Welcome!

On Resource Analysis of Imperative Programs

Lars Kristiansen

Department of Mathematics, University of Oslo

Department of Informatics, University of Oslo

Resource Analysis

ABSTRACT

The execution of a computer program requires resources – time and space. To what extent is it possible to estimate a program's resource requirements by analyzing the program code? Resource analysis has been a fairly popular research area the recent years. I started my research on resource analysis of imperative programs about 13 years ago, and I published my last paper on the subject about a year ago. I will try to share some of the insights I have gained during these years.

The talk will not be very technical, and any computer scientist familiar with elementary complexity theory should be able to follow.

Resource Analysis

The summer of 2000: I tried to understand Bellantoni-Cook recursion.
I worked with with simple imperative programming languages and studied examples like . . .

Resource Analysis

The summer of 2000: I tried to understand Bellantoni-Cook recursion.
I worked with with simple imperative programming languages and studied examples like . . .

```
Y:=1; Z:=1;
loop X { Z:=0; loop Y { Z:=Z+2 } Y:=Z }
```

Resource Analysis

The summer of 2000: I tried to understand Bellantoni-Cook recursion.
I worked with with simple imperative programming languages and studied examples like ...

```
Y:=1; Z:=1;
loop X { Z:=0; loop Y { Z:=Z+2 } Y:=Z }
```

and

```
Y:=1; Z:=0;
loop X { loop Y { Z:=Z+2 } Y:=Y+1 }
```

Resource Analysis

Observation I

If

> *no loop has a circles in its data-flow graph*

then

> *the computed values will be bounded by polynomials in the input (and thus the program run in polynomial time).*

Resource Analysis

Observation I

If

no loop has a circles in its data-flow graph

then

the computed values will be bounded by polynomials in the input (and thus the program run in polynomial time).

Observation II

"circles" can be detected by an algorithm.

Resource Analysis

Observation I

If

no loop has a circles in its data-flow graph

then

the computed values will be bounded by polynomials in the input (and thus the program run in polynomial time).

Observation II

"circles" can be detected by an algorithm.

I wrote a paper (with Niggl) based on these observations (TCS, 2004).

Resource Analysis

In this paper we studied a rudimentary imperative programming language manipulating stacks:

Resource Analysis

In this paper we studied a rudimentary imperative programming language manipulating stacks:

- `push(a,stack)` `pop(stack)` `nil(stack)`

Resource Analysis

In this paper we studied a rudimentary imperative programming language manipulating stacks:

- `push(a,stack)`   `pop(stack)`   `nil(stack)`
- `P;Q`

Resource Analysis

In this paper we studied a rudimentary imperative programming language manipulating stacks:

- `push(a,stack)` `pop(stack)` `nil(stack)`
- `P;Q`
- `if top(stack)=symbol then { P }`

Resource Analysis

In this paper we studied a rudimentary imperative programming language manipulating stacks:

- `push(a,stack)`   `pop(stack)`   `nil(stack)`
- `P;Q`
- `if top(stack)=symbol then { P }`
- `for each element on stack do { P }`

Resource Analysis

We defined something we called a *measure*. Our measure was a computable function

$$\nu \;:\; \text{programs} \to \mathbb{N}$$

Resource Analysis

We defined something we called a *measure*. Our measure was a computable function

$$\nu \ : \ \text{programs} \rightarrow \mathbb{N}$$

with the property

$\nu(\mathtt{P}) = 0$ if and only if

no loop of $\mathtt{P}$ has a "circle" in its data-flow graph.

Resource Analysis

We defined something we called a *measure*. Our measure was a computable function

$$\nu \ : \ \text{programs} \to \mathbb{N}$$

with the property

$\nu(\mathtt{P}) = 0$ if and only if

no loop of $\mathtt{P}$ has a "circle" in its data-flow graph.

Theorem. A function is computable in polynomial time if, and only if, it can be computed by a program with ν-measure 0.

Resource Analysis

So, we characterized the functions computable in polynomial time.

Resource Analysis

So, we characterized the functions computable in polynomial time.

We also characterized the Grzegorczyk class $\mathcal{E}^{i+3}$ as the functions computable by programs with ν-measure $i+1$.

Resource Analysis

So, we characterized the functions computable in polynomial time.

We also characterized the Grzegorczyk class $\mathcal{E}^{i+3}$ as the functions computable by programs with ν-measure $i+1$.

Now, ...why characterize complexity classes?

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)
- Bellantoni & Cook's characterization of *Ptime* (1992, the first so-called *implicit characterization*)

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)
- Bellantoni & Cook's characterization of *Ptime* (1992, the first so-called *implicit characterization*)
- ... we have characterizations based on proof calculi,

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)
- Bellantoni & Cook's characterization of *Ptime* (1992, the first so-called *implicit characterization*)
- ... we have characterizations based on proof calculi, rewriting systems,

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)
- Bellantoni & Cook's characterization of *Ptime* (1992, the first so-called *implicit characterization*)
- ... we have characterizations based on proof calculi, rewriting systems, linear logic, modal logic,

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)
- Bellantoni & Cook's characterization of *Ptime* (1992, the first so-called *implicit characterization*)
- ... we have characterizations based on proof calculi, rewriting systems, linear logic, modal logic, programming languages, digital circuits

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)
- Bellantoni & Cook's characterization of *Ptime* (1992, the first so-called *implicit characterization*)
- ... we have characterizations based on proof calculi, rewriting systems, linear logic, modal logic, programming languages, digital circuits ... THIS IS A BIG INDUSTRY ...

