Structural Operational Semantics with transitivity rules and execution time

Patricia Peratto
psperatto@adinet.com.uy
Tecnólogo en Informática
FING-CETP.UdelaR-ANEP

March 2009

Abstract
We define an structural operational semantics of the core of an imperative language. It has a measure in the transitions corresponding to the number of steps of evaluation that takes place in the transition (from the point of view of usual complexity theory) and transitivity rules that allow to prove in the theory what is usually proved in the meta-theory.

1 Introduction

Our semantics has transitions of the form \( \Rightarrow^k \) where \( k \) is the number of steps of evaluation that takes place in the transition.

Rules to measure the time of evaluation of arithmetic expressions and statements for the natural semantics have been given in [3]. To our knowledge, haven’t been presented for the structural operational semantics.

This semantics applies to the evaluation of expressions and statements and can allow to measure the complexity of a program at the same time that we derive its construction applying the rules. The semantics is compositional and the measure of time also.

We show our semantics is deterministic and equivalent to the structural operational semantics without transitivity rules. We also show it is equivalent to the natural semantics with exact execution times presented in [3].

The outline of the paper is as follows: in section 2 we present the abstract syntax of the language and we introduce categories for different kind of expressions and for statements; follows in section 3 the semantics of the expressions and the set of rules for the semantics of statements. In section 4 we prove that the semantics is deterministic. In section 5 we present the equivalence with structural operational semantics. In section 6 we present the equivalence with natural semantics with execution times. In section 7 we present conclusions and further work.

2 Abstract syntax of the language

We use a syntactic notation based on BNF and we will use parenthesis (not indicated in our BNF) to solve ambiguities and uniquely determine the corresponding parse tree.

We have the following Syntactic Categories and meta-variables ranging over them:
The meta-variables can be primed or subscripted for example $n,n',n_1,n_2$ all stand for numerals.

2.1 Abstract syntax for arithmetic variables

\[
x ::= \text{letter } lx
\]
\[
lx ::= \epsilon | \text{letter } lx | \text{digit } lx
\]

That is, an string of letters and digits beginning with a letter. We assume letter and digit as understood.

2.2 Abstract syntax for Arithmetic Expressions

\[
a ::= n | ax | a_1 + a_2 | a_1 \times a_2 | a_1 - a_2
\]

2.3 Abstract syntax for Boolean Expressions

\[
b ::= \text{true} | \text{false} | a_1 = a_2 | a_1 < a_2 | \neg b | b_1 \lor b_2 | b_1 \land b_2
\]

2.4 Abstract syntax for Statements

\[
S ::= ax := a | \text{skip} | S_1; S_2 | \text{if } b \text{ then } S_1 \text{ else } S_2 | \text{while } b \text{ do } S
\]

3 Measured semantics of expressions

We define the following semantic functions:

\[
A : \text{Aexp} \to (\text{State} \to \mathbb{Z} \times \mathbb{N}) \text{ for arithmetic expressions}
\]
\[
B : \text{Bexp} \to (\mathbb{T} \times \mathbb{N}) \text{ for boolean expressions}
\]

where

\[
\mathbb{N} \text{ is the semantic domain of natural numbers,}
\]
\[
\mathbb{Z} \text{ is the semantic domain of integers,}
\]
\[
\mathbb{T} \text{ is the semantic domain of truth values,}
\]

and

\[
\text{State} = \text{AVar} \to \mathbb{Z}
\]

We have the following metavariables ranging over semantic categories: $k \in \mathbb{N}$, $z \in \mathbb{Z}$, $t \in \mathbb{T}$ and $s \in \text{State} \text{ primed or subscripted.}$
3.1 Semantic of Arithmetic Expressions

We define a transition system with transitions of the form

\[ <a, s> \rightarrow^k z \]

We define the following semantic functions

\[ A(a)s = <z, k> \text{ if exists } k \text{ such that } <a, s> \rightarrow^k z \]

\[ Value(a)s = first(A(a)s) \]

\[ Time(a)s = second(A(a)s) \]

