BSS model of computation
over the Reals
and Choice Operator

Research Thesis

Submitted in Partial Fulfillment of The
Requirements for the Degree of
Master of Science in Computer Science

Yonit Magid

Submitted to the Senate of
the Technion - Israel Institute of Technology

Shevat, 5767 Haifa January 2007
The Research Thesis Was Done Under The Supervision of Prof. J. Makowsky in the Faculty of Computer Science.

I wish to thank my dear husband, Avi Magid, for his encouragement and support throughout my studies.
Contents

1 Introduction ................................................. 2
   1.1 Problems over the Reals ................................. 2
   1.2 Linear problems ........................................ 2
   1.3 Model of Computation .................................. 3
   1.4 Solving a problem ...................................... 3
   1.5 Quantifier Elimination ................................ 4
   1.6 Skolem Functions ...................................... 5
   1.7 Choice Operators ...................................... 6
   1.8 Main Results .......................................... 6
   1.9 Outline of Thesis ..................................... 7

2 Background .................................................. 8
   2.1 Formulas over Real Closed Fields ...................... 8
   2.2 Semi-Algebraic Sets ................................... 9
   2.3 Quantifier Elimination ................................ 9
   2.4 Quantifier Elimination Procedures .................... 10
     2.4.1 Improvement for QE Complexity - The Skolem Method 10
     2.4.2 Linear Quantifier Elimination ...................... 11
     2.4.3 Quantifier Elimination by Cylindrical Algebraic Decom-
           position ........................................ 12
   2.5 Problems and Linear Problems ......................... 12
   2.6 Choice Operators ..................................... 14
   2.7 Main result in [ER99] .................................. 14

3 Computational Model ........................................ 15
   3.1 The Blum Shub Smale Computation Model (BSS) ........ 15
   3.2 The Path Decomposition Theorem For the BSS Model .... 17
   3.3 The Eaves-Rothblum Computation Model (ER) ............ 19
   3.4 The Path Decomposition Theorem For the ER Model ..... 22
   3.5 Possible Variations over the ER Computation Model .... 23

4 Algorithms and Linear Algorithms .......................... 25
   4.1 Linear Problems Vs. Linear Algorithms ............... 25
   4.2 Problem Construction Given an Algorithm ............. 26
   4.3 Algorithm Construction Given a Problem ............... 27
   4.4 Choice Operators for the ER-Model (Revisited) ....... 30

5 Generalizing the Computational Model -
Polynomial Problems and Algorithms ........................ 31
   5.1 Skolem Functions ...................................... 31
   5.2 Using Skolem Functions for $Th(\mathbb{R})$ ............ 32
   5.3 Model Enhancements using Skolem Functions for $Th(\mathbb{R})$ 34
   5.4 The $ER^*$ Model and $\mathbb{R}^*$ Algorithms ............ 37
     5.4.1 The Path Decomposition Theorem for the $ER^*$ Model 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Content</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5</td>
<td>$\mathbb{R}^<em>$ Problems vs. $\mathbb{R}^</em>$ Algorithms</td>
<td>38</td>
</tr>
<tr>
<td>5.5.1</td>
<td>$\mathbb{R}^<em>$ Problem Construction Given an $\mathbb{R}^</em>$ Algorithm</td>
<td>39</td>
</tr>
<tr>
<td>5.5.2</td>
<td>$\mathbb{R}^<em>$ Algorithm Construction Given an $\mathbb{R}^</em>$ Problem</td>
<td>39</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Cylindrical Algebraic Decomposition (CAD) Algorithm Outline</td>
<td>40</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Back to Theorem 47</td>
<td>46</td>
</tr>
<tr>
<td>5.6</td>
<td>Choice Operators for the $ER^*$-Model (Revisited)</td>
<td>50</td>
</tr>
</tbody>
</table>
List of Figures

1 Figure for Example 49 ........................................... 42
2 Figure for Example 50 ........................................... 43
3 Figure for Example 51 ........................................... 45
4 Resulting CAD for Example 51 ............................... 46
5 CAD Algorithm Output - Cells Trees. Red cells denote true cells. 47
Abstract

B.C. Eaves and U.G. Rothblum studied the notion of linear algorithms and linear problems over the reals as an ordered field. They showed that by augmenting straight-line programs with random assignments from specified intervals one gets an exact correspondence between linear problems and algorithms.

The class of the linear problems is huge, diverse, complex and important. The class contains, for example, finding the rank of matrices, optimizing linear programming problems, optimizing bounded variable integer programs, solving linear complementary problems and determining satisfiability in sentential logic.

The notion of random assignments (i.e. arbitrarily picking a value to assign, no probability space involved) was introduced in order to achieve the complete solution set of a problem, which is useful in many cases.

The purpose of this research thesis is to place Eaves and Rothblum's result in a wider algorithmic context.

We extend this idea to algebraic problems and algorithms by introducing random assignments for roots of univariate polynomials. We also give an alternative proof of the Eaves-Rothblum characterization of the linear case. In both cases we give complexity estimates based on the complexity of cylindrical algebraic decomposition or, in the linear case, of the simplex algorithm. We also discuss the notion of choice operators as a more generalized form of random assignments.
1 Introduction

1.1 Problems over the Reals

We look at problems over the reals, which can be described by quantified first order formulas in the language of ordered fields. Following are a few examples for such problems:

Examples 1

- **Solvability of polynomial equations:**

\[ \exists \bar{x} \bigwedge_i p_i(\bar{x}) = 0 \]

- **Solvability of polynomial inequalities:**

\[ \exists \bar{x} \bigwedge_i p_i(\bar{x}) \leq 0 \]

- **Solvability with parametric condition:**

\[ \forall \bar{x}(\Phi(\bar{x}) \rightarrow \exists \bar{y} P(\bar{x}, \bar{y}) = 0) \]

1.2 Linear problems

In operation research very often such problems are linear in a subset of the problem’s variables. In particular, if we consider a partition of the problem’s variables into a set of data variables (the problem’s parameters) and a set of solution variables (the variables whose value we would like to obtain given a value of the data variables), then we define a problem to be linear if it is linear in the subset of its solution variables and its quantified variables.

