Flowchart Programs, Regular Expressions, and Decidability of Polynomial Growth-rate

Amir M. Ben-Amram Aviad Pineles

The Academic College of Tel-Aviv Yaffo

Flowchart Programs, Regular Expressions, and Decidability of Polynomial Growth-rate

A VERIFICATION PROBLEM:
Given a program **p**, does **p** run in polynomial time?

This study focuses on decidability
in weak programming languages

Amir M. Ben-Amram   Aviad Pineles

The Academic College of Tel-Aviv Yaffo

Flowchart Programs, Regular Expressions, and Decidability of Polynomial Growth-rate

We extend previous results to a new, stronger language by means of a program transformation

Amir M. Ben-Amram Aviad Pineles

The Academic College of Tel-Aviv Yaffo

Outline

- A known decidability result
- Flowchart programs
- The transformational technique

"Complexity analysis" is an important program analysis challenge

- Wegbreit 1975: analysing the complexity of LISP programs.

```
REVERSE(L) ≡
if NULL(L) then {} else
APPEND(REVERSE(CDR(
L)), CONS(CAR(L), {}))
```

$$\Longrightarrow \quad c_0 + c_1 \cdot n + c_2 \cdot n^2$$

- Many current (or recent) projects, e.g. COSTA, *Safe*, SPEED, AProVE..
- These are real tools, not decidability results

Simple "loop programs" have been studied wrt decidability

Kasai & Adachi, JCSS 1980
Kristiansen and Niggl, TCS 2004
Niggl & Wunderlich, SICOMP 2006
Jones & Kristiansen, TOCL 2009
Ben-Amram, Jones & Kristiansen, CiE 2008
Ben-Amram, DICE 2010

Focus on identifying a complexity class (polynomial time?)

- "Simple" decision problem
- Influence from the field of ICC (Implicit Comp. Complexity)

Simple "loop programs" have been studied wrt decidability

In these programs loops are explicitly bounded

do X times { ... }	not	while B do { ... }

Asking about time complexity, etc. is equivalent to asking about the growth rate of variables

Hence "the polynomial growth-rate problem"

BJK 2008

Polynomial growth rate is decidable for the language:

$$e \in \text{Expression} ::= X \mid e + e \mid e * e$$

$$\begin{aligned} C \in \text{Command} ::= {} & X := e \\ \mid {} & C_1 ; C_2 \\ \mid {} & \text{loop } X \, \{C\} \\ \mid {} & \text{choose } C_1 \text{ or } C_2 \end{aligned}$$

Moreover, the analysis algorithm is PTIME

BJK 2008

Decidability comes from restrictions of the language

$$e \in \text{Expression} ::= X \mid e + e \mid e * e$$

$$\begin{aligned} C \in \text{Command} ::= {} & X := e \\ \mid {} & C_1 ; C_2 \\ \mid {} & \text{loop } X \, \{C\} \\ \mid {} & \text{choose } C_1 \text{ or } C_2 \end{aligned}$$

BJK 2008

Decidability comes from restrictions of the language

$$e \in \text{Expression} ::= X \quad \boxed{e + e \mid e * e}$$

Restricted arithmetics

$$C \in \text{Command} ::= X := e$$
$$\mid C_1 \,;\, C_2$$
$$\mid \text{loop } X \,\{C\}$$
$$\mid \text{choose } C_1 \text{ or } C_2$$

BJK 2008

Decidability comes from restrictions of the language

$e \in \text{Expression} ::= X \mid e + e \mid e * e$

Restricted arithmetics

$C \in \text{Command} ::= X := e$

$\mid C_1 ; C_2$

$\mid \boxed{\text{loop } X \{C\}}$

Explicitly bounded loops:
At most X iterations

$\mid \text{choose } C_1 \text{ or } C_2$

BJK 2008

Decidability comes from restrictions of the language

$$e \in \text{Expression} ::= X \mid e + e \mid e * e$$

Restricted arithmetics

$$\begin{aligned} C \in \text{Command} ::= {} & X := e \\ \mid {} & C_1 ; C_2 \\ \mid {} & \text{loop } X \{C\} \\ \mid {} & \boxed{\text{choose } C_1 \text{ or } C_2} \end{aligned}$$