Transition system:

\[ <n, s> \rightarrow^1 n \]

\[ <ax, s> \rightarrow^1 s(ax) \]

\[ <a_1, s> \rightarrow^{k_1} z_1 \quad <a_2, s> \rightarrow^{k_2} z_2 \]

\[ <a_1 + a_2, s> \rightarrow^{k_1+k_2+1} z_1 + z_2 \]

\[ <a_1, s> \rightarrow^{k_1} z_1 \quad <a_2, s> \rightarrow^{k_2} z_2 \]

\[ <a_1 * a_2, s> \rightarrow^{k_1+k_2+1} z_1 * z_2 \]

\[ <a_1, s> \rightarrow^{k_1} z_1 \quad <a_2, s> \rightarrow^{k_2} z_2 \]

\[ <a_1 - a_2, s> \rightarrow^{k_1+k_2+1} z_1 - z_2 \]

Theorem 3.1. \( A, Value \) and \( Time \) are functions.

Given \( a \) and \( s \), there exists only one \( k \) and \( z \) such that \( <a, s> \rightarrow^k z \).

Proof. By structural induction on \( a \). \( \square \)

3.2 Semantic of Boolean Expressions

We define a transition system with transitions of the form

\[ <b, s> \rightarrow^k t \]

We define the following semantic functions

\[ B(b)s = <t, k> \text{ if exists } k \text{ such that } <b, s> \rightarrow^k t \]

\[ Value(b)s = first(B(b)s) \]

\[ Time(b)s = second(B(b)s) \]

Transition system:
\[ <\text{true}, s> \rightarrow^1 \text{true} \]
\[ <\text{false}, s> \rightarrow^1 \text{false} \]
\[ A(a_1)s = <z_1, k_1> \quad A(a_2)s = <z_2, k_2> \]
\[ < a_1 = a_2, s > \rightarrow^{k_1 + k_2 + 1} z_1 = z_2 \]
\[ A(a_1)s = <z_1, k_1> \quad A(a_2)s = <z_2, k_2> \]
\[ < a_1 < a_2, s > \rightarrow^{k_1 + k_2 + 1} z_1 < z_2 \]
\[ < b, s > \rightarrow^k t \]
\[ < \neg b, s > \rightarrow^{k+1} \neg t \]
\[ < b_1, s > \rightarrow^{k_1} t_1 \quad < b_2, s > \rightarrow^{k_2} t_2 \]
\[ < b_1 \lor b_2, s > \rightarrow^{k_1 + k_2 + 1} t_1 \lor t_2 \]
\[ < b_1, s > \rightarrow^{k_1} t_1 \quad < b_2, s > \rightarrow^{k_2} t_2 \]
\[ < b_1 \land b_2, s > \rightarrow^{k_1 + k_2 + 1} t_1 \land t_2 \]

**Theorem 3.2.** \( B, \text{Value} \) and \( \text{Time} \) are functions.
Given \( b \) and \( s \), there exists only one \( k \) and \( t \) such that \(< b, s > \rightarrow^k t \).

**Proof.** By structural induction on \( b \).

### 3.3 Semantics of statements

The set of rules corresponding to statements is based in the rules for the Structural Operational Semantics of While presented in [3]. Are new the rules for transitivity and the rules for time. As is proved after, the rules for time are equivalent to the ones presented for natural semantics in the same book.

\[ <a \gets a, s> \Rightarrow^{\text{Time}(a)+1} s[a \gets \text{Value}(a)] \quad \text{ass1} \]
\[ <\text{skip}, s> \Rightarrow^1 s \quad \text{skip} \]
\[ < S_1, s> \Rightarrow^k < S'_1, s'> \]
\[ < S_1; S_2, s> \Rightarrow^k < S'_1; S'_2, s'> \quad \text{comp1} \]
\[ < S_1, s> \Rightarrow^k s' \]
\[ < S_1; S_2, s> \Rightarrow^k < S'_2, s'> \quad \text{comp2} \]
\[ \text{Value}(b)s = \text{true} \]
\[ <\text{if } b \text{ then } S_1 \text{ else } S_2, s> \Rightarrow^{\text{Time}(b)+1} < S_1, s> \quad \text{if1} \]
\[ \text{Value}(b)s = \text{false} \]
\[ <\text{if } b \text{ then } S_1 \text{ else } S_2, s> \Rightarrow^{\text{Time}(b)+1} < S_2, s> \quad \text{if2} \]
\[ <\text{while } b \text{ do } S, s> \Rightarrow^1 <\text{if } b \text{ then } (S; \text{while } b \text{ do } S) \text{ else } \text{skip}, s> \quad \text{while1} \]
\[ < S, s> \Rightarrow^{k_1} < S', s'> \quad < S', s'> \Rightarrow^{k_2} < S'', s''> \]
\[ < S, s> \Rightarrow^{k_1 + k_2} < S'', s''> \quad \text{trans1} \]
We define the semantic function
\[ S_{tsos}(S)s = s' \] in k steps if and only if \(< S, s \rangle \Rightarrow^k s'\)