**Example 2** (\(A\bar{x} = \bar{b}\)), with \(A, \bar{b}\) parameters (data variables) and \(\bar{x}\) solution variables, is a linear problem, since it is linear in its solution variables (\(\bar{x}\)) and has no quantified variables.

**Example 3** \(z(x_1^2 y + x_1 x_2^3 - z = 0)\), with \(x_1, x_2\) parameters (data variables) and \(y\) the solution variable, is a linear problem, since it is linear in its solution variables (\(y\)) and its quantified variables (\(z\)).
1.3 Model of Computation

A suitable model of computation for our purpose is the Blum Shub Smale model (BSS, see [BCSS97].) It is a unit cost model and therefore the issue of bit coding of real parameters is avoided. We introduce certain variations to this model, which better fit the needs of solving problems and linear problems. The first variation is the addition of non-determinism, by adding a random assignment instruction to the model. The second is restricting our discussion to loop-free machines, which we will refer to as BSS Circuits. Additional variations, are the addition of instructions for computing the square root of a value, or the root of an odd degree polynomial (leastroot or non-deterministic selection of one of the roots), as black box instructions.

For each computation model we characterize the machines input sets, halting sets and computed functions.

An Algorithm is given by a machine over a given computation model, and a specification of its input and output variables.

A Linear Algorithm is an algorithm for which its machine’s computation instructions are linear in the set of output variables and register variables. Here we are once more interested in the restriction to Linear Circuits (i.e a linear algorithm whose machine is also a circuit).

1.4 Solving a problem

First we define a syntactical match between problems and algorithms, when the problem’s data and solution variables coincide with the algorithm’s input and output variables respectively. Next, we say that an algorithm solves a given problem if they match syntactically, and for every assignment to the data/input variables, and every execution of the algorithm (for the non-deterministic case), the output value is a solution to the problem (given the input value).

In many cases it is significant to find an algorithm which represents the full solution set of the problem, i.e. an algorithm for which for every input value, each possible solution is also a possible output. In fact, this is the motivation behind adding non-determinism (random assignment instructions) to our computation model. Such an algorithm is said to completely solve the problem.

The following examples will illustrate the notions of solved by and completely solved by.

Example 4 Consider the problem

\[ ax^2 + bx + c = 0 \]

where \( a, b, c \) are the data variables and \( x \) is the solution variable. When solutions exist, the solutions are given by:

\[ x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]
One possible algorithm:

Take $a, b, c$ as input.

If $\neg((a = 0 \land b = 0 \land c = 0) \lor (a = 0 \land b \neq 0) \lor (a \neq 0 \land b^2 - 4ac \geq 0))$ then fire “No Solution”.

This algorithm solves the problem, but it does not completely solve the problem, because only one of the solutions can be output for all executions. An algorithm which completely solves the problem can be:

Take $a, b, c$ as input.

If $\neg((a = 0 \land b = 0 \land c = 0) \lor (a = 0 \land b \neq 0) \lor (a \neq 0 \land b^2 - 4ac \geq 0))$ then fire “No Solution”.

Otherwise if $a = 0 \land b = 0 \land c = 0$ output choose $z$ with $-\infty \leq z \leq \infty$

Otherwise if $a = 0$ output $-\frac{x}{b}$

Otherwise,

$z \leftarrow$ choose $z$ with $0 \leq z \leq 1$

If $z \geq 0.5$ output $\frac{a + \sqrt{b^2 - 4ac}}{2a}$.

If $z < 0.5$ output $\frac{b + \sqrt{b^2 - 4ac}}{2a}$.

In this case all of the problem’s solutions can be output for different algorithm executions, therefore this algorithm completely solves the problem.

Example 5 Consider the problem

$$a \leq x \leq b \land y \leq 3 - x$$

where $a, b$ are the data variables and $x, y$ are the solution variables. The algorithm:

Input $a, b$. Output $\frac{a + b}{2}, \frac{3 - a - b}{2}$.

solves the problem, but it doesn’t cover the full solution set, and therefore does not completely solve the problem.

An algorithm which completely solves the problem:

Input $a, b$.

$x \leftarrow$ choose $x$ with $a \leq x \leq b$

$y \leftarrow$ choose $y$ with $-\infty \leq y \leq 3 - x$

Output $x, y$

1.5 Quantifier Elimination

Quantifier Elimination is the process of taking a quantified formula $\phi$ and obtaining a quantifier free formula $\psi$ equivalent to $\phi$. Quantifier Elimination will be used in this work in order to simplify problems, so that it suffices for algorithms to address only quantifier free problems.
Example 6 For the formula $\exists x(ax^2 + bx + c = 0)$, an equivalent quantifier free formula would be: $(a = 0 \land b = 0 \land c = 0) \lor (a = 0 \land b \neq 0) \lor (a \neq 0 \land b^2 - 4ac \geq 0)$

For linear inequalities Fourier already gives an algorithm (rediscovered by Dines 1919, Motzkin 1936, for history see [Wei94]). Fourier’s method can be used to eliminate linear occurrences of quantifiers, see also [vdD81].

Eaves and Rothblum popularized this method in operations research. Their main observation is that choices can be described by parametrized intervals.

For arbitrary formulas Tarski developed the method of QE, independently discovered by Seidenberg, in purely algebraic terms. Tarski’s method was computationally improved by Collins, using Cylindrical Algebraic Decomposition to achieve quantifier elimination.

For a detailed historical and technical exposition of these methods see [CJ98].

1.6 Skolem Functions

Skolem functions are used for eliminating existential quantifiers to produce a formula equivalent to the original. Skolemization is an application of the (second-order) equisatisfiability

$$\forall x \exists y R(x, y) \iff \exists f \forall x R(x, f(x))$$

i.e., $\forall x \exists y R(x, y)$ is satisfiable iff there is a function $f(x)$ such that $\forall x R(x, f(x))$ is satisfiable. The function $f$ is called a Skolem Function.

A Theory $T$ admits definable Skolem Functions if for every formula $\phi$, such that $T \models \forall \bar{x} \exists \bar{y} \phi(\bar{x}, \bar{y})$, there exists a definable function $f$ for which $T \models \forall \bar{x} \phi(\bar{x}, f(\bar{x}))$.

The following examples refer to $\mathbb{R} = (\mathbb{R}, +, -, \cdot, : , <, 0, 1)$ and $T = Th(\mathbb{R})$.