Explicitly bounded loops:
At most X iterations

Non-deterministic branching

Compositionality important: algorithm a bottom-up computation of "growth-rate assertions"

$$\frac{C1 \vdash X \xrightarrow{\text{linear}} Y \qquad C2 \vdash Y \xrightarrow{\text{poly.}} Z}{C1\,;\,C2 \vdash X \xrightarrow{\text{poly.}} Z}$$

Compositionality important: algorithm a bottom-up computation of "growth-rate assertions"

$$\frac{C1 \vdash X \xrightarrow{\text{linear}} Y \qquad C2 \vdash Y \xrightarrow{\text{poly.}} Z}{C1\,;\,C2 \vdash X \xrightarrow{\text{poly.}} Z}$$

The crux of the result is deduction rules for loops

- When we know what the loop body does

  we can derive the effect of iterating it

Subsequent research explores decidability in extensions of the language

Even small extensions can make the problem undecidable

$$e \in \text{Expression} ::= X \mid X{+}Y \mid X{*}Y$$

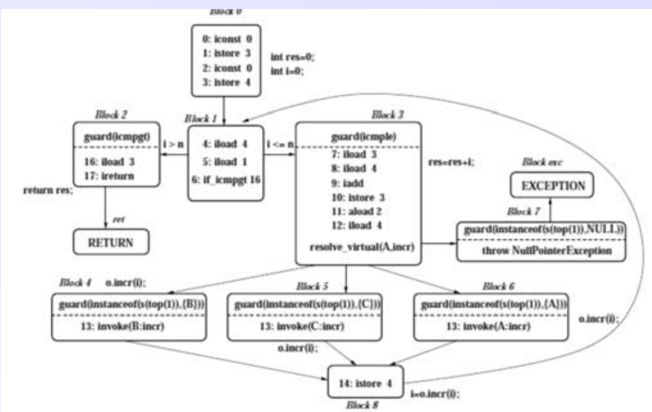
$$\begin{aligned} C \in \text{Command} ::= {} & X := e \\ \mid {} & C_1 \,;\, C_2 \\ \mid {} & \text{loop } X \, \{C\} \\ \mid {} & \text{if } X{=}Y \text{ then } C_1 \text{ else } C_2 \end{aligned}$$

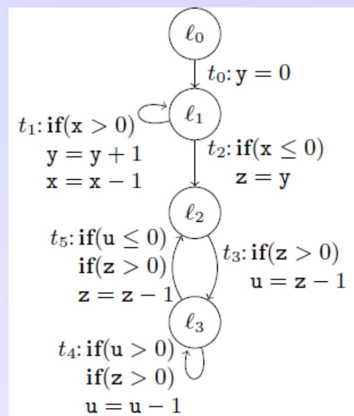
Outline

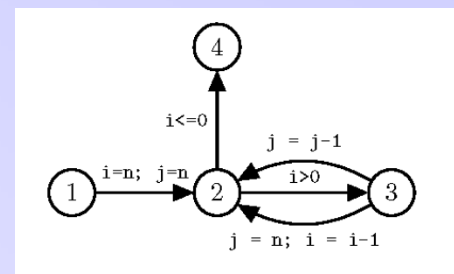
- A known decidability result
- Flowchart programs
- The transformational technique

Flowchart programs motivation

- Practical work often uses “flowchart programs”
  - JBC, LLVM, ad-hoc representations

