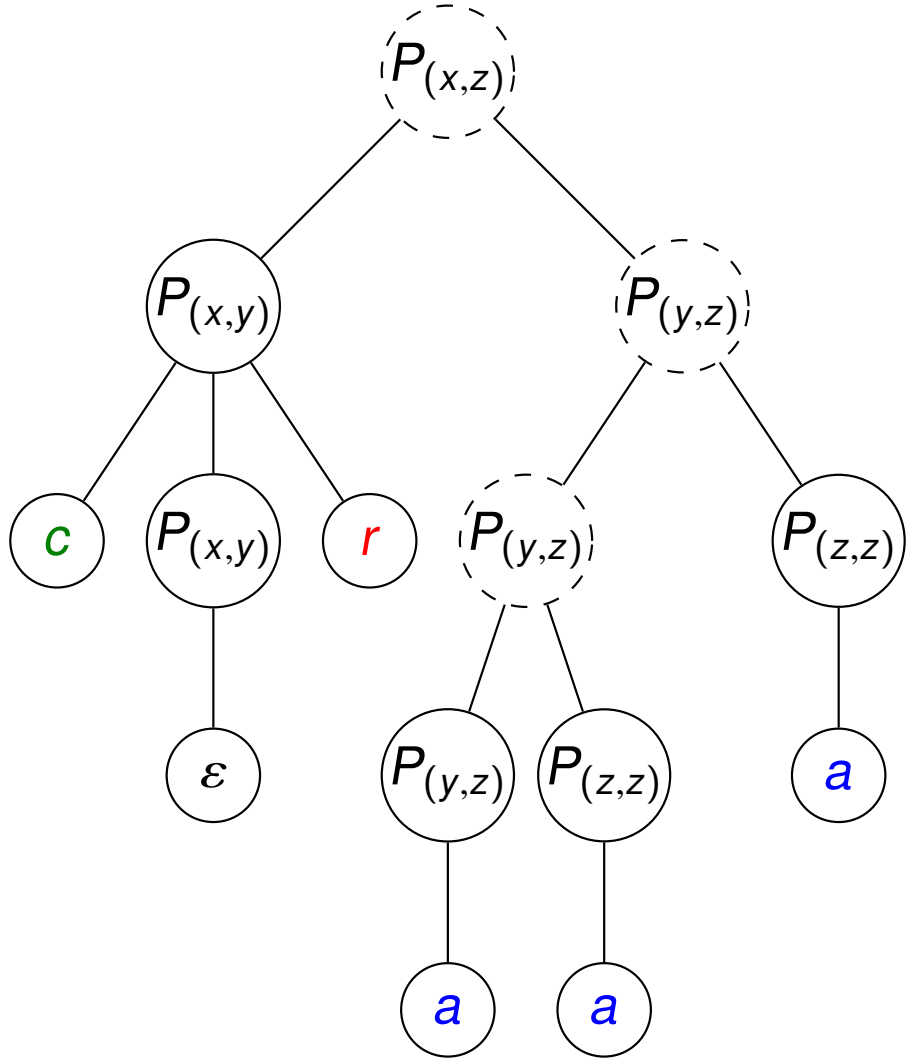
Consider a WMVPG G having starting nonterminal $P_{(x,z)}$ and explicit rewrite rules:

- $P_{(x,y)} \to \varepsilon$;
- $P_{(x,y)} \to c\, P_{(x,y)}\, r$;
- $P_{(y,z)} \to a$;
- $P_{(z,z)} \to a$.

The language generated by G is

$$\{c^n\, r^n\, a^m \mid m \geqslant 1 \wedge n \geqslant 0\} \ .$$

Example of a derivation tree for $c\, r\, a\, a\, a$:



Grammar Patterns (Partial Operators)

A *grammar pattern* of a WMVPG G on Σ_i, Σ_c and Σ_r is a *partial operator* obtained by

- replacing the terminals in G by variables (placeholders) of the same type (Σ_i, Σ_c or Σ_r);
  Examples:
  - $P_{(x,y)} \to \boxed{\alpha}\, P_{(x,y)}\, \boxed{\beta}$
  - $P_{(y,z)} \to \boxed{\gamma}$
- writing $P_{(x,y)} \to 1$ instead of $P_{(x,y)} \to \varepsilon$;
- adding, for convenience, rules of the form $P_{(x,y)} \to 0$.

The arity of a grammar pattern is the number of variables it contains.

Define $\mathcal{G}$ as the set of all grammar patterns on Σ_i, Σ_c and Σ_r.

Visibly Pushdown Regular Expressions (VPRE)

Definition

A visibly pushdown regular expression (on an alphabet $\Sigma = \Sigma_i \cup \Sigma_c \cup \Sigma_r$) is any well-formed expression that can be generated from the base elements:

$$0, \qquad 1, \qquad \{a \mid a \in \Sigma\},$$

the operators

$$\cdot, \qquad +, \qquad ^*,$$

and the operators of $\mathcal{G}$.

Theorem (Theorem *à la* Kleene for visibly pushdown languages)

Visibly Pushdown Regular Expressions (VPRE)

Definition

A visibly pushdown regular expression (on an alphabet $\Sigma = \Sigma_i \cup \Sigma_c \cup \Sigma_r$) is any well-formed expression that can be generated from the base elements:

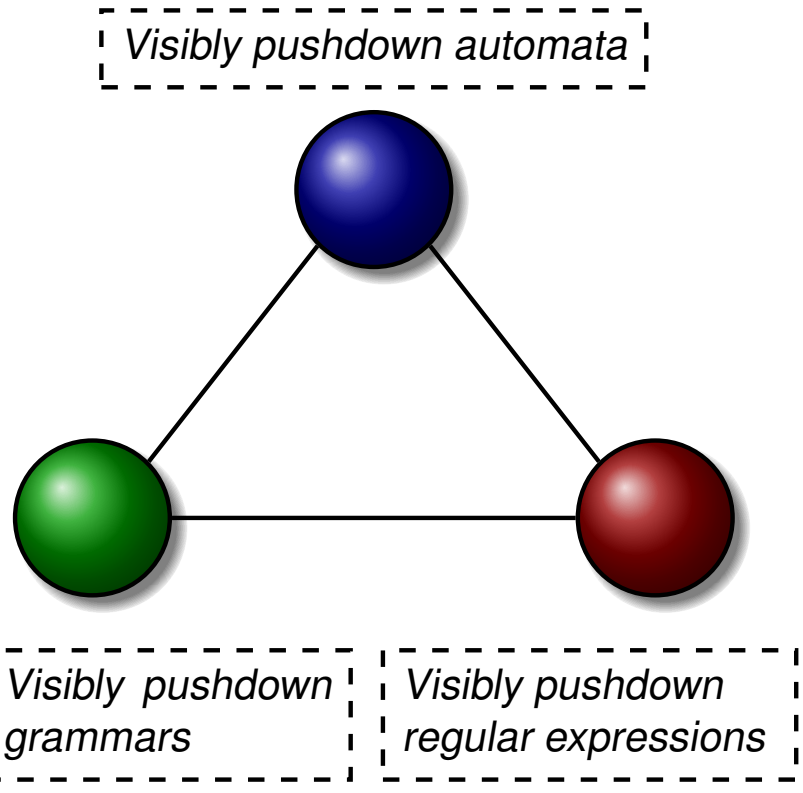
$$0, \qquad 1, \qquad \{a \mid a \in \Sigma\},$$

the operators

$$\cdot, \qquad +, \qquad ^*,$$

and the operators of $\mathcal{G}$.

Theorem (Theorem *à la* Kleene for visibly pushdown languages)



Motivation for a Concise Notation for Grammar Patterns

Personal thought: It is difficult to see the structure of a grammar pattern when writing it “linearly” in a VPRE.