## 4 Determinism

**Theorem 4.1** (The measured semantics presented in subsection 3.3 is deterministic.). If \(< S, s \rangle \Rightarrow^k < S', s' \rangle\) and \(< S, s \rangle \Rightarrow^k < S'', s'' \rangle\) then \( S' = S'' \) and \( s' = s'' \) and if \(< S, s \rangle \Rightarrow^k s'\) and \(< S, s \rangle \Rightarrow^k s''\) then \( s' = s'' \).

**Proof.** By induction on the shape of the derivation tree. If the statement is assignment, skip, if or while there is only one possible transition.

Assume the statement is \( S_1; S_2 \) (composition).

- suppose \(< S_1; S_2, s \rangle \Rightarrow^k < S_3, s' \rangle\) and \(< S_1; S_2, s \rangle \Rightarrow^k < S_4, s'' \rangle\) then the form of \( S_3 \) and \( S_4 \) is \( S_1 = S_1'; S_2 \) and \( S_1 = S_1''; S_1 \) necessarily follows that the hypothesis of the rule are \(< S_1, s \rangle \Rightarrow < S_1', s' \rangle\) or \(< S_1, s \rangle \Rightarrow < S_1'', s'' \rangle\). Then by inductive hypothesis \( S_1' = S_1'' \) and \( s' = s'' \).

- Suppose \(< S_1; S_2, s \rangle \Rightarrow^k < S_2, s' \rangle\) and \(< S_1; S_2, s \rangle \Rightarrow^k < S_2, s'' \rangle\) necessarily follows that the hypothesis of the rule are \(< S_1, s \rangle \Rightarrow s'\) or \(< S_1, s \rangle \Rightarrow s''\). Then by inductive hypothesis \( s' = s'' \).

Assume the rule applied is trans1.

- suppose \(< S, s \rangle \Rightarrow^k < S, s' \rangle\) and \(< S, s \rangle \Rightarrow k_1 + k_2 < S, s' \rangle\). The hypothesis of the rule must be of the form \(< S, s \rangle \Rightarrow^k < S', s' \rangle\) and \(< S', s' \rangle \Rightarrow^k < S_1, s_1 \rangle\) or \(< S', s' \rangle \Rightarrow^k < S_2, s_2 \rangle\). Follows by the inductive hypothesis \( S_1 = S_2 \) and \( s_1 = s_2 \).

- Suppose \(< S, s \rangle \Rightarrow^k < S_1, s_1 \) and \(< S, s \rangle \Rightarrow k_1 + k_2 s_2 \). The hypothesis of the rule must be of the form \(< S, s \rangle \Rightarrow^k < S', s' \rangle\) and \(< S', s' \rangle \Rightarrow^k < S_1, s_1 \rangle\) or \(< S', s' \rangle \Rightarrow^k < S_2, s_2 \rangle\). Follows by the inductive hypothesis \( s_1 = s_2 \).

\[ \square \]

**Theorem 4.2.** \(< S, s \rangle \Rightarrow^k s'\) and \(< S, s \rangle \Rightarrow k < S', s'' \rangle\) can not happen at the same time.

**Proof.** By induction on \( k \).

- By the corresponding rule \(< as := a, s \rangle \Rightarrow^k Time(a); s[ax := Value(a)]s\) and is not possible \(< as := a, s \rangle \Rightarrow^k < S, s' \rangle\) for any \( k, S \) and \( s' \).

- With \( \langle skip, s \rangle \) is similar.

- Consider the conclusion of the first rule for composition: \(< S_1; S_2, s \rangle \Rightarrow^k < S_1'; S_2, s' \rangle\) By hypothesis \(< S_1', S_2, s' \rangle \Rightarrow^k < S_2, s'' \rangle\) or \(< S_1'; S_2, s' \rangle \Rightarrow^k < S_3', s'' \rangle\) but not both, so \(< S_1; S_2, s \rangle \Rightarrow < S_2, s'' \rangle\) or \(< S_1; S_2, s \rangle \Rightarrow < S_3', s'' \rangle\) but not both.