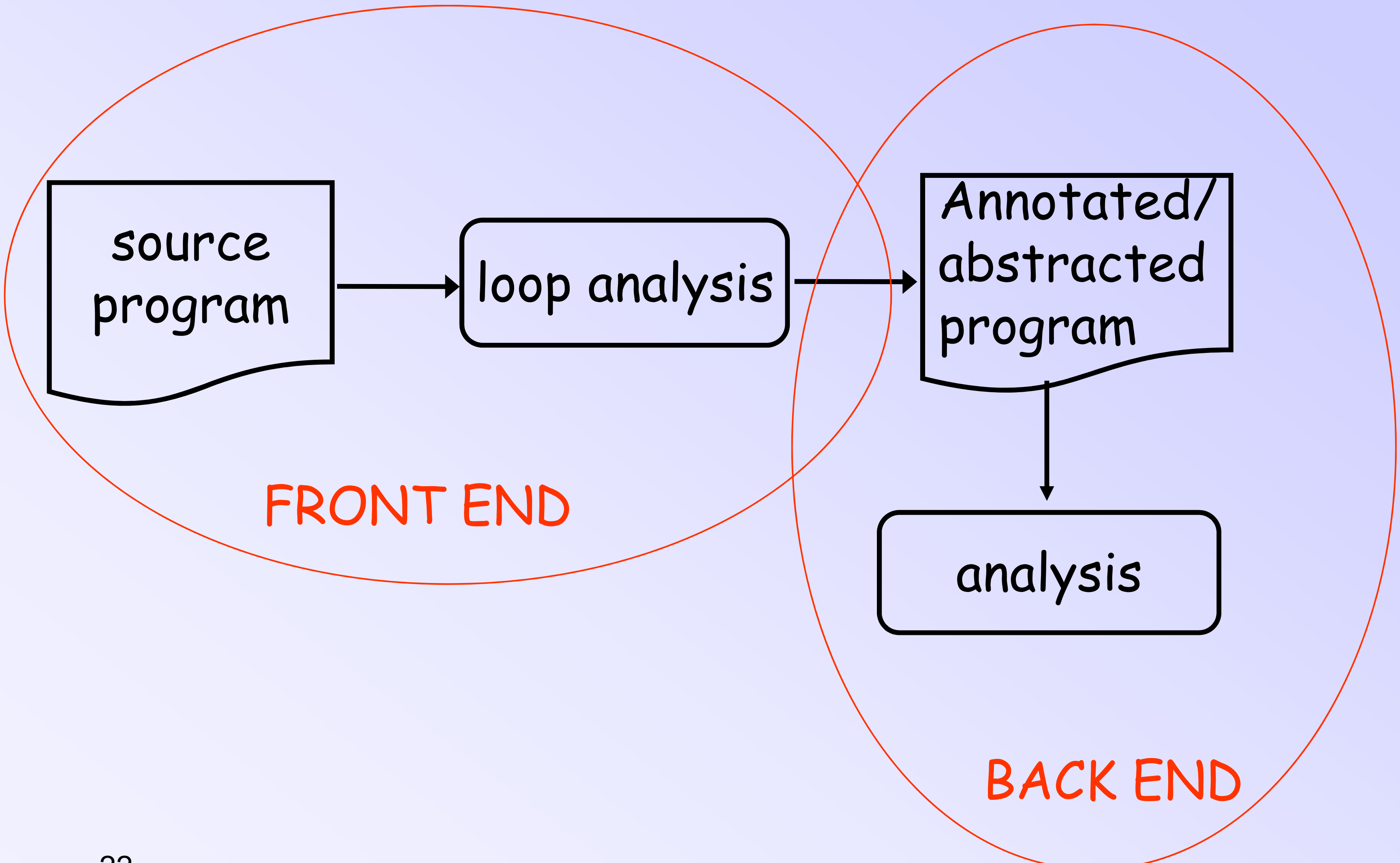




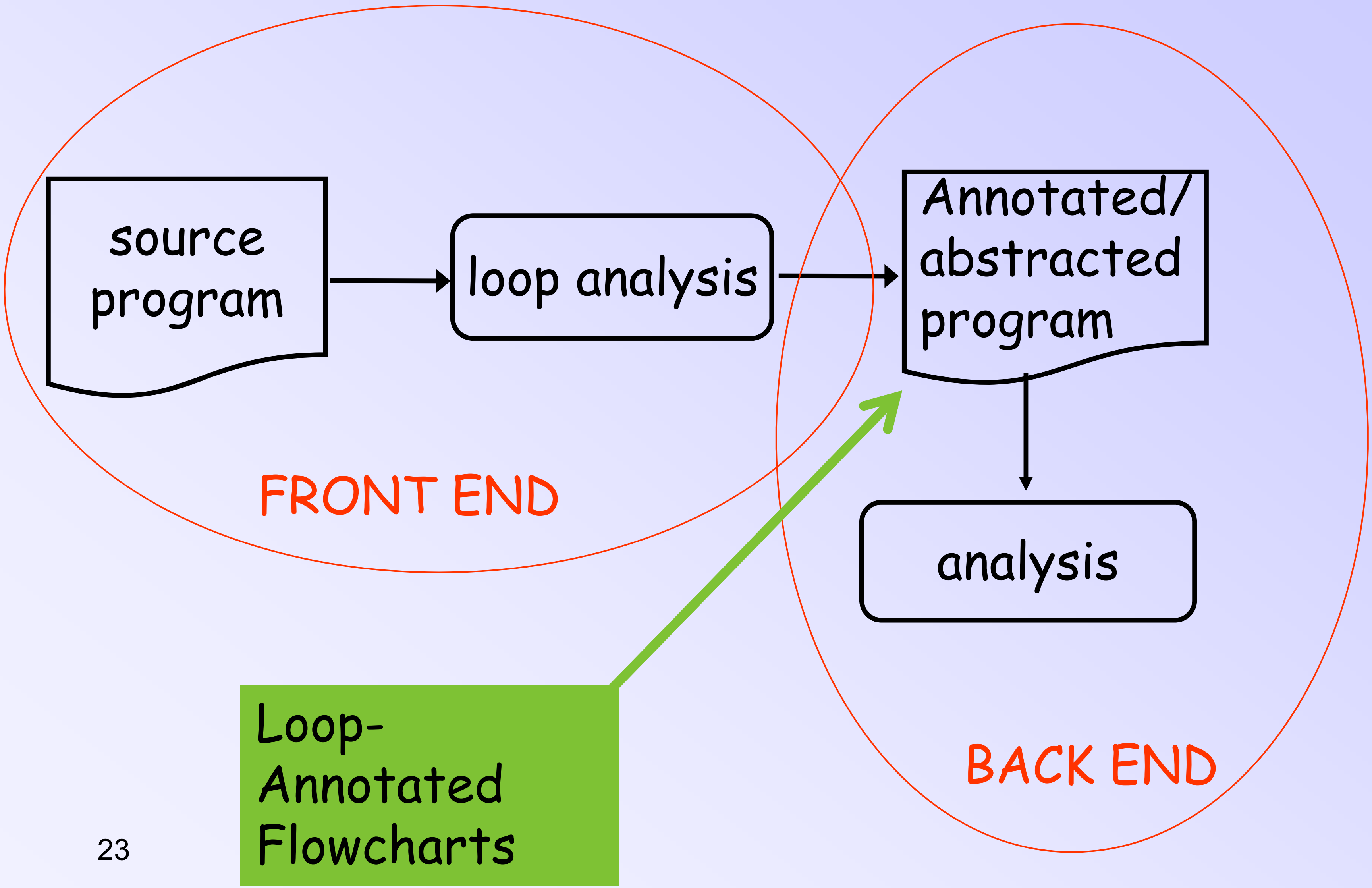


- A challenge to our methods: programs not compositional, loops not explicitly bounded, and are managed using operations that we cannot hope to handle