“Linear” notation for G:

$$(P_{(x,z)}, \{P_{(x,y)} \to 1,\ P_{(x,y)} \to c\, P_{(x,y)}\, r,\ P_{(y,z)} \to a,\ P_{(z,z)} \to a\})$$

Block Notation for Grammar Patterns

For explicit rewrite rules:

$P_{(x,y)} \to \gamma$	is written	$\underset{x}{[\,\gamma\,]^{y}}$	Unary block
$P_{(x,y)} \to 1$	is written	$\underset{x}{[\,1\,]^{y}}$	Unary block
$P_{(x,y)} \to 0$	is written	$\underset{x}{[\,0\,]^{y}}$	Unary block

Separate the structure on different “lines”.
More concise.

Block Notation for Grammar Patterns

For explicit rewrite rules:

$P_{(x,y)} \to \gamma$	is written	$\underset{x}{[}\ \gamma\ \overset{y}{]}$	Unary block
$P_{(x,y)} \to 1$	is written	$\underset{x}{[}\ 1\ \overset{y}{]}$	Unary block
$P_{(x,y)} \to 0$	is written	$\underset{x}{[}\ 0\ \overset{y}{]}$	Unary block
$P_{(x,y)} \to \alpha\, P_{(z,w)}\, \beta$	is written	$\underset{x}{[}\ \alpha \underset{z}{\downarrow} \overset{w}{\uparrow} \beta\ \overset{y}{]}$	Binary block

Emphasize call and return symbols.

Block Notation for Grammar Patterns

For explicit rewrite rules:

$P_{(x,y)} \to \gamma$	is written	$\underset{x}{[}\,\gamma\,\overset{y}{]}$	Unary block
$P_{(x,y)} \to 1$	is written	$\underset{x}{[}\,1\,\overset{y}{]}$	Unary block
$P_{(x,y)} \to 0$	is written	$\underset{x}{[}\,0\,\overset{y}{]}$	Unary block
$P_{(x,y)} \to \alpha\, P_{(z,w)}\, \beta$	is written	$\underset{x}{[}\,\alpha \underset{z}{\downarrow} \overset{w}{\uparrow} \beta\,\overset{y}{]}$	Binary block

For the starting nonterminal $P_{(s,t)}$:

Let $\mathcal{B}$ be the block notation of the explicit rewrite rules of a grammar pattern.

$P_{(s,t)}$ is used to surround $\mathcal{B}$ by writing $\underset{s}{(\!|}\,\mathcal{B}\,\overset{t}{|\!)}$

Comparison of the “Linear” and Block Notations

“Linear” notation for G:

$$(P_{(x,z)}, \{P_{(x,y)} \to 1,\ P_{(x,y)} \to c\, P_{(x,y)}\, r,\ P_{(y,z)} \to a,\ P_{(z,z)} \to a\})$$

Block notation for G:

$$\left(\!\!\left| \begin{smallmatrix} & y \\ x & x \end{smallmatrix} \right[1 \left] \right|\!\!\right), \left[\begin{smallmatrix} & y \\ x & x \end{smallmatrix} c \begin{smallmatrix} \downarrow \\ x \end{smallmatrix} \begin{smallmatrix} y \\ \uparrow \end{smallmatrix} r \right], \left[\begin{smallmatrix} & z \\ y & \end{smallmatrix} a \right], \left[\begin{smallmatrix} & z \\ z & \end{smallmatrix} a \right]$$

Outline

Visibly Pushdown Kleene algebra (VPKA) (1)

Goal: To have an algebra that axiomatises the equational theory of VPLs.

VPKA is a structure $(K, +, \cdot, ^*, 0, 1, \mathcal{G})$ such that

- the structure $(K, +, \cdot, ^*, 0, 1)$ is a Kleene algebra;
- the following axioms are satisfied for any valid finite set of explicit rewrite rules $\mathcal{B}$ (in block notation):

$$\left(\!\!\left| \mathcal{B} \right|\!\!\right)_x^y \geqslant a \qquad \text{for any unary block } [_x\, a\,]^y \text{ of } \mathcal{B}$$

$$\left(\!\!\left| \mathcal{B} \right|\!\!\right)_x^y \geqslant c \cdot \left(\!\!\left| \mathcal{B} \right|\!\!\right)_z^w \cdot r \qquad \text{for any binary block } [_x\, c \downarrow_z \uparrow^w r\,]^y \text{ of } \mathcal{B}$$

Axioms based on *explicit* rewrite rules

Visibly Pushdown Kleene algebra (VPKA) (1)

Goal: To have an algebra that axiomatises the equational theory of VPLs.

VPKA is a structure $(K, +, \cdot, ^*, 0, 1, \mathcal{G})$ such that

- the structure $(K, +, \cdot, ^*, 0, 1)$ is a Kleene algebra;
- the following axioms are satisfied for any valid finite set of explicit rewrite rules $\mathcal{B}$ (in block notation):

$$\left(\!\!\left| \mathcal{B} \right|\!\!\right)_x^y \geqslant a \qquad \text{for any unary block } [_x\, a\,]^y \text{ of } \mathcal{B}$$

$$\left(\!\!\left| \mathcal{B} \right|\!\!\right)_x^y \geqslant c \cdot \left(\!\!\left| \mathcal{B} \right|\!\!\right)_z^w \cdot r \qquad \text{for any binary block } [_x\, c \downarrow_z \uparrow^w r\,]^y \text{ of } \mathcal{B}$$

$$\left(\!\!\left| \mathcal{B} \right|\!\!\right)_x^y \geqslant \left(\!\!\left| \mathcal{B} \right|\!\!\right)_x^{y'} \cdot \left(\!\!\left| \mathcal{B} \right|\!\!\right)_{y'}^y$$

Axiom based on *implicit* rewrite rules

Visibly Pushdown Kleene algebra (VPKA) (2)

$$\Big(\wedge\, u, u' \mid (u,u') \in \mathsf{F}^*_{\mathcal{B}}(\{(x,y)\}) :$$

$$\quad (\wedge\, a \mid [\,{}_{u}\, a \,{}^{u'}_{}\,] \in \mathcal{B}^1 : s_{(u,u')} \geqslant a)$$

$$\wedge\ (\wedge\, a, v \mid [\,{}_{u}\, a \,{}^{v}_{}\,] \in \mathcal{B}^1 : s_{(u,u')} \geqslant a \cdot s_{(v,u')})$$

$$\wedge\ (\wedge\, c, z, r, w \mid [\,{}_{u}\, c \downarrow_{z} \uparrow^{w} r \,{}^{u'}_{}\,] \in \mathcal{B}^2 : s_{(u,u')} \geqslant c \cdot s_{(z,w)} \cdot r)$$

$$\wedge\ (\wedge\, c, z, r, w, v \mid [\,{}_{u}\, c \downarrow_{z} \uparrow^{w} r \,{}^{v}_{}\,] \in \mathcal{B}^2 : s_{(u,u')} \geqslant c \cdot s_{(z,w)} \cdot r \cdot s_{(v,u')}) \Big)$$

$$\rightarrow\ s_{(x,y)} \geqslant (\!|\, \mathcal{B} \,|\!)^{y}_{x}$$

It is the least solution

- Similar axiom for the backward strategy

Visibly Pushdown Kleene algebra (VPKA) (2)

$$\Big(\wedge\, u, u' \mid (u,u') \in \mathrm{F}^*_{\mathcal{B}}(\{(x,y)\}) :$$

$$\quad (\wedge\, a \mid [\,{}_{u}\, a \,{}^{u'}_{}\,] \in \mathcal{B}^1 : s_{(u,u')} \geqslant a)$$

$$\wedge\ (\wedge\, a, v \mid [\,{}_{u}\, a \,{}^{v}_{}\,] \in \mathcal{B}^1 : s_{(u,u')} \geqslant a \cdot s_{(v,u')})$$

$$\wedge\ (\wedge\, c, z, r, w \mid [\,{}_{u}\, c \downarrow_{z} \uparrow^{w} r \,{}^{u'}_{}\,] \in \mathcal{B}^2 : s_{(u,u')} \geqslant c \cdot s_{(z,w)} \cdot r)$$

$$\wedge\ (\wedge\, c, z, r, w, v \mid [\,{}_{u}\, c \downarrow_{z} \uparrow^{w} r \,{}^{v}_{}\,] \in \mathcal{B}^2 : s_{(u,u')} \geqslant c \cdot s_{(z,w)} \cdot r \cdot s_{(v,u')})\Big)$$