- The second rule for composition is similar.

- By the if1 rule \(< if \ b \ then \ S_1 \ else \ S_2, s \rangle \Rightarrow^k Time(b); s[ax := Value(a)]s\) and is not possible \(< S_1, s \rangle \Rightarrow^k s'\) or \(< S_1, s \rangle \Rightarrow^k < S_1', s'' \rangle\) but not both. Then \(< if \ b \ then \ S_1 \ else \ S_2, s \rangle \Rightarrow < S_1, s \rangle \Rightarrow^k < S_1', s'' \rangle\) but not both.

5
• The rule \texttt{if2} is likely.
• The rule \texttt{while} is likely.
• Consider the transitivity rules. Can happen the following cases: \(< S, s > \Rightarrow^k < S', s' >\) and \(< S', s' > \Rightarrow^k < S'', s'' >\) or \(< S', s' > \Rightarrow^k s'' >\). Depending on the last transitions we chose rule \texttt{trans1} or \texttt{trans2}. The other possibility is \(< S, s > \Rightarrow^k s' >\) case in which this is the conclusion.

Corollary 4.3. If \(< S, s > \Rightarrow^k < S', s' >\) and \(< S, s > \Rightarrow^k s'' >\) then \(k_1 \leq k_2\).

Corollary 4.4. If \(< S, s > \Rightarrow^k < S', s' >\) and \(< S, s > \Rightarrow^k s'' >\) then \(k_1 \leq k_2\).

Proof. Suppose \(k_2 < k_1\). Then exists states \(s', s''\) and a statement \(S'\) such that \(< S, s > \Rightarrow^{k_2} s'' >\) and \(< S, s > \Rightarrow^{k_2} < S', s' >\) what by the previous theorem can not happen.

Corollary 4.5 (\(S\) is a function). If \(< S, s > \Rightarrow^k k_1 s' >\) and \(< S, s > \Rightarrow^k k_2 s'' >\) then \(k_1 = k_2\) and \(s' = s''\).

Proof. Suppose \(k_1 \neq k_2\), in particular \(k_1 < k_2\). Then happens \(< S, s > \Rightarrow^{k_1} s' >\) and \(< S, s > \Rightarrow^{k_2} s'' >\) what is absurd.

5 Equivalence with the structural operational semantics

5.1 Structural operational semantics

See [3] chapter 2 section 2 for the structural operational semantics of the language.

5.2 Equivalence

Lemma 5.1. If \(< S, s > \Rightarrow^* s' >\) then there exists \(k\) such that \(< S, s > \Rightarrow^k s' >\)

Proof. The rules of the structural operational semantics are included in the measured semantics with the exception that in the last, we have, the measure of time and transitivity rules. In the structural operational semantics the transitive closure is made outside of the rules, but has the same result. If we reach a final state in the structural operational semantics, the same state is reached in the measured semantics.

Lemma 5.2. If \(< S, s > \Rightarrow^k s' >\) then \(< S, s > \Rightarrow^* s' >\)

Proof. For each application of a rule in the measured semantics, apply the corresponding one step rule in the structural operational semantics.

Theorem 5.3 (\(S_{mos} = S_{tmos}\)).

Proof. Lemmas 5.1 and 5.2.

6 Equivalence with the natural semantics

6.1 Natural semantics

See [3] chapter 6 section 5 for the natural semantics of the language with exact execution times.
6.2 Equivalence

Lemma 6.1. If \( S, s \rightarrow^k s' \) then \( S, s \rightarrow^{k'} s'' \)

Proof. By induction on the time of the derivation.