An “abstract and conquer” setting

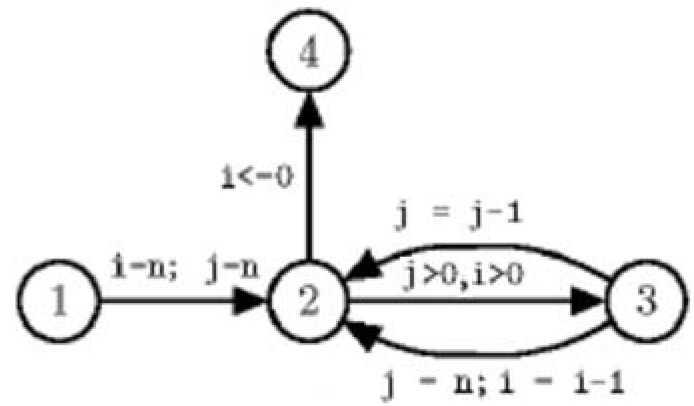


An “abstract and conquer” setting



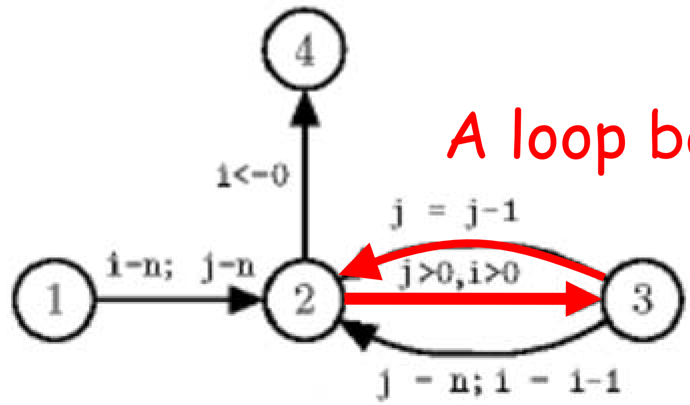
Our program model is inspired by the work of Alias et al. (2010)

```
assume (n>=0);
i = n;
j = n;
while (i > 0) {
  if (j>0) {
    j = j-1;
  } else {
    j = n;
    i = i-1;
  }
}
```



Our program model is inspired by the work of Alias et al. (2010)

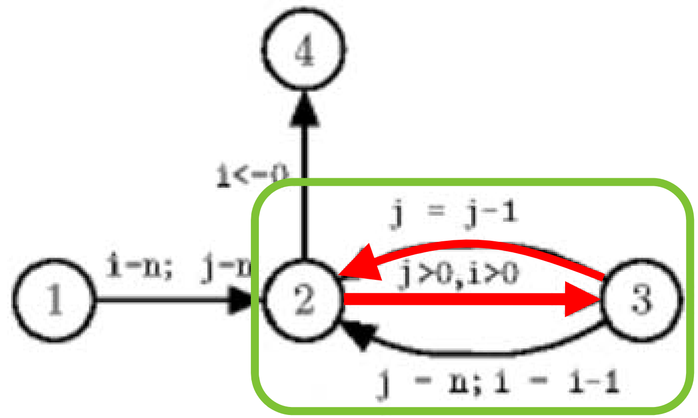
```
assume (n>=0);
i = n;
j = n;
while (i > 0) {
   if (j>0) {
      j = j-1;
   } else {
      j = n;
      i = i-1;
   }
}
```



A loop bounded by j

Our program model is inspired by the work of Alias et al. (2010)

```
assume (n>=0);
i = n;
j = n;
while (i > 0) {
   if (j>0) {
      j = j-1;
   } else {
      j = n;
      i = i-1;
   }
}
```

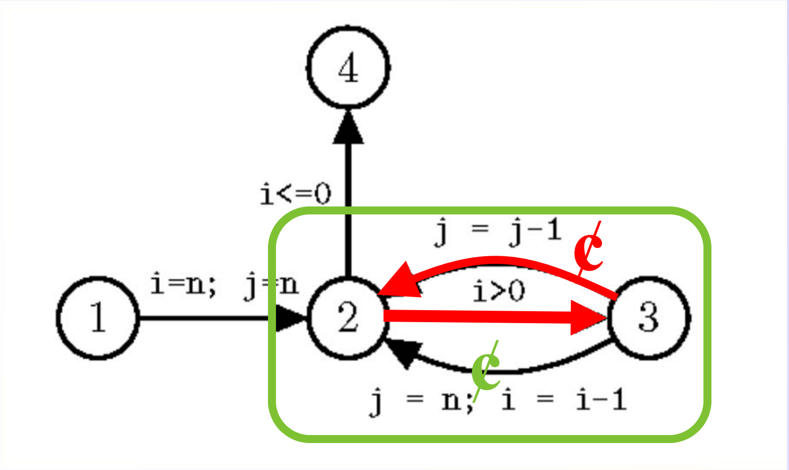


An outer loop bounded by i

A loop-annotated flowchart consists of

- A flowchart over our restricted instruction set
- A set of (well-nested) subgraphs called loops. Every subset has a bound.