$$\rightarrow\ s_{(x,y)} \geqslant (\!|\, \mathcal{B} \,|\!)^{y}_{x}$$

Solutions work for *explicit* rewrite rules

- Similar axiom for the backward strategy

Visibly Pushdown Kleene algebra (VPKA) (2)

$$\Big(\wedge\, u, u' \mid (u,u') \in \mathsf{F}^*_{\mathcal{B}}(\{(x,y)\}) : \\ \quad (\wedge\, a \mid \underset{u}{[}\, a \,\overset{u'}{]} \in \mathcal{B}^1 : s_{(u,u')} \geqslant a) \\ \wedge\ (\wedge\, a, v \mid \underset{u}{[}\, a \,\overset{v}{]} \in \mathcal{B}^1 : s_{(u,u')} \geqslant a \cdot s_{(v,u')}) \\ \wedge\ (\wedge\, c, z, r, w \mid \underset{u}{[}\, c \,\underset{z}{\downarrow} \overset{w}{\uparrow}\, r \,\overset{u'}{]} \in \mathcal{B}^2 : s_{(u,u')} \geqslant c \cdot s_{(z,w)} \cdot r) \\ \wedge\ (\wedge\, c, z, r, w, v \mid \underset{u}{[}\, c \,\underset{z}{\downarrow} \overset{w}{\uparrow}\, r \,\overset{v}{]} \in \mathcal{B}^2 : s_{(u,u')} \geqslant c \cdot s_{(z,w)} \cdot r \cdot s_{(v,u')}) \Big) \\ \rightarrow\ s_{(x,y)} \geqslant \underset{x}{(\!|}\, \mathcal{B} \,\overset{y}{|\!)}$$

Solutions work for *implicit* rewrite rules when using the *forward strategy*

- Similar axiom for the backward strategy

Visibly Pushdown Kleene algebra (VPKA) (2)

$$\Big(\wedge\, u, u' \mid (u,u') \in \mathsf{F}^*_{\mathcal{B}}(\{(x,y)\}) : \\ \quad (\wedge\, a \mid [_{u}\, a\,]^{u'} \in \mathcal{B}^1 : s_{(u,u')} \geqslant a) \\ \quad \wedge\ (\wedge\, a, v \mid [_{u}\, a\,]^{v} \in \mathcal{B}^1 : s_{(u,u')} \geqslant a \cdot s_{(v,u')}) \\ \quad \wedge\ (\wedge\, c, z, r, w \mid [_{u}\, c \downarrow_{z} \uparrow^{w} r\,]^{u'} \in \mathcal{B}^2 : s_{(u,u')} \geqslant c \cdot s_{(z,w)} \cdot r) \\ \quad \wedge\ (\wedge\, c, z, r, w, v \mid [_{u}\, c \downarrow_{z} \uparrow^{w} r\,]^{v} \in \mathcal{B}^2 : s_{(u,u')} \geqslant c \cdot s_{(z,w)} \cdot r \cdot s_{(v,u')}) \Big) \\ \rightarrow\ s_{(x,y)} \geqslant (\!|\, \mathcal{B} \,|\!)^{y}_{x}$$

- Similar axiom for the backward strategy

Theoretical Results for VPKA

- The axiomatization of VPKA presented here is “lighter” than the “original” complete axiomatization.
- Two axioms were omitted because they are not used in this work.
- The full axiomatization represents exactly the equational theory of VPLs and is EXPTIME-complete.

Visibly Pushdown Kleene algebra with Tests

We can add tests to VPKA.

(Boolean algebra)

Can We Allow Less Restricted Explicit Rewrite Rules?

Yes!

We call them metablocks.

Metablocks are only abbreviations to use regular expression's operators in blocks:

- $[_x\, p \cdot q\,]^{x'} := [_x\, p\,]^{y}, [_y\, q\,]^{x'}$ for a fresh label y;
- $[_x\, p + q\,]^{x'} := [_x\, p\,]^{x'}, [_x\, q\,]^{x'}$;
- $[_x\, p^*\,]^{x'} := [_x\, 1\,]^{x'}, [_x\, p\,]^{x'}, [_x\, p\,]^{y}, [_y\, p\,]^{x'}, [_y\, p\,]^{y}$ for a fresh label y.

Most of the laws of VPKA extend "naturally" to metablocks.

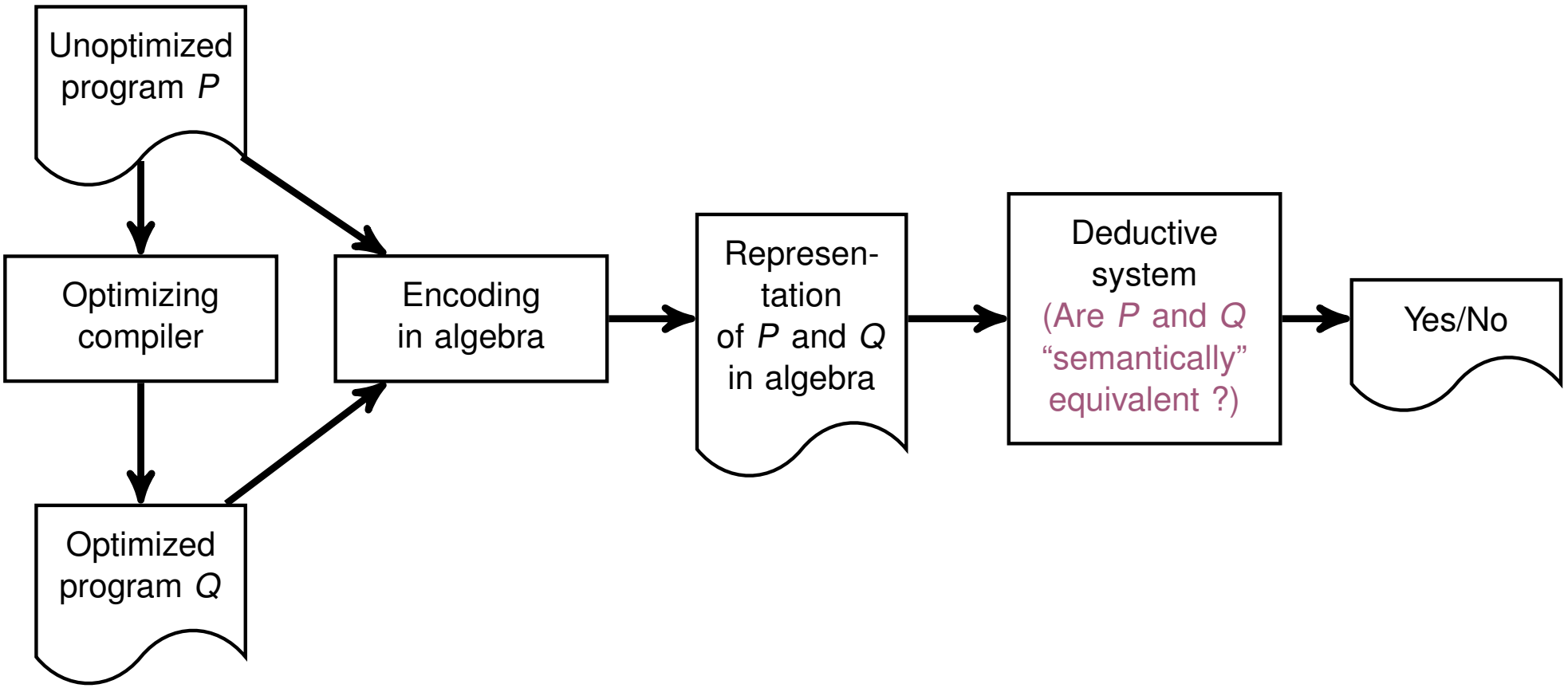
Outline

What We Have Done

Using the VPKA formalism, we were able to certify the correctness of several examples of common interprocedural compiler optimizations:

- Interprocedural dead code elimination
- Inlining of functions
- Tail-recursion elimination
- Procedure reordering
- Function cloning

How Did We Do That?