- \( S = ax := a \). We assume \( < ax := a, s > \rightarrow^{Time(a)+1} s[\text{inl}(ax) := \text{inl}(\text{Value}[a])[s]] \).
  Applying ass, we get \( < ax := a, s > \rightarrow^{Time(a)+1} s[\text{inl}(ax) := \text{inl}(\text{Value}(a))[s]] \).
- \( S = \text{skip} \) analogous.
- \( S = S_1 ; S_2 \). Assume \( < S_1 ; S_2, s > \rightarrow^k s'' \) because \( < S_1, s > \rightarrow^{k_1} s' \) and \( < S_2, s' > \rightarrow^{k_2} s'' \) and \( k = k_1 + k_2 \). Applying the induction hypothesis to the derivations we get \( < S_1, s > \rightarrow^{k_1} s' \) and \( < S_2, s' > \rightarrow^{k_2} s'' \). Applying rule \( \text{comp2} \) to \( < S_1, s > \rightarrow^{k_1} s' \) we derive \( < S_1 ; S_2, s > \rightarrow^{k_1} < S_2, s' > \), and applying \( \text{trans2} \) to this conclusion and \( < S_2, s' > \rightarrow^{k_2} s'' \) we get \( < S_1 ; S_2, s > \rightarrow^{k_1+k_2} s'' \).
- \( S = \text{if } b \text{ then } S_1 \text{ else } S_2 \). Assume that \( < \text{if } b \text{ then } S_1 \text{ else } S_2, s > \rightarrow^{Time(b)+k+1} s' \), because \( B[b]s = \text{true} \) and \( < S_1, s > \rightarrow^k s' \). By induction hypothesis applied to \( < S_1, s > \rightarrow^{k_1} s' \) we get \( < S_1, s > \rightarrow^{k_1} s' \). Applying \( \text{if1} \) to \( S \) and \( \text{trans2} \) we get \( < \text{if } b \text{ then } S_1 \text{ else } S_2, s > \rightarrow^{Time(b)+k_1+1} s' \). If \( B[b]s = \text{false} \) is analogous.
- \( S = \text{while } b \text{ do } S' \). Assume that \( < \text{while } b \text{ do } S', s > \rightarrow^k s' \) because \( B[b]s = \text{true} \) and \( < S', s > \rightarrow^{k_1} s' \) and \( < \text{while } b \text{ do } S', s' > \rightarrow^{k_2} s'' \). By induction hypothesis \( < S', s > \rightarrow^{k_1} s' \) and \( < \text{while } b \text{ do } S', s' > \rightarrow^{k_2} s'' \). Applying \( \text{while1} \), \( \text{if1} \), and \( \text{comp2} \) we get:
  \( < \text{while } b \text{ do } S', s > \rightarrow^{Time(b)+2} < \text{while } b \text{ do } S', s > \rightarrow^{k_1} < \text{while } b \text{ do } S', s' > \rightarrow^{k_1+k_2} s'' \)
  applying \( \text{trans2} \) two times we get \( < S'; b \text{ do } S', s > \rightarrow^{k_1+k_2} s'' \) finally applying \( \text{trans2} \) to both derivations, we get \( < \text{while } b \text{ do } S', s > \rightarrow^{Time(b)+k_1+k_2+2} s'' \).
- \( S = \text{while } b \text{ do } S' \). Assume that \( < \text{while } b \text{ do } S', s > \rightarrow^{Time(b)+3} s \) because \( B[b]s = \text{false} \). Then \( < \text{while } b \text{ do } S', s > \rightarrow^{Time(b)+2} < \text{skip}, s > \), i.e. \( < \text{while } b \text{ do } S', s > \rightarrow^{Time(b)+3} s \).

\( \square \)

Lemma 6.2. If \( < S_1 ; S_2, s > \rightarrow^k s'' \) (\( k \geq 2 \)), then there exists a state \( s' \) and positive integers \( k_1 \) and \( k_2 \) such that \( < S_1, s > \rightarrow^{k_1} s' \) and \( < S_2, s' > \rightarrow^{k_2} s'' \) where \( k = k_1 + k_2 \).

Proof. See Lemma 2.19 in [3]. For our semantics the proof is analogous. \( \square \)

Lemma 6.3. If \( S, s \rightarrow^{k} s' \) then \( S, s \rightarrow^{k'} s'' \)

Proof. By induction on \( k \).