The bound controls the number of traversals of certain cut-arcs



THM. The polynomial growth-rate problem for annotated flowcharts is decidable (PTIME)

- The technique: a transformation to a structured language which extends the BJK language
  - An automata-theoretic argument shows that flowcharts are more expressive than original BJK

- Key intuition to solution:

flowchart ~ automaton

loop program ~ regular expression

Loop-Annotated Regular Expressions (LARE)

built of instructions, concatenation $e_1 e_2$, alternation $e_1|e_2$, looping e^* and loop annotations

$[_{X1} (\; [_{X2} (\; \boxed{¢} \; \boxed{X3 := X2+X3} \;)^*] \; \boxed{¢} \; \boxed{X2 := X1*X1} \;)^*]$

```
loop X1 {
        loop X2 { X3 := X2+X3 } ;
        X2 := X1*X1
}
```

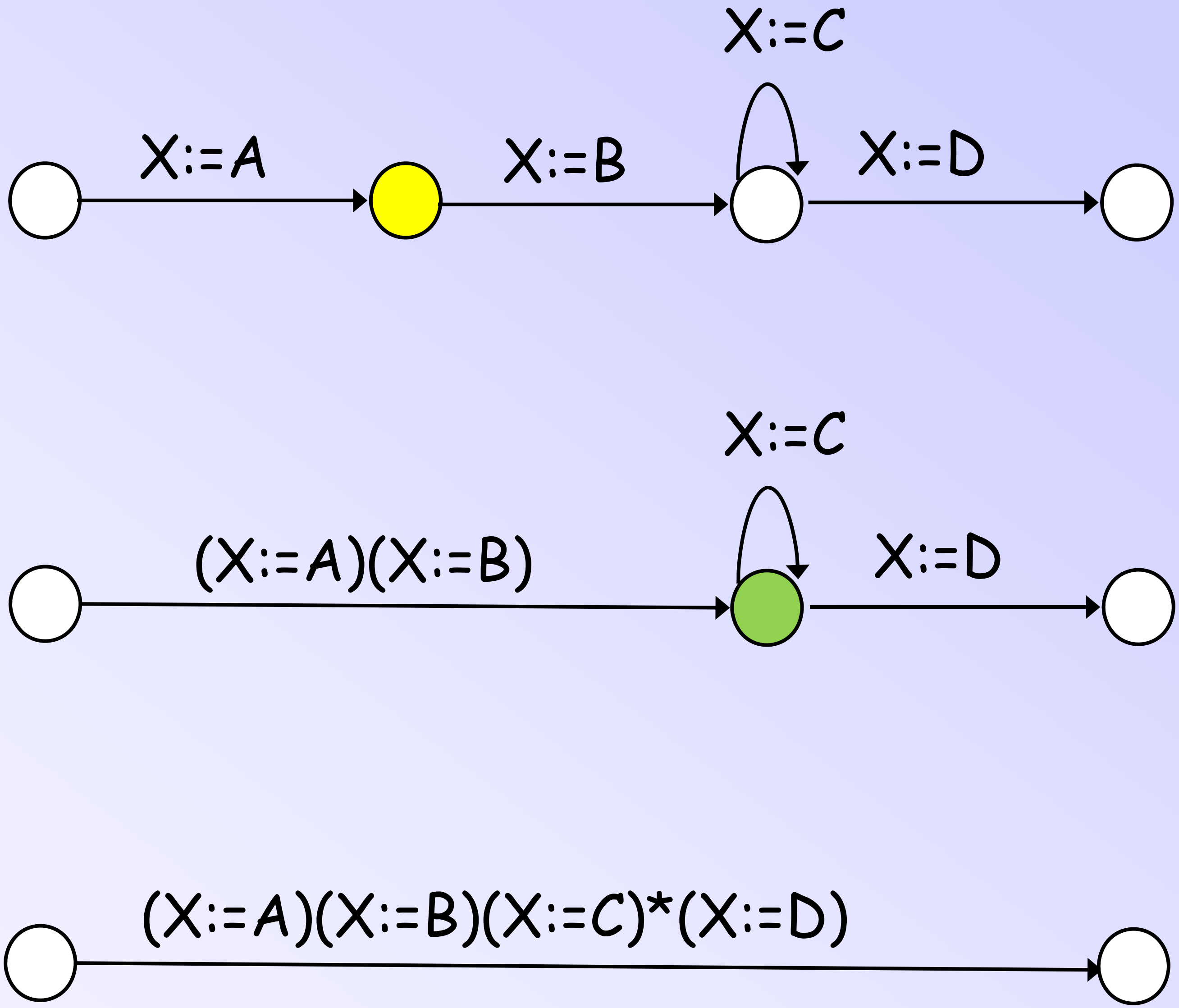
Loop-Annotated Regular Expressions (LARE)

built of instructions, concatenation $e_1 e_2$,
alternation $e_1|e_2$, looping e^* and loop annotations