An Example of Interprocedural Dead Code Elimination

```
int main(void) {
  int x = 2;

  increment(&x);
  increment(&x);

  return 0; /* Success. */
}

void increment(int* n) {
  if (n == NULL)
    printf(stderr, "Error");
  else
    *n += 1;
}
```

The pointer n cannot be null.

An Example of Interprocedural Dead Code Elimination

```
int main(void) {
  int x = 2;

  increment(&x);
  increment(&x);

  return 0; /* Success. */
}

void increment(int* n) {



  *n += 1;
}
```

Encoding of a Program in VPKA

Three steps:

1. Define the desired abstraction for atomic program instructions and variables through the atomic sets;
2. Encode the program's control flow by a VPRE;
3. Encode the desired semantics of atomic program instructions, variables and variable passing mechanism by a set of equational hypotheses $\mathcal{H}$.

N.B. Currently, the framework deals only with halting programs.

Encoding of the Control Flow by a VPRE

Any function f gives a metablock $[_f\ s\ ]^\tau$ where

- τ is the symbol used to indicate the end of the body of any function
- s is the body of the function expressed as a "metablock expression". In particular,

p; q	is represented by	$p \cdot q$
if b then p else q	is represented by	$b \cdot p + \overline{b} \cdot q$
while b do p	is represented by	$(b \cdot p)^* \cdot \overline{b}$

Encoding of the Control Flow of the Example

```
int main(void) {
  int x = 2;

  increment(&x);
  increment(&x);

  return 0; /* Success. */
}

void increment(int* n) {
  if (n == NULL)
    printf (stderr, "Error");
  else
    *n += 1;
}
```

```
m() {
  s

  f ()
  f ()

  t
}

f () {
  if (b̄)
    p
  else
    q
}
```

Encoding of the Control Flow of the Example

```
int main(void) {
  int x = 2;

  increment(&x);
  increment(&x);

  return 0; /* Success. */
}

void increment(int* n) {
  if (n == NULL)
    printf (stderr, "Error");
  else
    *n += 1;
}
```

```
m() {
  s

  f ()
  f ()

  t
}

f () {
  if (b̄)
    p
  else
    q
}
```

Let $\langle f \rangle := (\langle f \downarrow_{f}^{\tau} \uparrow f \rangle)$.

$[\, s \cdot \langle f \rangle \cdot \langle f \rangle \cdot t \,]_{m}^{\tau}$,

$[\, \bar{b} \cdot p + b \cdot q \,]_{f}^{\tau}$

Encoding of the Control Flow of the Example

```
int main(void) {
  int x = 2;

  increment(&x);
  increment(&x);

  return 0; /* Success. */
}

void increment(int* n) {
  if (n == NULL)
    printf(stderr, "Error");
  else
    *n += 1;
}
```

```
m() {
  s

  f()
  f()

  t
}

f() {
  if (b̄)
    p
  else
    q
}
```

$$\left(\!\!\left| \underset{m'}{\overset{\tau}{[\,\langle m\rangle\,]}}, \underset{m'}{\overset{\tau}{[\,\langle m\rangle\,]}}, \underset{m}{\overset{\tau}{[\,s\cdot\langle f\rangle\cdot\langle f\rangle\cdot t\,]}}, \underset{f}{\overset{\tau\ \tau}{[\,\bar{b}\cdot p + b\cdot q\,]}} \right|\!\!\right)$$

Encoding of the Desired Semantics

Powerful abstraction mechanism:

- Allows to concentrate on “useful aspects” of the program (Abstraction of the data flow);
- Refine a model by adding hypotheses.

Some hypotheses used:

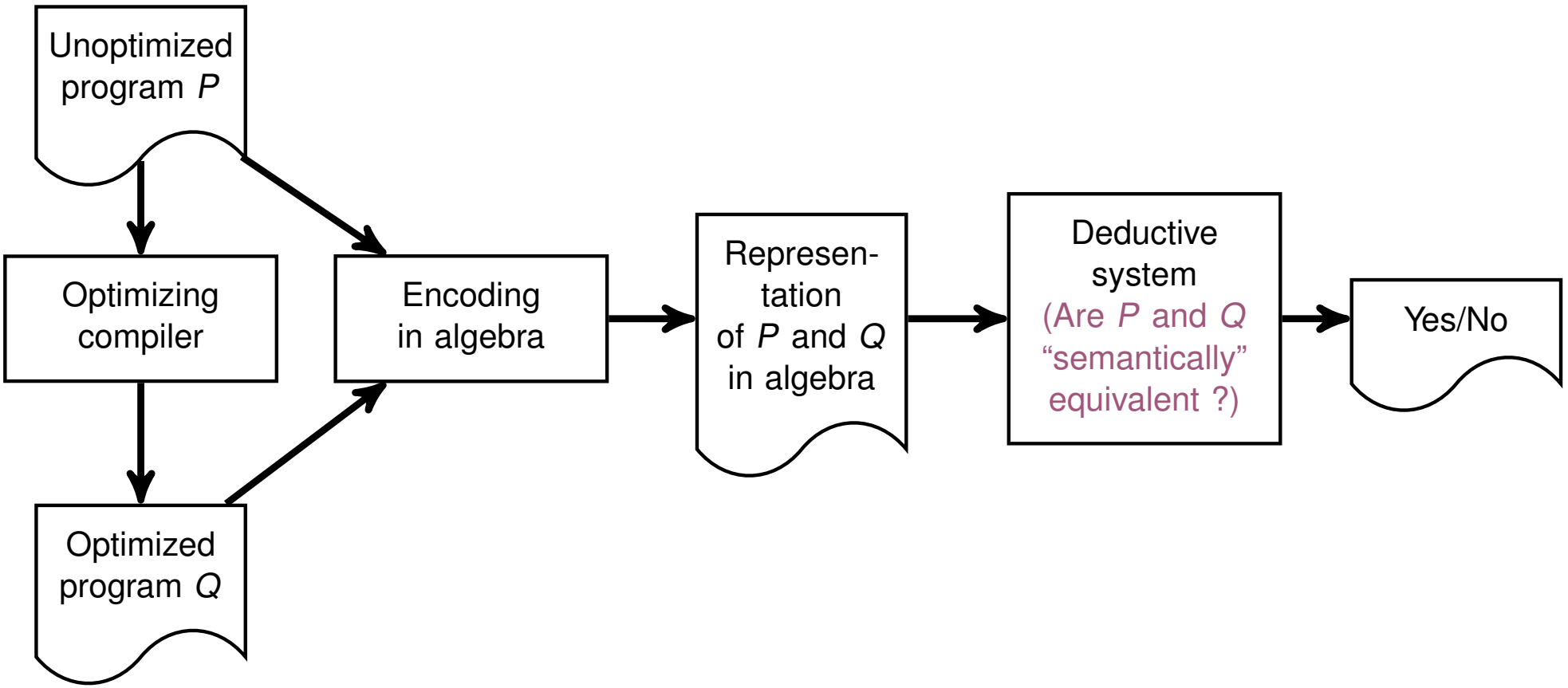
- Let a represent the test that the address of x exists.
- Creation of *x*: $s = s \cdot a$
- $*n$ += 1 modifies neither the test n != NULL nor the existence of x: $a \cdot b \cdot q = q \cdot a \cdot b$
- x is passed to the function increment by a pointer:

$$a \cdot {}^{\langle}f = {}^{\langle}f \cdot a \cdot b$$

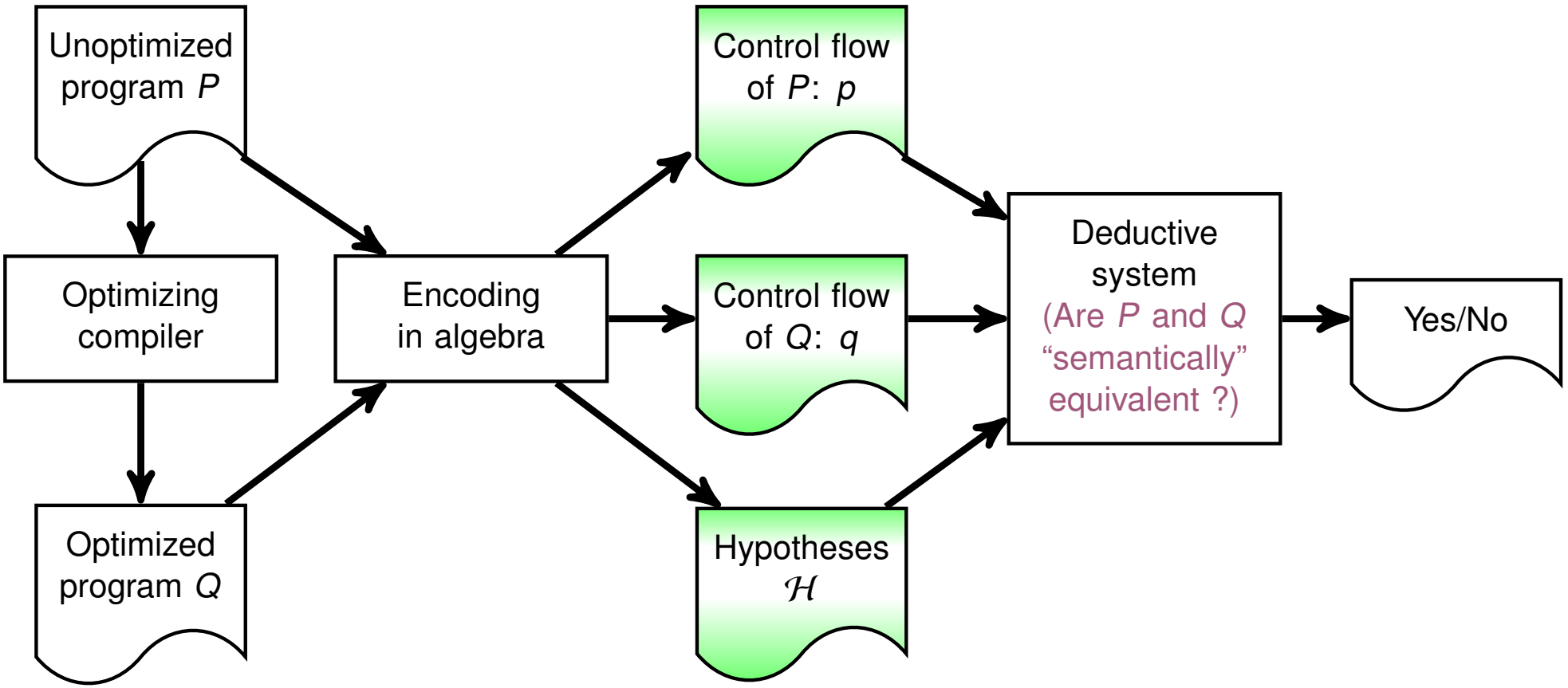
and

$$a \cdot b \cdot f^{\rangle} = f^{\rangle} \cdot a \ .$$

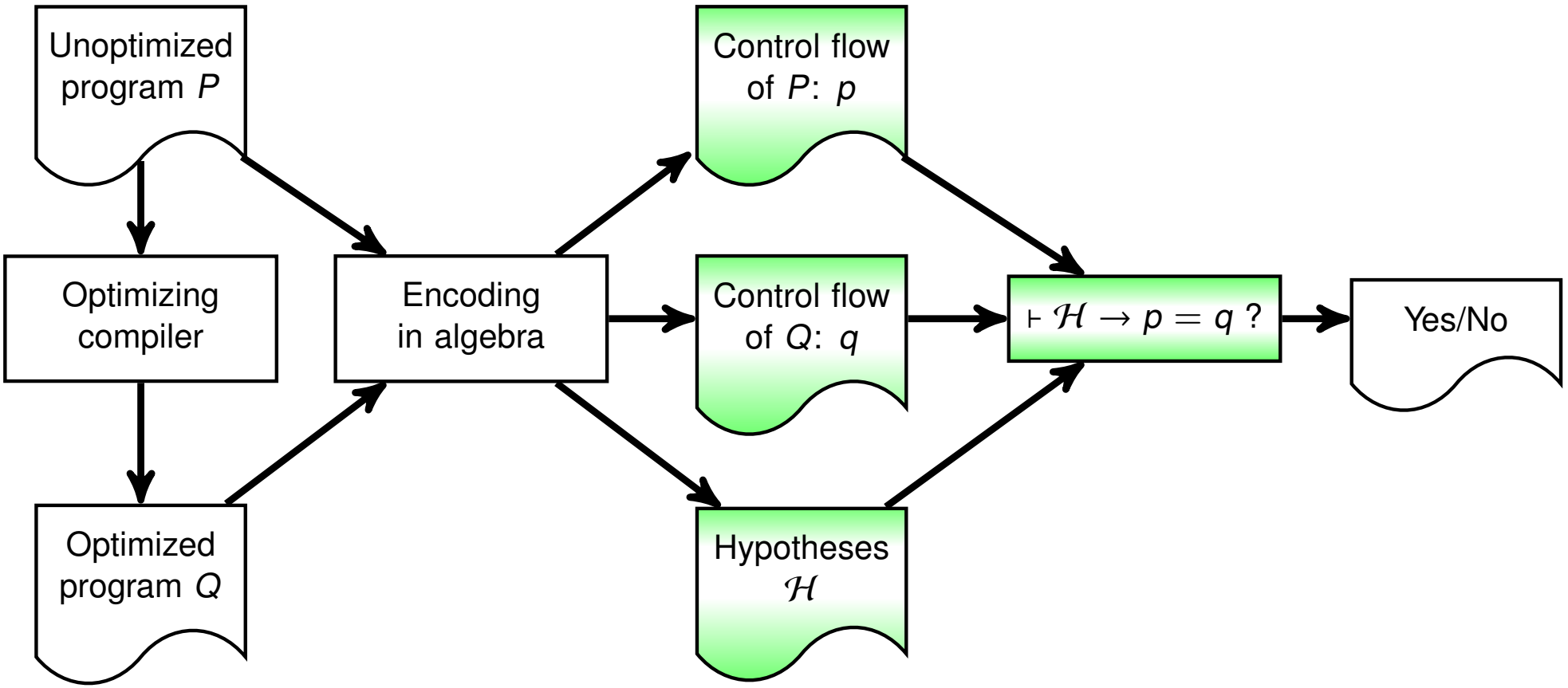
Framework for the Verification in VPKA



Framework for the Verification in VPKA



Framework for the Verification in VPKA



Verification of the Example

We have to prove that, under the preceding hypotheses,

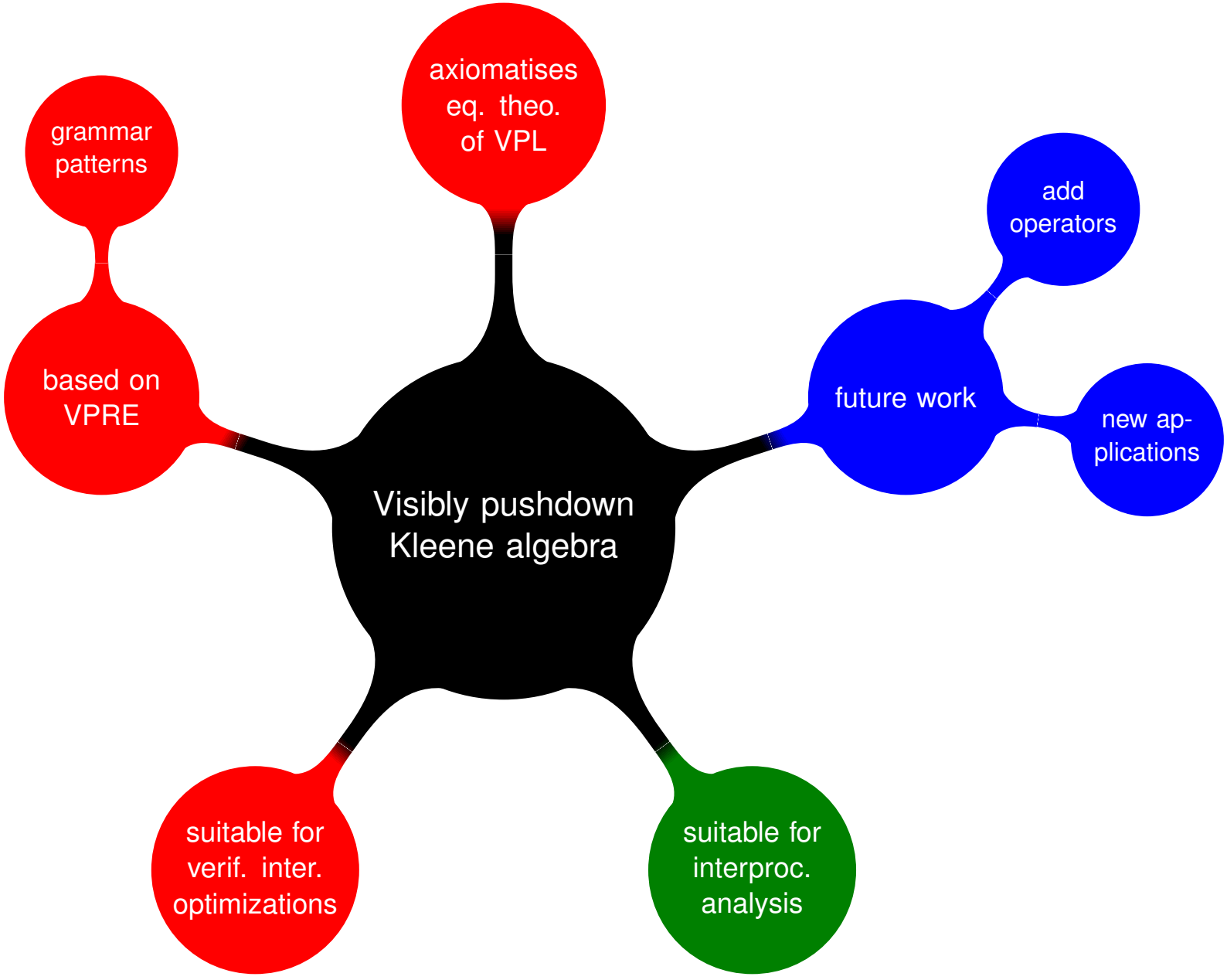
$$\begin{aligned} & \left(\!\!\left| \underset{m'}{\left[\underset{m'}{\langle m \rangle} \right]^{\tau}}, \underset{m}{\left[s \cdot \langle f \rangle \cdot \langle f \rangle \cdot t \right]^{\tau}}, \underset{f}{\left[\overline{b} \cdot p + b \cdot q \right]^{\tau}} \right|\!\!\right)^{\tau} \\ = & \\ & \left(\!\!\left| \underset{m'}{\left[\underset{m'}{\langle m \rangle} \right]^{\tau}}, \underset{m}{\left[s \cdot \langle f \rangle \cdot \langle f \rangle \cdot t \right]^{\tau}}, \underset{f}{\left[q \right]^{\tau}} \right|\!\!\right)^{\tau} . \end{aligned}$$

Verification of the Example (Quick Proof)

$$
\begin{aligned}
& \left(\!\!\left(\underset{m'}{[} \langle m \rangle \underset{m'}{]^{\tau}}, \underset{m}{[} s \cdot \langle f \rangle \cdot \langle f \rangle \cdot t \underset{m}{]^{\tau}}, \underset{f}{[} \overline{b} \cdot p + b \cdot q \underset{f}{]^{\tau}} \right)\!\!\right)^{\tau} \\
= & \quad \{\text{Metablock \& VPKA axioms}\} \\
& \langle m \cdot \left(\!\!\left(\underset{m}{[} \langle m \rangle \underset{m'}{]^{\tau}}, \underset{m}{[} s \cdot \langle f \rangle \cdot \langle f \rangle \cdot t \underset{m}{]^{\tau}}, \underset{f}{[} \overline{b} \cdot p + b \cdot q \underset{f}{]^{\tau}} \right)\!\!\right)^{\tau} \cdot m \rangle \\
= & \quad \{\text{Metablocks \& VPKA axioms}\} \\
& \langle m \cdot s \cdot \langle f \cdot \left(\!\!\left(\underset{f}{[} \langle m \rangle \underset{m'}{]^{\tau}}, \underset{m}{[} s \cdot \langle f \rangle \cdot \langle f \rangle \cdot t \underset{m}{]^{\tau}}, \underset{f}{[} \overline{b} \cdot p + b \cdot q \underset{f}{]^{\tau}} \right)\!\!\right)^{\tau} \cdot f \rangle \\