Example 7 For the formula

$$\forall x \forall y \exists z(x \neq y \rightarrow ((x < y \land x < z < y) \lor (y < x \land y < z < x)))$$

there exists a skolem function (definable over our language): $f(x, y) = \frac{x + y}{2}$, such that

$$\forall x \forall y(x \neq y \rightarrow ((x < y \land x < f(x, y) < y) \lor (y < x \land y < f(x, y) < x)))$$

holds.

Example 8 For the formula

$$\forall x \exists y(y^2 = x)$$

there is no definable skolem function over our language.
Eaves and Rothblum combine QE for linear quantifiers with skolem functions for linear inequalities. Their approach is to replace the deterministic skolem function by a choice operator.

We extend this idea to the non-linear case, by replacing the deterministic skolem function for polynomial roots (which produces the least root) by a choice operator (choosing over the relevant polynomial roots).

For a general treatment of definability of skolem functions see [Sco88].

1.7 Choice Operators

Choice operators were studied in programming in various contexts. In dynamic logic [Har84] a random assignment for first order variables was introduced to discuss program behaviour. In [BG00] the expressive and computational power of various random assignments is discussed.

Algorithms using random assignments are not randomized algorithms. Randomized algorithms are run many times and the result depends on the statistics of these many outcomes.

An algorithm with a random assignment has one specific output for each random assignment.

To avoid confusion we call the random assignment pick instructions. The pick instruction which we use in our model is interval-pick (i.e. random assignment over a parametric interval). We also discuss several alternative pick instructions (alternative choice operators). Using the pick instructions enables us to obtain algorithms which completely solve problems.

1.8 Main Results

Following are the main results in this thesis:

- Proving an extended version of the path decomposition theorem in a model with choice operators.

- Proving the following theorem for the algebraic case:

**Theorem 9** For each algebraic problem $P$ there exists an algebraic algorithm $A$ such that $P$ is completely solved by $A$, and vice versa, i.e. for each algebraic algorithm $A$ there exists an algebraic problem $P$ such that $P$ is completely solved by $A$.

- Providing an alternative proof to the Eaves-Rothblum theorem via the cylindrical algebraic decomposition algorithm, as a special case of the algebraic case.

- Examining possible alternatives of choice operators for our model, we conclude that the (seemingly much stronger) alternatives we discuss would not add expressive nor computational power to our model (compared to the choice operator of random assignments over a parametric interval).
1.9 Outline of Thesis

Section 2 contains the background to this work.

In section 3 we present the Blum Shub Smale (BSS) computational model and a variation of it inspired by Eaves-Rothblum work, which will be used for the linear case. We also present and prove the path decomposition theorem, which characterizes the models' input and output, for both models. This section also includes an initial discussion of possible variations to our model using different choice operators.

Section 4 presents our alternative proof of the Eaves-Rothblum characterization of the linear case. It starts by defining a linear algorithm using the computational model defined in the previous section. Then problems and algorithms equivalency, and the relations “A Problem Solved By an Algorithm” and “A Problem Completely Solved By an Algorithm” are defined. Then our proof of Eaves-Rothblum theorem for the linear case is presented. Ultimately, the discussion of choice operators is revisited in the context of the proof presented in this section.

Section 5 contains the extension of the Eaves-Rothblum work to algebraic problems and algorithms. It presents the extended computational model for the algebraic case, using skolem functions which are introduced at the beginning of this section, and adding random assignments for roots of univariate polynomials. Then the proof of the characterization of the algebraic case is presented, which is based on the CAD algorithm. Ultimately, the discussion of choice operators is revisited once more in the context of the proof presented in this section.
2 Background

This section provides the basic definitions used from this point on.

2.1 Formulas over Real Closed Fields

Real Closed Fields:
Real closed fields are ordered fields in which every polynomial of odd degree has a root and every positive member has a square root.

From this point on, unless otherwise stated everything is restricted to being over real closed fields. However, some of the results also hold for any ordered field.

Although the order does not exist in the original definition of real closed fields, it can be added using the following definition:
We define a positive cone $C$ by $\{x\exists y(y^2 = x)\}$ and now we define the order relation by

$$x < y \text{ iff } y - x \in C$$

Formulas Over Real Closed Fields:

Terms: Polynomials over $\mathbb{Q}[X]$, $X = \{x_1, x_2, \ldots\}$ with coefficients from $\mathbb{Q}$.

Atomic Formulas: $(p = q)$ or $(p > q)$ where $p$ and $q$ are polynomials.

Formulas are obtained by induction using the logical connectives ($\&$, $\land$, $\lor$, etc.), parentheses and quantifiers ($\forall$, $\exists$). For example $\forall x \exists y(y^2 = x \lor y^2 = -x)$ is a formula.

A quantifier free formula is in Disjunctive Normal Form or DNF if it is in the following form:

$$\bigvee_{i \in I} \bigwedge_{j \in J(i)} \phi_j$$

where $\phi_j$ are atomic formulas.

A quantified formula is in Prenex Normal Form if it is in the following form:

$$Q_1 x_1 \ldots Q_n x_n \phi$$

where each $Q_i$ represents a quantifier ($\exists$, $\forall$) and $\phi$ is a quantifier free formula in DNF.

For every formula an equivalent formula which is in Prenex Normal Form can be obtained.

Linear Formulas (in a set of variables):
A formula (quantified or unquantified) is defined to be linear in a set of variables
(quantified and/or unquantified) if all of the occurrences of these variables in all of the polynomials appearing in the formula are linear.

An occurrence of a variable $x$ in a polynomial $p(x_1, \ldots, x_n, x)$ has the following form: $p'(x_1, \ldots, x_n) \cdot x^j$. Such an occurrence is linear with respect to the set of linear variables $\{x_i, \ldots, x_n\}$ if $j \leq 1$ and if $j = 1$ then for each $x_i \in \{x_1, \ldots, x_n\}$ the maximal degree of $x_i$ in $p'$ is 0.

The following example will help illustrate this definition.