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)
- Bellantoni & Cook's characterization of *Ptime* (1992, the first so-called *implicit characterization*)
- ... we have characterizations based on proof calculi, rewriting systems, linear logic, modal logic, programming languages, digital circuits ... THIS IS A BIG INDUSTRY ... I am responsible for quite a few such characterizations myself ...

Resource Analysis

Let me remind you of a few characterizations . . .

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)
- Bellantoni & Cook's characterization of *Ptime* (1992, the first so-called *implicit characterization*)
- . . . we have characterizations based on proof calculi, rewriting systems, linear logic, modal logic, programming languages, digital circuits . . . THIS IS A BIG INDUSTRY . . . I am responsible for quite a few such characterizations myself . . . and I bet I can say the same about others in this room . . .

Resource Analysis

Let me remind you of a few characterizations ...

- Ritchie's characterization of *LINSPACE* (1962)
- Cobham's characterization of *Ptime* (1965)
- Bellantoni & Cook's characterization of *Ptime* (1992, the first so-called *implicit characterization*)
- ... we have characterizations based on proof calculi, rewriting systems, linear logic, modal logic, programming languages, digital circuits ... THIS IS A BIG INDUSTRY ... I am responsible for quite a few such characterizations myself ... and I bet I can say the same about others in this room ...

Haven't we seen enough such characterizations by now? Why do we indulge in this business?

Resource Analysis

There is several (good) motivations for characterizing complexity classes, e.g.

Resource Analysis

There is several (good) motivations for characterizing complexity classes, e.g.

- to show that the complexity classes are natural mathematical entities not depending on a particular machine model

Resource Analysis

There is several (good) motivations for characterizing complexity classes, e.g.

- to show that the complexity classes are natural mathematical entities not depending on a particular machine model
- to understand the complexity classes better and shed light upon some of the (notorious hard) open problems of complexity theory

Resource Analysis

There is several (good) motivations for characterizing complexity classes, e.g.

- to show that the complexity classes are natural mathematical entities not depending on a particular machine model
- to understand the complexity classes better and shed light upon some of the (notorious hard) open problems of complexity theory
- to understand the computational power of constructions in programming languages

Resource Analysis

There is several (good) motivations for characterizing complexity classes, e.g.

- to show that the complexity classes are natural mathematical entities not depending on a particular machine model
- to understand the complexity classes better and shed light upon some of the (notorious hard) open problems of complexity theory
- to understand the computational power of constructions in programming languages
- . . . etc.

Resource Analysis

There is several (good) motivations for characterizing complexity classes, e.g.

- to show that the complexity classes are natural mathematical entities not depending on a particular machine model
- to understand the complexity classes better and shed light upon some of the (notorious hard) open problems of complexity theory
- to understand the computational power of constructions in programming languages
- ... etc.

However, in my and Niggl's case I felt it was rather pointless to characterize *Ptime*, and yet we did! (This was the theorem that the editor and the referee wanted.)

Resource Analysis

The most essential part of the paper was not the characterization of *Ptime*. . . or the characterization of the Grzegorczyk classes
. . . BUT

Resource Analysis

The most essential part of the paper was not the characterization of *Ptime*. . . or the characterization of the Grzegorczyk classes
. . . BUT

- the data-flow analysis
- the computational method based on this analysis.

Resource Analysis

The most essential part of the paper was not the characterization of *Ptime*. . . or the characterization of the Grzegorczyk classes . . . BUT

- the data-flow analysis
- the computational method based on this analysis.

This was a method for estimating the computational complexity of programs.

Resource Analysis

The most essential part of the paper was not the characterization of *Ptime*. . . or the characterization of the Grzegorczyk classes . . . BUT

- the data-flow analysis
- the computational method based on this analysis.

This was a method for estimating the computational complexity of programs.

The method was never explicated in the paper . . . just buried down in proofs and theorems.

Resource Analysis

We had programs

$$P_0\ ,\ P_1\ ,\ P_2\ ,\ P_3\ ,\ \ldots$$

written in a rudimentary but natural programming language. Some programs run in polynomial time – some do not.

Resource Analysis

We had programs

$$P_0\ ,\ P_1\ ,\ P_2\ ,\ P_3\ ,\ \dots$$

written in a rudimentary but natural programming language. Some programs run in polynomial time – some do not.

We had a computational method

$$\mathcal{M}\ :\ \text{programs}\ \to\ \{\text{yes}, \text{no}\}$$

such that

$$\{P_i \mid \mathcal{M}(P_i) = \text{yes}\}\ \subset\ \{P_i \mid P_i \text{ runs in polynomial time}\}\ .$$

Resource Analysis

We had programs

$$P_0 \ , \ P_1 \ , \ P_2 \ , \ P_3 \ , \ \ldots$$

written in a rudimentary but natural programming language. Some programs run in polynomial time – some do not.