& \qquad \cdot \langle f \cdot \left(\!\!\left(\underset{f}{[} \langle m \rangle \underset{m'}{]^{\tau}}, \underset{m}{[} s \cdot \langle f \rangle \cdot \langle f \rangle \cdot t \underset{m}{]^{\tau}}, \underset{f}{[} \overline{b} \cdot p + b \cdot q \underset{f}{]^{\tau}} \right)\!\!\right)^{\tau} \cdot f \rangle \cdot t \cdot m \rangle \\
= & \quad \{\text{Metablocks \& VPKA axioms}\} \\
& \langle m \cdot s \cdot \langle f \cdot (\overline{b} \cdot p + b \cdot q) \cdot f \rangle \cdot \langle f \cdot (\overline{b} \cdot p + b \cdot q) \cdot f \rangle \cdot t \cdot m \rangle \\
= & \quad \{\text{Hypothesis: } s = s \cdot a\} \\
& \langle m \cdot s \cdot a \cdot \langle f \cdot (\overline{b} \cdot p + b \cdot q) \cdot f \rangle \cdot \langle f \cdot (\overline{b} \cdot p + b \cdot q) \cdot f \rangle \cdot t \cdot m \rangle \\
= & \quad \{\text{Hypotheses \& Kleene algebra}\} \\
& \langle m \cdot s \cdot \langle f \cdot q \cdot f \rangle \cdot \langle f \cdot q \cdot f \rangle \cdot t \cdot m \rangle \\
= & \quad \{\text{Metablocks \& VPKA axioms}\} \\
& \left(\!\!\left(\underset{m'}{[} \langle m \rangle \underset{m'}{]^{\tau}}, \underset{m}{[} s \cdot \langle f \rangle \cdot \langle f \rangle \cdot t \underset{m}{]^{\tau}}, \underset{f}{[} q \underset{f}{]^{\tau}} \right)\!\!\right)^{\tau}
\end{aligned}
$$

Outline

Conclusion


axiomatises eq. theo. of VPL
grammar patterns
based on VPRE
Visibly pushdown Kleene algebra
future work
add operators
new ap- plications
suitable for verif. inter. optimizations
suitable for interproc. analysis

Thanks !

Do you have any questions ?

FAQ Materials

Representation of Context-Free Languages

Implicit Rewrite Rules

FAQ Materials

Representation of Context-Free Languages

Implicit Rewrite Rules

Related Work on the Representation of (a Subset of) Context-Free Languages

General idea: Directly use the least fixed point operator μ.

Leiß (1991): Kleene algebra with recursion (3 levels of context-free languages).

Bloom and Ésik (1990-present): Iteration theories (they develop a robust theory for fixed point operators).

Remark: Too expressive for visibly pushdown languages (must restrict the μ-expressions allowed which is not easy).

Other interesting work:

Leiß (2006): Use Kleene modules to represent the linear context-free languages.

Remark: Incomparable with visibly pushdown languages.