**Example 10** The formula $xy + 1 > 0$ is linear in $\{x\}$, linear in $\{y\}$, but not linear in $\{x, y\}$.

### 2.2 Semi-Algebraic Sets

A subset of $\mathbb{R}^n$ is **semi-algebraic** if it is definable by a formula over real closed fields which is quantifier free. Since, as shown in the next subsection, there is a quantifier elimination procedure for real closed fields, a subset of $\mathbb{R}^n$ is semi-algebraic if and only if it is definable by a formula over real closed fields.

### 2.3 Quantifier Elimination

**Quantifier Elimination** is the process of taking a quantified formula $\phi$ and obtaining a quantifier free formula $\psi$ equivalent to $\phi$. Quantifier Elimination will be used later in this work in order to simplify problems by regarding only quantifier free problems (problems will be defined in subsection 2.5).

**Examples:**

- For the formula $\exists x(ax + b = c)$, an equivalent quantifier free formula would be: $(a = 0) \land (b = c) \lor (a \neq 0)$.

- For the formula $\exists x(ax^2 + bx + c = 0)$, an equivalent quantifier free formula would be: 
  
  $((a = 0 \land b = 0 \land c = 0) \lor (a = 0 \land b \neq 0) \lor (a \neq 0 \land b^2 - 4ac \geq 0))$.

In the first example we can see that the equivalent quantifier free formula expresses the condition for the existence of such an $x$ such that $ax + b = c$.

In the second example, the equivalent quantifier free formula expresses the condition for the existence of a solution $x$ to the quadratic equation $ax^2 + bx + c = 0$.

This is also the case for general existential formulas (obtaining the equivalent quantifier free formula is finding the condition (using the unquantified variables) for the existence of the quantified variables such that the quantifier free part of the original formula holds). As mentioned before we can limit our discussion to formulas in normal form without loss of generality.
2.4 Quantifier Elimination Procedures

Quantifier elimination procedures are an important tool in model theory. For a theory $T$ for which such a procedure exists (i.e. $T$ admits quantifier elimination), decision problems (problems of deciding whether a sentence of the theory, i.e. a formula with no free variables, is true or false) are reduced to the decision of quantifier free sentences with respect to $T$. Since the truth of every quantifier free sentence of a theory over ordered fields can be decided effectively (since there are no variables the problem is reduced to the decision of term inequalities and then applying the logical connectives on true and false terms), a quantifier elimination procedure yields a decision algorithm for the theory.

Therefore many quantifier elimination procedures have been developed. However, complexity-wise, many of them are far from being feasible.

**Theorem 11 (Tarski)** $Th((\mathbb{R}, +, \cdot, 1, 0, -1, =, >))$ admits quantifier elimination.

Traditional quantifier elimination procedures, such as Tarski's method which can be found in [CJ98], eliminate one quantifier at a time, at each step handling a formula $\exists x(\phi)$, where $\phi$ is quantifier free and simple enough (in DNF). Therefore, before each elimination step we have to bring the formula to DNF. But any such manipulation may increase the length of the formula exponentially. So, for an unbounded number of quantifiers this causes an exponential explosion in the length of the formula (the length of the formula behaves as an exponential tower function of the number of quantifiers).

We will present the outlines of two feasible quantifier elimination procedures for Real Closed Fields which will be used later on in this work. Both are of double exponential time complexity.

The first is a linear quantifier elimination procedure, i.e. a quantifier elimination procedure which is suitable for linear formulas and which preserves the formula's linearity. If the original formula is linear in a subset of its variables then the obtained formula will also be linear in these variables. This property will be significant later on. Such procedures are presented in [vdD81], [Wei88] and [FR75].

The second procedure - Quantifier Elimination by Cylindrical Algebraic Decomposition - can be used in the general (non-linear) case.

2.4.1 Improvement for QE Complexity - The Skolem Method

The new approach in these methods is to manipulate only atomic formulas while preserving the formula's boolean structure (up to a small boolean "noise" at the innermost level of the formula), so no repetitive DNF rearrangement is necessary. For some theories, the exponential explosion problem of the traditional quantifier elimination methods can be circumvented by disregarding the internal structure of the formula $\exists x(\phi)$ and trying to find a finite set of terms (depending on the formula's free variables) to be used as witnesses for the existence of an element $x$ satisfying $\phi$. This set is called a set of Skolem Terms. In [Wei88] it
is shown, that if such a set can be constructed in polynomial time for each \( x, \Psi \) (where \( \Psi \) is a finite set of atomic formulas), and some bounds of the Skolem Terms set size and each term’s “size” hold, then there is a quantifier elimination procedure with complexity of double exponential time, and the formula’s growth is also at most double exponential.

In the next subsection we present a quantifier elimination procedure based on this method.

2.4.2 Linear Quantifier Elimination

For the case of linear formulas over ordered fields (which is of special interest here), a substitution method of linear variables by linear terms in a linear formula is presented, so that after substitution the formula remains linear. This yields a Linear Quantifier Elimination procedure which uses linear Skolem Terms and the above substitution method, so that if the formula is linear, then the resulting equivalent quantifier-free formula is also linear. The details of this procedure and of its complexity bounds are discussed in [Wei88].

Following is an outline of one such method of linear quantifier elimination procedures - Rackoff’s method (see [FR75]). We describe the construction of the Skolem terms for the linear case.

For handling \( \exists x(D(x, \bar{y})) \), \( D \) quantifier free:

Atomic formulas of \( D \) are of the form: \( \sim a \Delta x, \Delta \in \{<, =, >\} \) where \( b \) is a linear term in which \( x \) does not appear, and \( a \) does not contain any linear variables (it contains non-linear variables and constants).

We calculate \( D_{\infty} \) and \( D_{-\infty} \), which mean \( D \) when assigning to \( x \) a very small value (\( x \to -\infty \)) or a very large value (\( x \to \infty \)) accordingly, by replacing each atomic formula of \( D \) by the following, according to its type:

- \( x < t \) replace by \textit{true} for \( D_{\infty} \) and by \textit{false} for \( D_{-\infty} \). (If the value of \( x \) is small enough then it is true that \( x < t \) and therefore this subformula of \( D \) yields \textit{true} in \( D_{\infty} \). If the value of \( x \) is large enough then \( x < t \) doesn’t hold and therefore this subformula of \( D \) yields \textit{false} in \( D_{-\infty} \).