- \( S = ax := a \). Holds because \( < ax := a, s > \rightarrow^{Time(a)+1} s[\text{inl}(ax) := \text{inl}(\text{Value}(a))[s]] \).
- \( S = \text{skip} \) is analogous.
- \( S = S_1 ; S_2 \). Assume that \( < S_1 ; S_2, s > \rightarrow^k s'' \). Applying 6.2 we know that there exists \( s', k_1 \) and \( k_2 \) such that \( < S_1, s > \rightarrow^{k_1} s' \) and \( < S_2, s' > \rightarrow^{k_2} s'' \) where \( k_1 + k_2 = k \). Follows \( k_1 < k \) and \( k_2 < k \).
  By induction hypothesis, there are derivations in the natural semantics for \( < S_1, s > \rightarrow^{k_1} s' \) and \( < S_2, s' > \rightarrow^{k_2} s'' \). Using the rule in the natural semantics for composition we get \( < S_1 ; S_2, s > \rightarrow^k s'' \).

7
• $S = \text{if } b \text{ then } S_1 \text{ else } S_2, \text{Value}(b)s = \text{true}$. Assume $<i f \ b \ then \ S_1 \ else \ S_2,s \succ \rightarrow^k s'$; we must have transitions $<i f \ b \ then \ S_1 \ else \ S_2,s \succ \rightarrow^{\text{Time}(b)s+1} S_1,s >$ and $< S_1,s \succ \rightarrow^{\text{Time}(b)s-1} s'$.

The case $\text{Value}(b)s = \text{false}$ is analogous.

• $S = \text{while } b \ do \ S'$. Suppose we have $< \text{while } b \ do \ S',s \succ \rightarrow^k s'$, $\text{Value}(b)s = \text{true}$.

The derivation in our semantics is

$< \text{while } b \ do \ S',s \succ \rightarrow^1 <i f \ b \ then \ (S'; \text{while } b \ do \ S') \ else \ \text{skip},s >$

and we must have a derivation for

$<i f \ b \ then \ (S'; \text{while } b \ do \ S') \ else \ \text{skip},s \succ \rightarrow^{k-1} s'$ that reduces to

$<i f \ b \ then \ (S'; \text{while } b \ do \ S') \ else \ \text{skip},s \succ \rightarrow^{\text{Time}(b)s+1} (S'; \text{while } b \ do \ S'),s >$.

We have in the natural semantics

$< \text{while } b \ do \ S',s \succ \rightarrow^{\text{Time}(b)s+t't'\rightarrow^2 s''}, \text{if } < S',s \succ \rightarrow^{t'} s'$ and $< \text{while } b \ do \ S',s' \succ \rightarrow^{t''} s''$.

$<i f \ b \ then \ (S'; \text{while } b \ do \ S') \ else \ \text{skip},s \succ \rightarrow^{\text{Time}(b)s+t+1} s'$ where $t = t' + t$

and $< (S'; \text{while } b \ do \ S'),s \succ \rightarrow^{t'} s'$. By Lemma 6.3 $< (S'; \text{while } b \ do \ S'),s \succ \rightarrow^{t}$

$s'$, then $k - 1 = 1 + t + \text{Time}(b)s$

follows $k - 1 = \text{Time}(b)s + t + 1$ then $k = \text{Time}(b)s + t + 2$.

• $S = \text{while } b \ do \ S'$. Suppose we have $< \text{while } b \ do \ S',s \succ \rightarrow^k s$, $\text{Value}(b)s = \text{false}$.

The derivation in our semantics is

$< \text{while } b \ do \ S',s \succ \rightarrow^1 <i f \ b \ then \ (S'; \text{while } b \ do \ S') \ else \ \text{skip},s >$

And we must have a derivation for

$<i f \ b \ then \ (S'; \text{while } b \ do \ S') \ else \ \text{skip},s \succ \rightarrow^{k-1} s$.

We have in the natural semantics

$<i f \ b \ then \ (S'; \text{while } b \ do \ S') \ else \ \text{skip},s \succ \rightarrow^{\text{Time}(b)s+2} s$, where $< \text{skip},s \succ \rightarrow^1 s$ and $k - 1 = \text{Time}(b)s+2$ then it follows $< \text{while } b \ do \ S',s \succ \rightarrow^k s$.

Theorem 6.4 ($S_{\text{nts}} = S_{\text{tsos}}$).

Proof. Lemmas 6.1 and 6.3.

7 Conclusions and further work

We have presented a semantics that takes account of the number of steps of evaluation of expressions and statements following traditional complexity theory.

We expect in a future work, extend the semantics presented in this paper with functions and procedures. We are interested in finding recurrence expressions to measure the time of iterative and recursive programs. Recursivity will give rise to recurrences and in this way can be possible to find equations for the time of a program.
References