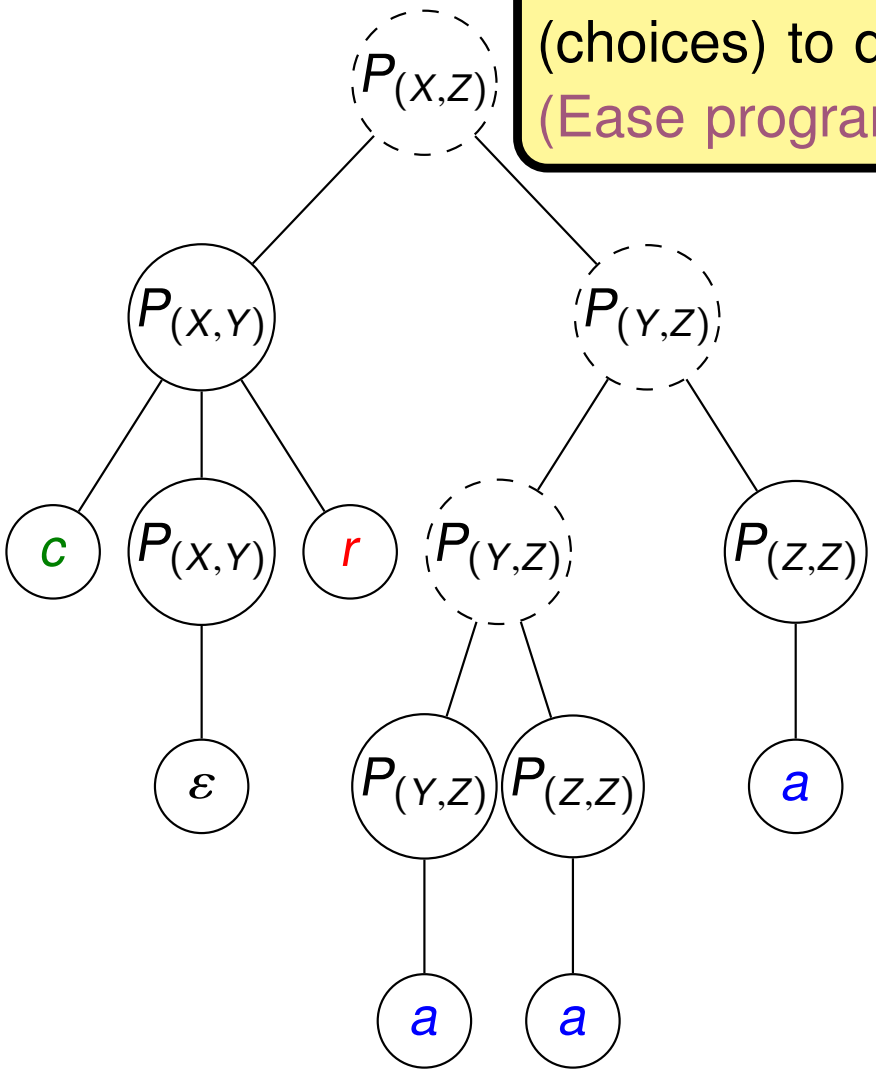
FAQ Materials

Representation of Context-Free Languages

Implicit Rewrite Rules

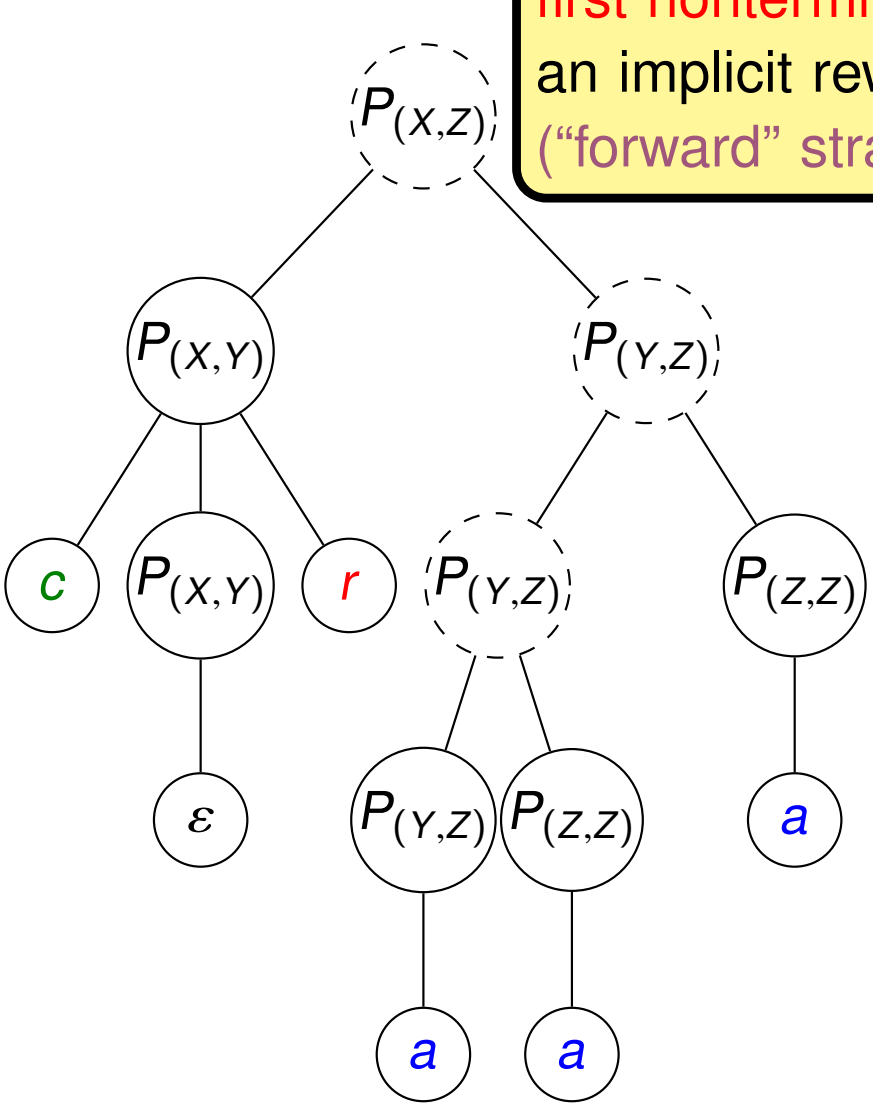
Remark: Flexibility Given by the Implicit Rewrite Rules

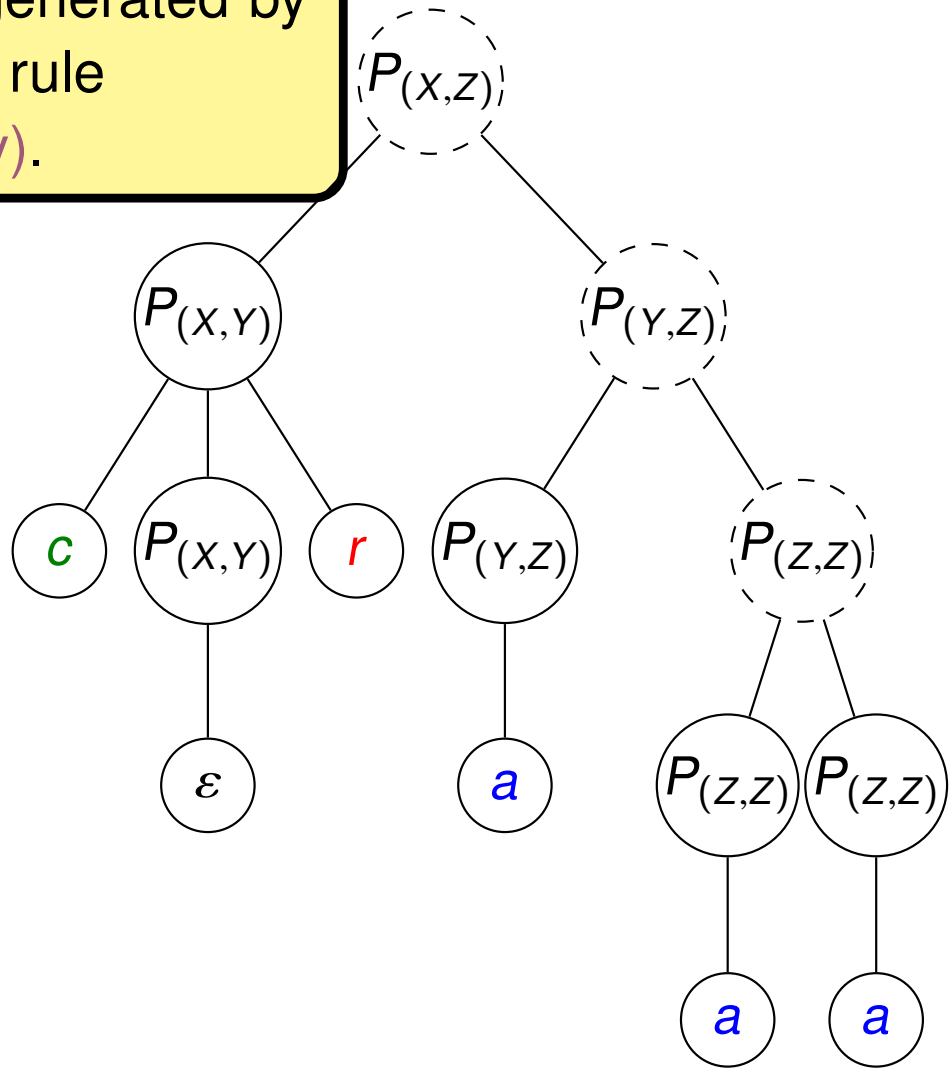
Allow multiple strategies (choices) to derive a word.
(Ease program manipulation.)