- \( t < x \) replace by \textit{false} for \( D_{\infty} \) and by \textit{true} for \( D_{-\infty} \). (If the value of \( x \) is large enough then it is true that \( t < x \) and therefore this subformula of \( D \) yields \textit{true} in \( D_{\infty} \). If the value of \( x \) is small enough then \( t < x \) doesn’t hold and therefore this subformula of \( D \) yields \textit{false} in \( D_{-\infty} \).

- \( t = x \) replace by \textit{false} for both \( D_{\infty} \) and \( D_{-\infty} \). (For both a large enough value of \( x \) and a small enough value of \( x \) the equality \( t = x \) doesn’t hold and therefore this subformula of \( D \) yields \textit{false} in both \( D_{\infty} \) and \( D_{-\infty} \).

Let \( U = \{ t \mid (x \Delta t) \in D \} \). Now the formula \( \exists x(D(x, \bar{y})) \) will be replaced by:

\[
D_{\infty} \lor D_{-\infty} \lor \bigvee_{t, \bar{v} \in U} D\left(\frac{t + \bar{v}}{2}, \bar{y}\right)
\]

which is equivalent (if there exists a value of \( x \) such that \( D \) holds then it is either a large enough value of \( x \) (\( \infty \)) or a small enough value of \( x \) (\( -\infty \), or a
value which is between 2 relevant terms \( \left( \frac{t + v}{2} \right) \) when for example \( t < x \) and \( x < v \) are subformulas of \( D \)).

**Complexity of Rackoff’s Method:**
The new formula’s size in each step would be:

\[
|D_\infty|, |D_{\infty}| = O(|D|)
\]

\[
\left| \bigvee_{t,v \in \mathcal{U}} D\left( \frac{t + v}{2}, \bar{y} \right) \right| = n^2 O(|D|)
\]

Therefore the growth in each step is polynomial. For handling universal quantifiers, \( \forall x \) is replaced by \( \neg \exists x \).

Therefore the time complexity would be:

\[
t_n = (t_{n-1})^2 = \ldots = (n)^2^n
\]

I.e. the time complexity is double exponential.

### 2.4.3 Quantifier Elimination by Cylindrical Algebraic Decomposition

We will present an outline of this method, which will be used for the general (non-linear) case, i.e., for quantifier elimination of non-linear formulas. This method is presented in detail in [CJ98].

The Cylindrical Algebraic Decomposition (CAD) algorithm receives as input a set of polynomials in \( r \) variables and then constructs a decomposition of real \( r \)-dimensional space into a finite number of connected regions, called cells, in which each polynomial is invariant in sign. Moreover, these cells are arranged in a certain cylindrical manner.

The QE algorithm receives as input a formula and applies the CAD algorithm to the set of polynomials occurring in this formula. The QE algorithm then uses the output of the CAD algorithm and applies the quantifiers in the formula by using a sample point from each cell to determine the (invariant) truth value of the input formula in that cell. This application of quantifiers reveals for which cells in the subspace of the free variables the formula yields true. This knowledge is then used to construct an equivalent quantifier free formula. This is achieved by a method called augmented projection that provides a quantifier free formula for each of the true cells.

### 2.5 Problems and Linear Problems

In this section we define one of the basic elements to be used throughout this work - a *problem*.

**Problems Over Ordered Fields:**
A problem over an ordered field \( F \), is a subset of \( F^{n+m} \). We will look only at FOL definable such subsets.
Let $P$ be an FOL formula over ordered fields, and let
\[ x = (x_1, \ldots, x_m) \]
\[ y = (y_1, \ldots, y_n) \]
where $x \cup y$ is the set of all unquantified variables in $P$ and $x$ and $y$ are disjoint. Then, $(P, x, y)$ defines a problem over ordered fields or in short, a problem. The variables of $x$ and $y$ are respectively called the data and solution variables of the problem.

From this point on a formula $P$ defining a problem will be referred to, in short, as a problem. We will alternatively also refer to $(P, x, y)$ as a polynomial problem.

Example 12

\[ ((ax^2 + bx + c = 0), (a, b, c), x) \]
is a polynomial problem, where the quadratic equation is the problem formula, $(a, b, c)$ are the data variables and $x$ is the solution variable.

Problem Instance: Let $(P, x, y)$ be a problem, let $F$ be an ordered field and let $\bar{x}$ be an assignment of $x$ in $F$. Then $((P, x, y), F, \bar{x})$ is a problem instance. Given a problem instance, the task is to find an assignment $\bar{y}$ in $F$ so that $P(x \leftarrow \bar{x}, y \leftarrow \bar{y})$ is true in $F$. Then $\bar{y}$ solves the problem instance, and $(\bar{x}, \bar{y})$ is a data - solution pair for $P$ over $F$.

Example 13 For the example problem above, if we take $\mathbb{R}$ as the ordered field and the assignment $(1, 4, 3)$ to the data variables $(a, b, c)$, then

\[ (((ax^2 + bx + c = 0), (a, b, c), x), \mathbb{R}, (1, 4, 3)) \]
is a problem instance, and $((1, 4, 3), -1)$ and $((1, 4, 3), -3)$ are the data - solution pairs for the problem over $\mathbb{R}$.

Linear Problems: A Linear Problem is a problem definable by a formula $P(\bar{x}, \bar{y})$ which is linear in the set of quantified and solution variables.

Example 14

The linear equations system problem:

\[ ((A\bar{x} = \bar{b}), (A, \bar{b}), \bar{x}) \]
is a linear problem, since the problem formula is quantifier free and linear in $\bar{x}$ which is the solution variable of this problem.

Example 15

For the formula $P = xy + xz + w^3 \geq 0$:
$(P, (y, z, w), (x))$ is a linear problem.
$(P, (z, w), (x, y))$ is not a linear problem.
2.6 Choice Operators

In [ER99] the notion of \textit{pick} is introduced in order to add non-determinism to the computational model. We use a similar notion in the definition of our computational model (which will be presented in later sections), and also discuss possible variations of using more general \textit{choice operators}, inspired by the discussion in [BG00]. We examine how using different choice operators would affect the computation power of our model.

2.7 Main result in [ER99]

\textbf{Theorem 16 ([ER99])} \textit{There is a one-to-one and onto correspondence from equivalence classes of linear problems to equivalence classes of randomized linear algorithms.}

The purpose of this thesis is to place this theorem in a wider context.