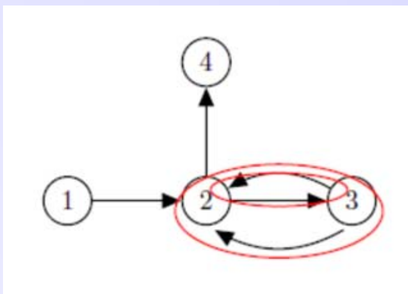
$[_{X1}\ (\ [_{X2}\ (\ \boxed{\not c}\ \boxed{X3 := X2+X3}\ )^*\ ]\ \boxed{\not c}\ \boxed{X2 := X1 * X1}\ )^*\ ]$

The analysis of [BJK] is easily adapted to LARE.

We transform flowchart programs to LARE using, essentially, a textbook algorithm (NFA->regexp)


X:=C
X:=A
X:=B
X:=D
X:=C
(X:=A)(X:=B)
X:=D
(X:=A)(X:=B)(X:=C)*(X:=D)

An apparent difficulty: **complexity.**

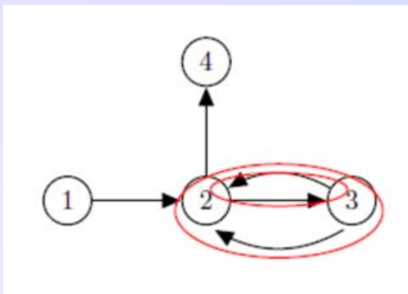


$$[_4(\not{c}(\boxed{X_3 := X_1 + X_2}\,\boxed{X_2 := X_3})\,)^*]\;[_4(\not{c}(\boxed{X_2 := X_1 + X_4} \mid \boxed{X_2 := X_2 + X_4})\,)^*]$$

exponential growth

Ehrenfeucht & Zeiger, 1974:
"Some of our colleagues have considered using a regular expression … that represents a program … the question of how large or complex one might expect such an expression to be naturally arose"

Our solution: eliminate the explicit construction of the expersssion.

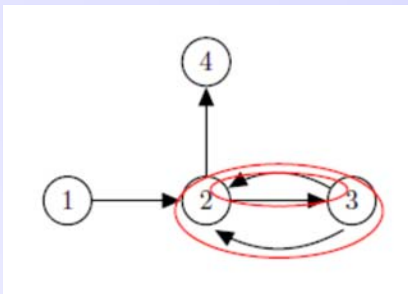


$$[_4(\text{¢}(\boxed{X_3 := X_1 + X_2}\ \boxed{X_2 := X_3}))^*]\ [_4(\text{¢}(\boxed{X_2 := X_1 + X_4} \mid \boxed{X_2 := X_2 + X_4}))^*]$$

$$C1 \vdash X \xrightarrow{\text{lin}} Y \qquad C2 \vdash Y \xrightarrow{\text{poly}} Z$$

$$C1 ; C2 \vdash X \xrightarrow{\text{poly.}} Z$$

Our solution: eliminate the explicit construction of the experssion ("function fusion")



$C1 \mid\!- \; X \xrightarrow{\text{lin}} Y$ $\qquad$ $C2 \mid\!- \; Y \xrightarrow{\text{poly}} Z$

$$C1 \,;\, C2 \;\; \mid\!- \;\; X \xrightarrow{\text{poly.}} Z$$

Conclusion

- Loop-annotated flowcharts allowed us to extend the techniques of [BJK '08] to a more complex language, motivated by practice
- The technique may be worth noting
  - turn a flowchart program into a structured one for analysis by the NFA->regexp transform. Then eliminate the regexp
- Open problems regarding the growth-rate problem:
  - Extend the instruction set
  - Generate tight upper bounds

Thank you for your attention