We had a computational method

$$\mathcal{M} \ : \ \text{programs} \ \rightarrow \ \{\text{yes}, \text{no}\}$$

such that

$$\{P_i \mid \mathcal{M}(P_i) = \text{yes}\} \ \subset \ \{P_i \mid P_i \text{ runs in polynomial time}\} \ .$$

I noted two things!

Resource Analysis

On the one hand the set

$$\{P_i \mid P_i \text{ runs in polynomial time}\}$$

is undecidable.

Thus, it is impossible to find a computational method $\mathcal{M}'$ such that

$$\{P \mid \mathcal{M}'(P) = \text{yes}\} \;=\; \{P \mid P \text{ runs in polynomial time}\}\ .$$

Resource Analysis

On the other hand, it should be possible to find a computational method $\mathcal{M}'$ such that

$$\begin{aligned}\{P \mid \mathcal{M}(P) = \text{yes}\} \;\subset\; & \{P \mid \mathcal{M}'(P) = \text{yes}\} \\ \subset\; & \{P \mid P \text{ runs in polynomial time}\}\end{aligned}$$

and

Resource Analysis

On the other hand, it should be possible to find a computational method $\mathcal{M}'$ such that

$$\begin{aligned}\{P \mid \mathcal{M}(P) = \text{yes}\} &\subset \{P \mid \mathcal{M}'(P) = \text{yes}\} \\ &\subset \{P \mid P \text{ runs in polynomial time}\}\end{aligned}$$

and

$\{P \mid \mathcal{M}'(P) = \text{yes}\}$ contains far more natural programs than $\{P \mid \mathcal{M}(P) = \text{yes}\}$.

Resource Analysis

- I could extend the language . . .

Resource Analysis

- I could extend the language . . .

```
X:=Y   X:=X+(Z*Z)   if <test> then ...  else ....
```

Resource Analysis

- I could extend the language . . .

  `X:=Y   X:=X+(Z*Z)   if <test> then ...  else ....`

- I could refine the data-flow analysis.

Resource Analysis

- I could extend the language ...

```
X:=Y   X:=X+(Z*Z)   if <test> then ...   else ....
```

- I could refine the data-flow analysis.

  I identified three types of flow:

  - *m*-flow
  - *w*-flow
  - *p*-flow

Resource Analysis

- I could extend the language . . .

```
X:=Y   X:=X+(Z*Z)   if <test> then ...  else ....
```

- I could refine the data-flow analysis.

  I identified three types of flow:

  - *m*-flow
  - *w*-flow
  - *p*-flow

Terminology: *Harmful* things are things that cause a program to compute a value not bounded by a polynomial in the input. (Thus, if a program does not do harmful things, it will run i polynomial time.) *Harmless* things are things that are not harmful.

Resource Analysis

m-flow is harmless
(when it occurs isolated from other types of flow).

Resource Analysis

m-flow is harmless
(when it occurs isolated from other types of flow).

This is an example of *m*-flow:

```
loop U { X:=Y; Y:=Z; Z:=X }.
```

Resource Analysis

m-flow is harmless
(when it occurs isolated from other types of flow).

This is an example of *m*-flow:

```
loop U { X:=Y; Y:=Z; Z:=X } .
```

- no need to put restrictions on *m*-flow
- "circles" may very well occur.

Resource Analysis

Both w-flow and p-flow might be harmful.

Resource Analysis

Both w-flow and p-flow might be harmful.

A "circle" of such flow (inside a loop) will cause a program to compute a value not bounded by a polynomial in the input.

Resource Analysis

Both *w*-flow and *p*-flow might be harmful.

A “circle” of such flow (inside a loop) will cause a program to compute a value not bounded by a polynomial in the input.

This is an example of *w*-flow:

```
loop Z { X:=Y+Y; Y:=X }.
```

Resource Analysis

Both *w*-flow and *p*-flow might be harmful.

A "circle" of such flow (inside a loop) will cause a program to compute a value not bounded by a polynomial in the input.

This is an example of *w*-flow:

```
loop Z { X:=Y+Y; Y:=X }.
```

To avoid harmful things, we must put restrictions on *w*-flow and *p*-flow.

Resource Analysis

- *w*-flow is iteration-*in*dependent
- *p*-flow is iteration-dependent.

Resource Analysis

- *w*-flow is iteration-*in*dependent
- *p*-flow is iteration-dependent.

```
loop X { Y:= U + V }
```

The value computed into `Y` does not depend on the number of times the loop's body is iterated.

Resource Analysis

- *w*-flow is iteration-*in*dependent
- *p*-flow is iteration-dependent.

```
loop X { Y:= U + V }
```

The value computed into `Y` does not depend on the number of times the loop's body is iterated.

```
loop X { Y:= Y + Z }
```

The value computed into `Y` depends on the number of times the loop's body is iterated.