We start by defining and characterizing our computational model. Then we present a proof for this theorem using our computational model. Next, we generalize this result to the context of polynomial (non-linear) problems and algorithms. For this we present an enhanced computational model, and prove the following result using this model:

\textbf{Theorem 17} \textit{There is a one-to-one and onto correspondence from equivalence classes of polynomial problems to equivalence classes of randomized polynomial algorithms.}
3 Computational Model

In this section we present the computational model we will use in the following sections. The model is based on the Blum Shub Smale (BSS) model of computation, which is a register machine over the reals at unit-cost, i.e. each register holds a real number with unlimited precision. The BSS model is presented in [BCSS97]. Our model also uses the concept of the model presented in the Eaves-Rothblum preprint [ER99], of the addition of non-determinism to our computation model. Subsections 3.1 and 3.2 are also based on the preprint of [MM02]. In subsection 3.1 we present the definition of the BSS model of computation. In subsection 3.2 we present the Path Decomposition Theorem for the BSS model, which characterizes the model's input and output, with some additional definitions, and a more detailed proof for the path decomposition theorem than presented in [MM02], as a preparation for later sections. In subsection 3.3 we present our model of computation, which is based on the BSS model, with several modifications inspired by the model presented in [ER99]. This model will be used in the following section to prove the theorem presented in [ER99] using a different approach - a BSS-like model. In subsection 3.4 we present and prove a Path Decomposition Theorem for the new model.

3.1 The Blum Shub Smale Computation Model (BSS)

Following is a definition of the BSS model of computation, as presented in [MM02].

Definition 18

\[
\mathbb{R}^\infty := \bigcup_{i=1}^{\infty} \mathbb{R}^i := \{(x_1, x_2, \ldots) | x_i \in \mathbb{R} \land \exists k \in \mathbb{N} \forall j > k \land (x_j = 0)\}
\]

Equivalently to Turing machines, which operate over boolean strings, the BSS machine model operates over strings of real numbers, i.e. over inputs from \(\mathbb{R}^\infty\).

Let \(Y \subseteq \mathbb{R}^\infty := \bigcup_{k \in \mathbb{N}} \mathbb{R}^k\). A Blum-Shub-Smale machine \(M\) over \(\mathbb{R}\) with admissible input set \(Y\) (shortly: a BSS machine over \(Y\)) is given by a finite set \(I\) of instructions labeled by \(1, \ldots, N\), and an infinite set of registers \(R_1, R_2, \ldots\). Each of the machine's registers holds a real number (i.e. this is a unit-cost model).

A configuration of \(M\) is a quadruple \((n, i, j, x) \in I \times N \times N \times \mathbb{R}^\infty\), where \(n\) denotes the currently executed instruction, \(i\) and \(j\) are used as addresses (copy-registers) and \(x\) is the actual content of the registers \(R_1, R_2, \ldots\) of \(M\).

The initial configuration of \(M\)'s computation on input \(y \in Y\) is \((1, 1, 1, y)\). If \(n = N\), i.e. the current configuration is \((N, i, j, x)\) then the computation stops. A predetermined part of the vector \(x\) is then outputted.

The instructions \(M\) can perform are of the following types:
• **computation**: \( n : x_s \leftarrow x_k \circ_n x_i \), where \( \circ_n \in \{+, -, \cdot, \div\} \), or \( n : x_s \leftarrow \alpha \) for some constant \( \alpha \in \mathbb{R} \).

  The register \( x_s \) will get the value \( x_k \circ_n x_i \) or \( \alpha \) respectively. All other register entries remain unchanged. The next instruction will be \( n + 1 \) and the copy registers are changed selectively according to \( i \leftarrow i + 1 \) or \( i \leftarrow 1 \) and similarly for \( j \).

• **branch**: \( n : \text{if } x_1 \geq 0 \text{ goto } \beta(n) \text{ else goto } n + 1 \).

  According to the answer of the test the next instruction is determined \((\beta(n) \in I)\). All other registers are not changed.

• **copy**: \( n : x_i \leftarrow x_j \)

  The content of the read register is copied into the write register. The next instruction is \( n + 1 \). All other register entries remain unchanged.

All \( \alpha \) appearing among the computation instructions build up the (finite) set of **machine constants** of \( M \).

**Example 19** An example for a BSS machine \( M \) can be given by the following set of instructions:

1: if \( x_1 \geq 0 \) goto 3 else goto 2

2: \( x_4 \leftarrow x_1 \cdot x_1 \)

3: \( x_2 \leftarrow 2.5 \cdot x_1 \)

4: \( x_3 \leftarrow x_2 + 5 \)

For this machine the set of machine constants is \( \{2.5, 5\} \). We can see that this machine has two computation paths which the computation can follow, depending on the result of the branch condition in the first instruction. The two possible paths would be: \((1 \rightarrow 2 \rightarrow 3 \rightarrow 4)\) and \((1 \rightarrow 3 \rightarrow 4)\).

**Definition 20** Let \( M \) be a BSS machine over the reals with input set \( Y \subseteq \mathbb{R}^\infty \).

(i) The partial function \( \phi_M : Y \rightarrow \mathbb{R}^\infty \) which is obtained when performing the computation of \( M \) on an input \( y \) and taking the output as the result, is called the function computed by \( M \).

(ii) A partial function \( f : Y \rightarrow \mathbb{R}^\infty \) is computable by a BSS machine iff there exists a BSS machine computing it.

(iii) The set \( \Omega_M \) of all \( y \in Y \) for which \( \phi_M(y) \) is defined is called the halting set of machine \( M \). It is also called the language accepted by \( M \). The image \( \phi_M(Y) \) of \( \phi_M \) is the output set of \( M \).

**Definition 21** Let \( M \) be a BSS machine over the reals. A computation path of \( M \) is called a **linear computation path** (in a set of register variables) if all the computation instructions along the path are linear (in these variables).
Example 22 (Continued) For the BSS machine $M$ shown in the previous example:

- $Y = \{y \in \mathbb{R}^\infty | \forall i (i > 0 \rightarrow y_i = 0)\}$, i.e. the machine’s input set $Y$ is all the inputs for which the contents of all but the first register contain the value 0.