Remark: Flexib Rewrite Rules

Example: Always use an explicit rewrite rule to derive the first nonterminal generated by an implicit rewrite rule (“forward” strategy).

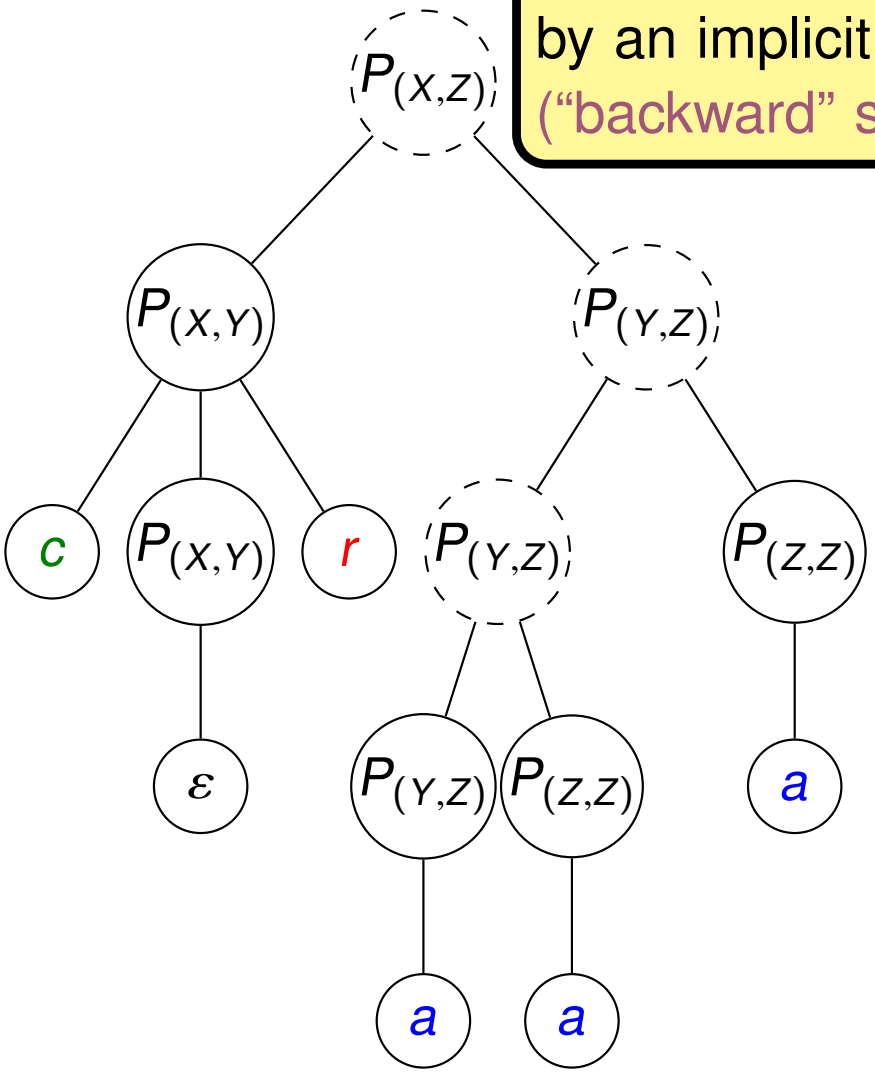


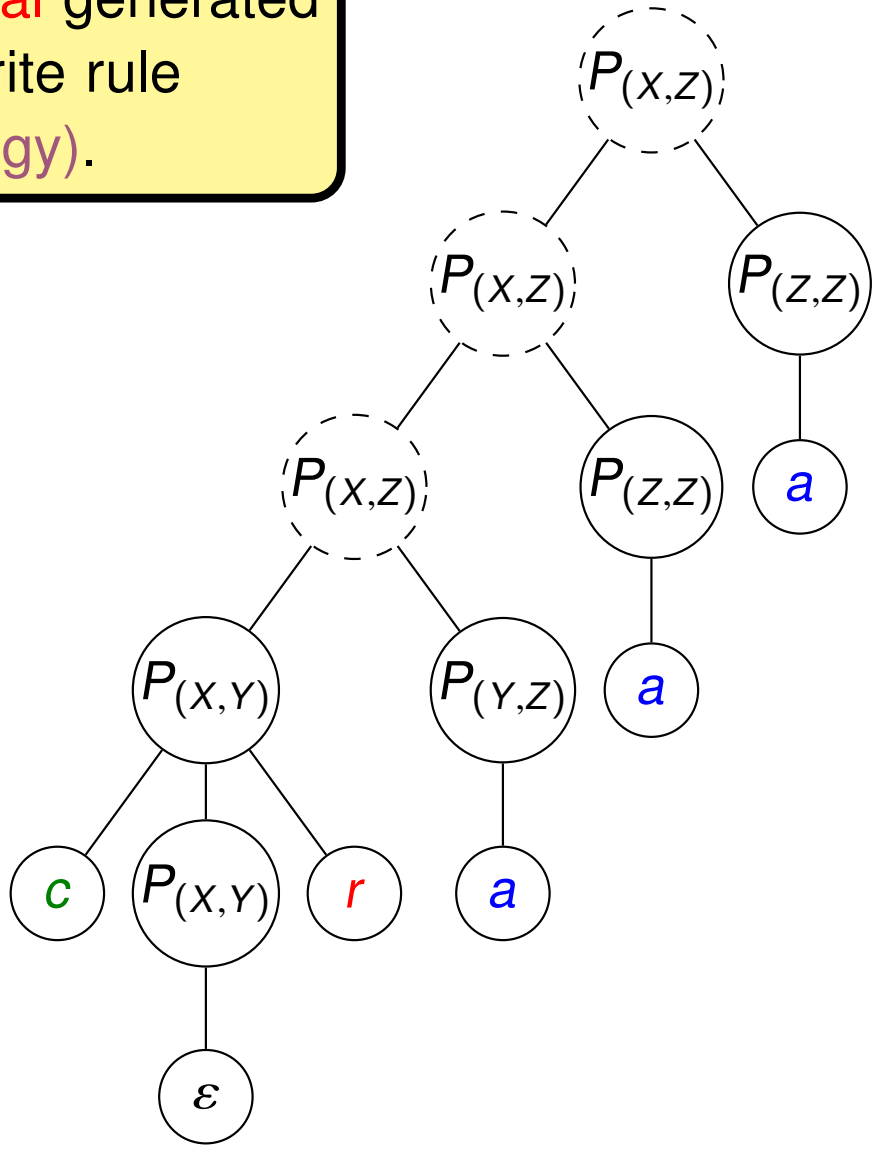


UNIVERSITÉ LAVAL

Remark: Flexib Rewrite Rules

Example: Always use an explicit rewrite rule to derive the second nonterminal generated by an implicit rewrite rule ("backward" strategy).





Remark: Flexibility Given by the Implicit Rewrite Rules

When reasoning algebraically, it is sometimes useful to fix the derivation strategy.