Resource Analysis

- *w*-flow is iteration-*in*dependent
- *p*-flow is iteration-dependent.

```
loop X { Y:= U + V }
```

The value computed into Y does not depend on the number of times the loop's body is iterated.

```
loop X { Y:= Y + Z }
```

The value computed into Y depends on the number of times the loop's body is iterated.

w-flow and *p*-flow make it possible to distinguish between these to situations.

Resource Analysis

I wrote a paper (with Neil Jones) based on these ideas (CiE 2005 and ACM Transactions of Computational Logic 10 2009).

Resource Analysis

I wrote a paper (with Neil Jones) based on these ideas (CiE 2005 and ACM Transactions of Computational Logic 10 2009).

We developed a syntactic proof calculus for assigning *mwp*-flow graphs to programs.

The flow graphs were represented matrices.

Resource Analysis

I wrote a paper (with Neil Jones) based on these ideas (CiE 2005 and ACM Transactions of Computational Logic 10 2009).

We developed a syntactic proof calculus for assigning *mwp*-flow graphs to programs.

The flow graphs were represented matrices.

Here is some of our inference rules:

Resource Analysis

$$\vdash \texttt{skip} : \mathbf{1} \qquad \frac{\vdash \texttt{e} : V}{\vdash \texttt{X}_j \texttt{:=e} : \mathbf{1} \overset{j}{\leftarrow} V}$$

$$\frac{\vdash \texttt{C}_1 : A \qquad \vdash \texttt{C}_2 : B}{\vdash \texttt{C}_1 \texttt{;} \texttt{C}_2 : A \otimes B} \qquad \frac{\vdash \texttt{C}_1 : A \qquad \vdash \texttt{C}_2 : B}{\vdash \texttt{if b then } \texttt{C}_1 \texttt{ else } \texttt{C}_2 : A \oplus B}$$

$$\frac{\vdash \texttt{C} : M}{\vdash \texttt{loop } \texttt{X}_\ell \, \{\texttt{C}\} : M^* \oplus \{ \overset{p}{\ell} \!\!\rightarrow j \mid \exists\, i \, [M^*_{ij} = p] \}} \; (\text{if } \forall i [M^*_{ii} = m])$$

Resource Analysis

Here comes a derivation:

$$
\dfrac{\dfrac{\dfrac{\vdash \mathtt{X}_1 : \begin{pmatrix} m \\ 0 \\ 0 \end{pmatrix} \qquad \vdash \mathtt{X}_1 : \begin{pmatrix} m \\ 0 \\ 0 \end{pmatrix}}{\vdash \mathtt{X}_1 \mathtt{+} \mathtt{X}_1 : \begin{pmatrix} p \\ 0 \\ 0 \end{pmatrix}}}{\vdash \mathtt{X}_2 \mathtt{:=} \mathtt{X}_1 \mathtt{+} \mathtt{X}_1 : \begin{pmatrix} m & p & 0 \\ 0 & 0 & 0 \\ 0 & 0 & m \end{pmatrix}}}{\vdash \mathtt{loop}\, \mathtt{X}_3\, \{\mathtt{X}_2 \mathtt{:=} \mathtt{X}_1 \mathtt{+} \mathtt{X}_1\} : \begin{pmatrix} m & p & 0 \\ 0 & m & 0 \\ 0 & p & m \end{pmatrix}}
$$

Don't try to understand this stuff!

Resource Analysis

Here is (a part of) another derivation:

$$
\dfrac{
  \dfrac{
    \dfrac{\vdash \mathtt{X}_2\mathtt{+}\mathtt{X}_3 : \begin{pmatrix} 0 \\ w \\ w \\ 0 \end{pmatrix}}
    {\vdash \mathtt{X}_1 \mathtt{:=} \mathtt{X}_2\mathtt{+}\mathtt{X}_3 : \begin{pmatrix} 0 & 0 & 0 & 0 \\ w & m & 0 & 0 \\ w & 0 & m & 0 \\ 0 & 0 & 0 & m \end{pmatrix}}
    \qquad
    \dfrac{\dfrac{\vdots}{\vdash \mathtt{X}_2\mathtt{+}\mathtt{X}_4 : \begin{pmatrix} 0 \\ m \\ 0 \\ p \end{pmatrix}}}
    {\vdash \mathtt{X}_2 \mathtt{:=} \mathtt{X}_2\mathtt{+}\mathtt{X}_4 : \begin{pmatrix} m & 0 & 0 & 0 \\ 0 & m & 0 & 0 \\ 0 & 0 & m & 0 \\ 0 & p & 0 & m \end{pmatrix}}
  }
  {\vdash \mathtt{X}_1 \mathtt{:=} \mathtt{X}_2\mathtt{+}\mathtt{X}_3 \mathtt{;}\ \mathtt{X}_2 \mathtt{:=} \mathtt{X}_2\mathtt{+}\mathtt{X}_4 : \begin{pmatrix} 0 & 0 & 0 & 0 \\ w & m & 0 & 0 \\ w & 0 & m & 0 \\ 0 & p & 0 & m \end{pmatrix}}
  \qquad
  \dfrac{\vdash \mathtt{X}}{\vdash \mathtt{X}_3 \mathtt{:=} \mathtt{X}_4\mathtt{+}}
}
{\vdash \mathtt{X}_1 \mathtt{:=} \mathtt{X}_2\mathtt{+}\mathtt{X}_3 \mathtt{;}\ \mathtt{X}_2 \mathtt{:=} \mathtt{X}_2\mathtt{+}\mathtt{X}_4 \mathtt{;}\ \mathtt{X}_3 \mathtt{:=} \mathtt{X}_4\mathtt{+}\mathtt{X}_4 : \begin{pmatrix} 0 & 0 & 0 & 0 \\ w & m & 0 & 0 \\ w & 0 & m & 0 \\ 0 & p & p & m \end{pmatrix}}
$$