- $\phi_M((x_1, 0, ...)) = \begin{cases} (x_1, 2.5 \times x_1, 2.5 \times x_1 + 5, 0, ...) & \text{if } x_1 \geq 0 \\ (x_1, 2.5 \times x_1, 2.5 \times x_1 + 5, x_1 \times x_1, 0, ...) & \text{otherwise} \end{cases}$

- Since $M$’s computation would halt for any input $y \in Y$, $\Omega_M = Y$.

- The computation path $(1 \rightarrow 3 \rightarrow 4)$ is a linear computation path in $x_1$ whereas the computation path $(1 \rightarrow 2 \rightarrow 3 \rightarrow 4)$ is not a linear computation path in $x_1$ due to the second instruction.

3.2 The Path Decomposition Theorem For the BSS Model

The following theorem characterizes the BSS machines input sets, halting sets and computed functions. We present a more detailed proof of the theorem than the proof presented in [MM02], as a preparation for later sections.

Notations:

For a BSS-Machine $M$, let $\gamma$ denote one computation path of $M$. Then the set $V_\gamma = \{x \in \Omega_M \cap \mathbb{R}^\infty | M$ on input $x$ performs precisely the instructions given by $\gamma\}$ is the input set computed along $\gamma$.

Theorem 23 (Path Decomposition) Let $M$ be a BSS machine with halting set $\Omega_M \subseteq \mathbb{R}^\infty$.

a) There exist at most countable many computation paths for $M$

b) For each computation path $\gamma$ of $M$, $V_\gamma$ is a semi-algebraic subset of one of the spaces $\mathbb{R}^m, m \in \mathbb{N}$. The function $\phi_M$ computed by $M$, when restricted to a path set $V_\gamma$, is a rational function.

c) $\Omega_M = \bigcup_{\gamma \text{path of } M} V_\gamma$, i.e. each halting set is a disjoint, countable union of semi-algebraic sets.

Proof:

a) Since the instructions set is finite, and since each instruction has at most two successor instructions, for each length $l$, there are a finite number of paths of length $l$ (at most $2^l$). Therefore, we can count $M$’s computation paths by ordering them according to length (first all the length 1 paths, then paths of length 2, etc.). Therefore there exist at most countable many computation paths for $M$. 

17
b) In order to prove this section we will use the same method used for proving program correctness in Floyd’s Proof Systems. For each computation path \( \gamma \) we define:

- \( R_\gamma (x) \) is a (quantifier free) formula representing the condition on the input \( x \) for the machine to follow the computation path \( \gamma \). Hence, \( V_\gamma \equiv \{ x \mid R_\gamma (x) \} \).
- \( T_\gamma (x) \) is the transformation function of input \( x \), i.e. the output value for input value \( x \) when \( M \) follows \( \gamma \), i.e. it is the function \( \phi_M(x) \) when restricted to a computation path \( \gamma \).

The above functions are obtained by backtracing the instructions along \( \gamma \), in an inductive manner, as follows:

Enumerate the instructions on \( \gamma \) from 0 to \( k \) according to the order of their execution. Let \( R_k^\gamma \) be the condition for \( M \) to follow the path from the \( i \)th instruction to the \( k \)th, and \( T_k^\gamma \) be the transformation function along the path from the \( i \)th instruction to the \( k \)th. Hence, \( R_k^\gamma = R_k^\circ \) and \( T_k^\gamma = T_k^\circ \). We start with: \( R_0^\circ (x) = \text{true} \) and \( T_0^\circ (x) = x \) (the identity function). Then the functions are computed inductively by tracing the path backwards. Each \( R_m^\gamma \), \( T_m^\gamma \) are then computed according to the type of the \( m \)th instruction:

- **computation:**
  \[
  R_m^\gamma (x) = R_{m+1}^\gamma (x)[x_s \leftarrow x_k \circ_n x_i] \quad \text{and} \\
  T_m^\gamma (x) = T_{m+1}^\gamma (x)[x_s \leftarrow x_k \circ_n x_i] \quad \text{or} \\
  R_m^\gamma (x) = R_{m+1}^\gamma (x)[x_s \leftarrow \alpha] \quad \text{and} \\
  T_m^\gamma (x) = T_{m+1}^\gamma (x)[x_s \leftarrow \alpha].
  \]

- **positive branch:** (i.e. the \( m \)th instruction is a branch instruction and the \( m + 1 \) instruction is the one in the goto section of the branch):
  \[
  R_m^\gamma (x) = R_{m+1}^\gamma (x) \land (x_1 \geq 0) \quad \text{and} \\
  T_m^\gamma (x) = T_{m+1}^\gamma (x).
  \]

- **negative branch:** (i.e. the \( m \)th instruction is a branch instruction and the \( m + 1 \) instruction is the one in the else goto section of the branch):
  \[
  R_m^\gamma (x) = R_{m+1}^\gamma (x) \land \neg(x_1 \geq 0) \quad \text{and} \\
  T_m^\gamma (x) = T_{m+1}^\gamma (x).
  \]

- **copy:**
  \[
  R_m^\gamma (x) = R_{m+1}^\gamma (x)[x_i \leftarrow x_j] \quad \text{and} \\
  T_m^\gamma (x) = T_{m+1}^\gamma (x)[x_i \leftarrow x_j].
  \]

From this construction we can see that \( T_\gamma \) is a rational function, because we start with the identity function which is a rational function and it remains a rational function after each manipulation; Each branch instruction leaves it unchanged; For each copy instruction we replace one variable with another; And for each computation instruction, since all the expressions we are assigning to the corresponding variable are themselves rational
functions, then the function remains rational (a composition of rational functions is a rational function).