Resource Analysis

A program is *derivable* if the calculus assigns a matrix to the program.

Resource Analysis

A program is *derivable* if the calculus assigns a matrix to the program.

Theorem.

$$\mathtt{P} \text{ is derivable} \quad \Rightarrow \quad \mathtt{P} \text{ runs in polynomial time .}$$

Resource Analysis

A program is *derivable* if the calculus assigns a matrix to the program.

Theorem.

$$\mathrm{P} \text{ is derivable } \quad \Rightarrow \quad \mathrm{P} \text{ runs in polynomial time }.$$

This yields a computational method $\mathcal{M}$ such that

$$\mathcal{M}(\mathrm{P}) = \text{yes} \quad \Rightarrow \quad \mathrm{P} \text{ runs in polynomial time }.$$

(The *mwp*-method.)

Resource Analysis

The *mwp*-stuff may seem fancy.

Resource Analysis

The *mwp*-stuff may seem fancy.

There is a heavy mathematical machinery involved:

- a matrix algebra: $A \otimes B \ldots A \oplus B \ldots A^*$
- a proof calculus.

Resource Analysis

The *mwp*-stuff may seem fancy.

There is a heavy mathematical machinery involved:

- a matrix algebra: $A \otimes B \ldots A \oplus B \ldots A^*$
- a proof calculus.

But all this machinery may be good for nothing!

Resource Analysis

The *mwp*-stuff may seem fancy.

There is a heavy mathematical machinery involved:

- a matrix algebra: $A \otimes B \ldots A \oplus B \ldots A^*$
- a proof calculus.

But all this machinery may be good for nothing! How powerful is the *mwp*-method?

Resource Analysis

The *mwp*-stuff may seem fancy.

There is a heavy mathematical machinery involved:

- a matrix algebra: $A \otimes B \ldots A \oplus B \ldots A^*$
- a proof calculus.

But all this machinery may be good for nothing! How powerful is the *mwp*-method?

The soundness result

$$\mathcal{M}(\mathtt{P}) = \text{yes} \quad \Rightarrow \quad \mathtt{P} \text{ runs in polynomial time}$$

does of course not tell us anything in this respect. (If $\mathcal{M}(\mathtt{P}) = \text{no}$ for every $\mathtt{P}$, then the method is sound.)

Resource analysis

How can we argue that such a method is powerful?

Resource analysis

How can we argue that such a method is powerful?

(I) *Empirical experiments:* Apply the method to large quanta of programs, and then ... (this is not my cup of tea).

Resource analysis

How can we argue that such a method is powerful?

(I) *Empirical experiments:* Apply the method to large quanta of programs, and then ... (this is not my cup of tea).

(II) *Argue by (paradigm) examples:* Hey, my method can deal with this well-known sorting algorithm whereas yours cannot... (we will at least have lively and passionate discussions)

Resource analysis

How can we argue that such a method is powerful?

(I) *Empirical experiments:* Apply the method to large quanta of programs, and then ... (this is not my cup of tea).

(II) *Argue by (paradigm) examples:* Hey, my method can deal with this well-known sorting algorithm whereas yours cannot... (we will at least have lively and passionate discussions)

(III) *Characterize* Ptime *(and other complexity classes):* Prove that for any $f \in Ptime$ there exists a program $\mathtt{P}$ such that

$$\mathcal{M}(\mathtt{P}) = \text{yes and } \mathtt{P} \text{ computes } f.$$

Resource analysis

How can we argue that such a method is powerful?

(I) *Empirical experiments:* Apply the method to large quanta of programs, and then ... (this is not my cup of tea).

(II) *Argue by (paradigm) examples:* Hey, my method can deal with this well-known sorting algorithm whereas yours cannot... (we will at least have lively and passionate discussions)

(III) *Characterize* Ptime *(and other complexity classes):* Prove that for any $f \in \mathit{Ptime}$ there exists a program $\mathtt{P}$ such that

$$\mathcal{M}(\mathtt{P}) = \text{yes and } \mathtt{P} \text{ computes } f.$$

(This tells next to nothing, ... it tells that your method can recognize a program that simulates a polynomially clocked Turing machine. It would be very strange if your method couldn't.)

Resource analysis

Luckily, we found a fourth way to argue!

Resource analysis

Luckily, we found a fourth way to argue!

This is the situation from a more general point of view:

- we want to find an algorithm for solving a problem $\mathcal{A}$
- but we cannot – and we know that cannot – because we know that $\mathcal{A}$ is undecidable.

Resource analysis

Luckily, we found a fourth way to argue!

This is the situation from a more general point of view:

- we want to find an algorithm for solving a problem $\mathcal{A}$
- but we cannot – and we know that cannot – because we know that $\mathcal{A}$ is undecidable.

Recall that a problem is the same as a language. So

$$\mathcal{A} \quad = \quad \{\mathtt{P} \mid \mathtt{P} \text{ runs in polynomial time}\} \ .$$

Resource analysis

An obvious thing to do is

- to find a subset $\mathcal{A}'$ of $\mathcal{A}$ that in some sense is a good approximation of $\mathcal{A}$
- and then try to find an algorithm for solving $\mathcal{A}'$.

Resource analysis

An obvious thing to do is

- to find a subset $\mathcal{A}'$ of $\mathcal{A}$ that in some sense is a good approximation of $\mathcal{A}$
- and then try to find an algorithm for solving $\mathcal{A}'$.

If we do not succeed

- try to prove that also $\mathcal{A}'$ is undecidable

Resource analysis

An obvious thing to do is

- to find a subset $\mathcal{A}'$ of $\mathcal{A}$ that in some sense is a good approximation of $\mathcal{A}$
- and then try to find an algorithm for solving $\mathcal{A}'$.

If we do not succeed

- try to prove that also $\mathcal{A}'$ is undecidable

If we succeed

- try to find a better approximation $\mathcal{A}''$
- determine the computational complexity of $\mathcal{A}'$. (*P*, *NP*-complete, *PSPACE*, ...? The better approximation $\mathcal{A}''$ will be of a least as high complexity as $\mathcal{A}'$)

Resource analysis

This seems like a reasonable (and rather obvious) way to proceed...

- a way that will make us wiser and yield some insight
- a way that may give us some mathematical tools for arguing that our method for estimating the computational complexity of programs works well

Resource analysis

This seems like a reasonable (and rather obvious) way to proceed...

- a way that will make us wiser and yield some insight
- a way that may give us some mathematical tools for arguing that our method for estimating the computational complexity of programs works well

But how do we find good approximations?

(This is the big question.)

Resource analysis

We want our approximations to the set

$$\{\mathrm{P} \mid \mathrm{P} \text{ runs in polynomial time}\}$$

to contain as many natural algorithm as possible.

We do not want approximations where we e.g. have imposed restrictions on the nesting depth loops . . . or something like that . . . in general, we do not want restrictions that exclude programs that programmers actually would like to write.

Resource analysis

Here is how we (me, Neil Jones and Amir Ben-Amram) found some good approximations.

Resource analysis

Here is how we (me, Neil Jones and Amir Ben-Amram) found some good approximations.

We weakened the semantics such that

if

> *P runs in polynomial time under the weak semantics*

then

> *P runs in polynomial time under the standard semantics.*

Resource Analysis

The set of programs running in polynomial time under the weak semantics turned out to be a decidable set. (But not by the *mwp*-method as I believed.)

Resource Analysis

The set of programs running in polynomial time under the weak semantics turned out to be a decidable set. (But not by the *mwp*-method as I believed.)

Moreover, this set contains fairly many natural programs. Thus, the set is a good approximation to the set of programs running in polynomial time.

Resource Analysis

We worked with the language

- `if <test> then ... else ...`
- `loop X { ... }`
- `X:=` expression in `*`, `+`, `X Y Z`... (variables, but no constants)

The standard semantics is obvious.

Resource Analysis

We worked with the language

- if <test> then ... else ...
- loop X { ... }
- X:= expression in *, +, X Y Z... (variables, but no constants)

The standard semantics is obvious. The nonstandard semantics is

- if <test> then ... else ... : *non-deterministic choice*

Resource Analysis

We worked with the language

- `if <test> then ... else ...`
- `loop X { ... }`
- `X:=` expression in `*`, `+`, `X Y Z`... (variables, but no constants)

The standard semantics is obvious. The nonstandard semantics is

- `if <test> then ... else ...` : *non-deterministic choice*
- `loop X { ... }`: *non-deterministic loop, the loop variable* `X` *gives an upper bound for the number of times the loop will be executed*

Resource Analysis

We worked with the language

- `if <test> then ... else ...`
- `loop X { ... }`
- `X:=` expression in $*$, $+$, `X Y Z`... (variables, but no constants)

The standard semantics is obvious. The nonstandard semantics is

- `if <test> then ... else ...` : *non-deterministic choice*
- `loop X { ... }`: *non-deterministic loop, the loop variable X gives an upper bound for the number of times the loop will be executed*
- the semantics for assignments is the standard one.

Resource Analysis

If

P runs in polynomial time under the non-standard semantics

then

P runs in polynomial time under the standard semantics.

Resource Analysis

If

P runs in polynomial time under the non-standard semantics

then

P runs in polynomial time under the standard semantics.

Why?

Resource Analysis

If

> *P runs in polynomial time under the non-standard semantics*

then

> *P runs in polynomial time under the standard semantics.*

Why? Because the only possible execution under the standard semantics corresponds to *one* of the many possible executions under the non-deterministic semantics.

Resource Analysis

If

> *P runs in polynomial time under the non-standard semantics*

then

> *P runs in polynomial time under the standard semantics.*

Why? Because the only possible execution under the standard semantics corresponds to *one* of the many possible executions under the non-deterministic semantics.

Thus, when every execution of the non-deterministic program runs in polynomial time, the only execution of the deterministic program runs in polynomial time.

Resource Analysis

I believed that the *mwp*-method could decide if a program (in the language given above) runs in polynomial time under the weak semantics. But I could not prove it!

Resource Analysis

I believed that the *mwp*-method could decide if a program (in the language given above) runs in polynomial time under the weak semantics. But I could not prove it!

Amir Ben-Amram cooked up a counterexample showing that my belief was wrong.

Resource Analysis

I believed that the *mwp*-method could decide if a program (in the language given above) runs in polynomial time under the weak semantics. But I could not prove it!

Amir Ben-Amram cooked up a counterexample showing that my belief was wrong.

Note that Amir could give a counterexample because I could state precisely (mathematically) what I believed the *mwp*-method was able to do.

Resource Analysis

Amir's example:

```
loop W  {
        if ?  then
          { Y:= X1; Z:= X2 }
        else
          { Y:= X2; Z:= X1 };
        U:= Y+Z;
        X1:= U
        }
```

EXPLAIN ON THE BLACKBOARD.

Resource Analysis

It turned out that *mwp*-matrices (flow-graphs) did not contain enough information to deal with Amir's example.

Resource Analysis

It turned out that *mwp*-matrices (flow-graphs) did not contain enough information to deal with Amir's example.

A few weeks later we had developed a method that worked perfect. (This method is an extension of the *mwp*-method.) That is, we proved that it is decidable if a program (in the language given above) runs in polynomial time under the weak semantics.

Resource Analysis

It turned out that *mwp*-matrices (flow-graphs) did not contain enough information to deal with Amir's example.

A few weeks later we had developed a method that worked perfect. (This method is an extension of the *mwp*-method.) That is, we proved that it is decidable if a program (in the language given above) runs in polynomial time under the weak semantics.

Moreover, we proved that this decision problem is in P. (This work is published Springer LNCS proceedings from CiE 2008.)

Resource Analysis

We worked with standard assignments

$$\mathtt{X} := \langle exp \rangle$$

where $\langle exp \rangle$ is a arbitrary expression that may contain variables and the operators $+$ and $*$.

Resource Analysis

We worked with standard assignments

$$\mathtt{X} := \langle \mathit{exp} \rangle$$

where $\langle \mathit{exp} \rangle$ is a arbitrary expression that may contain variables and the operators $+$ and $*$.

But no constants were allowed

(or equivalently, constants are allowed, but the weak semantics interprets a constant as an arbitrary number).

Resource Analysis

So programs cannot reset variables: `X:=0`!

Intuitively, it becomes harder to decide if a program runs in polynomial time (under the weak semantics) when programs also can reset variables.

Resource Analysis

Amir carried on the research on his own and proved that our problem still is decidable if we allow the constant 0 (zero) in the programming language.

Resource Analysis

Amir carried on the research on his own and proved that our problem still is decidable if we allow the constant 0 (zero) in the programming language.

But now the problem has become *PSPACE*-complete!

A. M. Ben-Amram: *On decidable growth-rate properties of imperative programs.* (DICE 2010), ed. P. Baillot (volume 23 of EPTCS, ArXiv.org, 2010), pp. 1–14.

Resource Analysis

So, we are doing better and better.

We can deal with better and better approximations to the set

$$\{\mathtt{P} \mid \mathtt{P} \text{ runs in polynomial time}\} \ .$$

Resource Analysis

So, we are doing better and better.

We can deal with better and better approximations to the set

$$\{\mathtt{P} \mid \mathtt{P} \text{ runs in polynomial time}\} \ .$$

Can we do even better? What if we allow the constant 1 in our programming language?

Resource Analysis

So, we are doing better and better.

We can deal with better and better approximations to the set

$$\{\mathtt{P} \mid \mathtt{P} \text{ runs in polynomial time}\} \ .$$

Can we do even better? What if we allow the constant 1 in our programming language?

Then, programs can count: `X:=X+1`!

Can we still decide if a program run in polynomial time?

Resource Analysis

Open Problem. *Let P be a program (in the language give above) that may contain the constants* 0 *and* 1*. Is it decidable if P runs in polynomial time under the weak semantics?*

Resource Analysis

Open Problem. *Let P be a program (in the language give above) that may contain the constants* 0 *and* 1*. Is it decidable if P runs in polynomial time under the weak semantics?*

So now we are one the edge of decidability!

Resource Analysis

Let P be a program (in the language give above) that does not contain any constants. Is it decidable if P runs in polynomial time under the weak semantics?

Resource Analysis

Let $\mathtt{P}$ be a program (in the language give above) that does not contain any constants. Is it decidable if $\mathtt{P}$ runs in polynomial time under the weak semantics?

Yes. PTIME.

Resource Analysis

Let $\mathtt{P}$ be a program (in the language give above) that does not contain any constants. Is it decidable if $\mathtt{P}$ runs in polynomial time under the weak semantics?

Yes. PTIME.

Let $\mathtt{P}$ be a program (in the language give above) that may contain the the constant 0. Is it decidable if $\mathtt{P}$ runs in polynomial time under the weak semantics?

Resource Analysis

Let P be a program (in the language give above) that does not contain any constants. Is it decidable if P runs in polynomial time under the weak semantics?
Yes. PTIME.

Let P be a program (in the language give above) that may contain the the constant 0. Is it decidable if P runs in polynomial time under the weak semantics?
Yes. PSPACE.

Resource Analysis

Let P be a program (in the language give above) that does not contain any constants. Is it decidable if P runs in polynomial time under the weak semantics?
Yes. PTIME.

Let P be a program (in the language give above) that may contain the the constant 0. Is it decidable if P runs in polynomial time under the weak semantics?
Yes. PSPACE.

Let P be a program (in the language give above) that may contain the constants 0 and 1. Is it decidable if P runs in polynomial time under the weak semantics?

Resource Analysis

Let $\mathtt{P}$ be a program (in the language give above) that does not contain any constants. Is it decidable if $\mathtt{P}$ runs in polynomial time under the weak semantics?
Yes. PTIME.

Let $\mathtt{P}$ be a program (in the language give above) that may contain the the constant 0. Is it decidable if $\mathtt{P}$ runs in polynomial time under the weak semantics?
Yes. PSPACE.

Let $\mathtt{P}$ be a program (in the language give above) that may contain the constants 0 and 1. Is it decidable if $\mathtt{P}$ runs in polynomial time under the weak semantics?
We don't know.

Resource Analysis

This should explain the title of – and the motivation for – the next paper in this line of research:

Resource Analysis

This should explain the title of – and the motivation for – the next paper in this line of research:

A. M. Ben-Amram & L. Kristiansen:
On the edge of decidability in complexity analysis of loop programs.
International Journal of Foundations of Computer Science, 2012.

Resource Analysis

This should explain the title of – and the motivation for – the next paper in this line of research:

A. M. Ben-Amram & L. Kristiansen:
On the edge of decidability in complexity analysis of loop programs.
International Journal of Foundations of Computer Science, 2012.

What do we find in this paper?

Resource Analysis

Terminology:

A program is *feasible* if every value computed by the program is bounded by polynomials in the inputs.

Resource Analysis

Terminology:

A program is *feasible* if every value computed by the program is bounded by polynomials in the inputs.

The *feasibility problem* for the programming language L is the decision problem

> input: *an L-program P*
> question: *Is P feasible?*

Resource Analysis

We consider the feasibility problem for a number of loop languages.

The syntax of a (typical) language we consider:

$$
\begin{array}{rcl}
\mathtt{X},\mathtt{Y},\mathtt{Z} \in \text{Variable} & ::= & \mathtt{X}_1 \mid \mathtt{X}_2 \mid \mathtt{X}_3 \mid \ldots \mid \mathtt{X}_n \\
\mathtt{C} \in \text{Command} & ::= & \mathtt{X{:=}Y} \mid \mathtt{X{:=}Y{+}Z} \mid \mathtt{X{:=}0} \mid \mathtt{X{:=}Y{+}1} \\
 & & \mid \mathtt{C}_1\,\mathtt{;}\,\mathtt{C}_2 \mid \mathtt{!loop}\ \mathtt{X}\ \{\mathtt{C}\}
\end{array}
$$

Resource Analysis

We consider two types of loops.

Definite loops: `!loop X { ....  }` (standard loops)

Indefinite loops: `?loop X { .....  }`

Resource Analysis

We consider three types of assignments.

Standard assignments: $\mathtt{X} := \langle exp \rangle$

Max assignments: $\mathtt{X} \stackrel{\max}{:=} \langle exp \rangle$

Weak assignments $\mathtt{X} :\leq \langle exp \rangle$

Resource Analysis

We consider three types of assignments.

Standard assignments: $\mathtt{X:=}\langle exp\rangle$

Max assignments: $\mathtt{X:}\stackrel{\max}{=}\ \langle exp\rangle$

Weak assignments $\mathtt{X} :\leq \langle exp\rangle$

We consider four different forms of $\langle exp\rangle$:

$$\mathtt{X+1} \qquad \mathtt{X+Y} \qquad \mathtt{Y} \qquad \mathtt{0}$$

Resource Analysis

A summary of our results.

expressions:	X+Y	X+Y, 0	X+Y, 0, X+1
indefinite loops	PTIME	PSPACE	?
definite loops max ass.	PTIME	PTIME	?
definite loops weak ass.	undecidable	undecidable	undecidable
definite loops standard ass.	undecidable	undecidable	undecidable

Thanks for your attention!

Thanks for your attention!

Thanks to my coauthors: Karl-Heinz Niggl, Neil Jones, Amir Ben-Amram, Jean-Yves Moyen, James Avery.

???????????????

. . . well, maybe I have time for a few more . . .