Looking at the construction of $R_{\gamma}$, we can see that it is of the form: $\land (T_{\gamma^*}(x_{i}) \Delta 0)$, where $\Delta \in \{\geq, <\}$ (depending on whether the branch instruction is positive or negative) and each $\gamma^*$ is the partial path to $\gamma$ starting from the first instruction and ending at the branch instruction for which this condition was added to the formula. Since we have shown that $T$ is a rational function, then each such condition in $R_{\gamma}$ can be brought to the form of $p(x) \Delta 0$ where $\Delta \in \{\geq, <\}$ and $p$ is polynomial. Therefore, $R_{\gamma}$ is a boolean combination of atomic formulas of the form $p(x) \Delta 0$, and therefore $V_{\gamma} = \{x | R_{\gamma}(x)\}$ is semi-algebraic. Also, $V_{\gamma}$ is a subset of the space $\mathbb{R}^m$ where $m$ is the maximal dimension of the machine’s input (i.e. the number of non-zero elements in the initial registers contents). The existence of such a maximal dimension is guaranteed because since the formula $R_{\gamma}$ is finite it can only refer to a finite number of indices in $x$.

c) Each input in the union of $V_{\gamma}$ belongs to one of the $V_{\gamma'}$ and therefore also belongs to $\Omega_M$ by definition.

For each input $x$ in $\Omega_M$, $M$ halts on input $x$. Therefore there is a computation path $\gamma$ which $M$ follows on input $x$. Therefore $x \in V_{\gamma}$, and therefore also belongs to the union of $V_{\gamma}$.

\[ \square \]

**Observation 24** Looking at the constructions presented in section b) of the proof we can see that for linear computation paths $\gamma$ (in a set of variables), the representation of the set $V_{\gamma}$, and the computed function $\Phi_M$ are also linear in these variables.

### 3.3 The Eaves-Rothblum Computation Model (ER)

The computation model to be presented in this section is based on the BSS model of computation presented in the previous sections, and is inspired by the model presented in the Eaves-Rothblum preprint [ER99].

We start introducing our model, by defining an **ER-Machine** to be a BSS machine with the following modifications:

- For convenience, we add input and output variables, $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$. The input variables consist of the non-zero components of the machine’s input (which are usually copied into the Registers, but we will refer to them as being stored in a different set of registers - **Input Registers**). The output variables consist of the non-zero components of the machine’s output (or a significant subset of them can be defined, and the values copied from the relevant Registers into these variables when the machine halts).
• Add handling for division by 0, by replacing the division operator by the following computation instruction:

\[(\cdot)^{-1} : n : x_s \leftarrow (x_l)^{-1}\]

The register \(x_s\) will get the value of \((x_l)^{-1}\) if \(x_l \neq 0\) and 0 otherwise. All other register entries remain unchanged. The next instruction will be \(n + 1\).

• We add the following instructions:

  - **interval-pick** :
    
    \[n : x_s \leftarrow \text{choose } x \text{ with } (x_l \leq x \leq x_k) \text{ or } x_s \leftarrow \text{choose } x \text{ with } (\alpha \leq x \leq \beta) \text{ for some constants } \alpha, \beta \in \mathbb{R} \cup \{\pm \infty\}.\]
    The register \(x_s\) will get a value within the corresponding interval. All other register entries remain unchanged. The next instruction will be \(n+1\). Although the intervals were described with \(\leq\), both open and closed intervals, and therefore also singleton intervals, can be used as well.

  - **discrete-pick** :
    
    \[n : x_s \leftarrow \text{choose } x \text{ with } x \in \{x_{l_1}, \ldots, x_{l_m}\}\]
    The register \(x_s\) will get one of the values in \(\{x_{l_1}, \ldots, x_{l_m}\}\). All other register entries remain unchanged. The next instruction will be \(n+1\).

• We limit our discussion to BSS Machines whose induced graph is directed and acyclic (i.e. straight line programs). Similar to *Boolean Circuits*, this subset of BSS machines can be defined as *BSS Circuits* (or *ER Circuits*).

• The machine will be allowed to announce “No Output” by assigning a special value (an additional character or value not in \(\mathbb{R}\)) to the output variables.

**Observation 25** *Adding the interval-pick instructions to our model, has over-strengthened the model.* Consider the following machine for example:

1: \(x_1 \leftarrow \text{choose } x \text{ with } (0 \leq x \leq 100)\)
2: \(\text{if } x_1 \geq 50 \text{ goto 4 else goto 3}\)
3: \(x_2 \leftarrow 2.5 \times x_1\)
4: \(x_2 \leftarrow x_2 + 5\)

Now let \(\gamma_1 = (1 \rightarrow 2 \rightarrow 3 \rightarrow 4), \gamma_2 = (1 \rightarrow 2 \rightarrow 4)\). Then this time, \(V_{\gamma_1}\) and \(V_{\gamma_2}\) are not disjoint sets.

Therefore we would like to limit the strength of the added interval-pick instructions, by limiting the use of register variables holding pick results within branch instructions.
Refining the ER model

**Definition 26** Tainted register variables will be defined recursively as follows:

- For each instruction
  \[ n: x \leftarrow \text{choose } x \text{ with } (x_i \leq x \leq x_k) \text{ or } x \leftarrow \text{choose } x \text{ with } (\alpha \leq x \leq \beta), \]
  \[ x_i \text{ is a tainted variable.} \]

- For each computation instruction
  \[ n: x \leftarrow x_k \circ_n x_1, \]
  \[ x_i \text{ is a tainted variable if } x_k \text{ is a tainted variable or } x_1 \text{ is a tainted variable.} \]

- For each copy instruction
  \[ n: x_i \leftarrow x_j \]
  \[ x_i \text{ is a tainted variable if } x_j \text{ is a tainted variable.} \]

Now we will add the following restriction to the ER-Model defined above:

- Tainted variables can not be used in branch instructions.

Note that with this restriction, the machine presented in example 4 in subsection 1.4 is not valid in our model. However, we can overcome this issue by replacing the dependency of the computation path on the chosen value by a dependency of the calculation on the chosen value. In some cases utilizing the copy instructions as a form of indirect addressing would also help overcome this issue. In this example, an equivalent valid machine would be:

Take \( a, b, c \) as input.
If \( \neg((a = 0 \land b = 0 \land c = 0) \lor (a = 0 \land b \neq 0) \lor (a \neq 0 \land b^2 - 4ac \geq 0)) \) then fire “No Solution”.
Otherwise if \( a = 0 \land b = 0 \land c = 0 \) output choose \( z \) with \(-\infty \leq z \leq \infty\)
Otherwise if \( a = 0 \) output \(-\frac{b}{c}\)
Otherwise,
\[ z \leftarrow \text{choose } z \text{ with } z \in \{-1, 1\} \]
output \( \frac{b + z \sqrt{b^2 - 4ac}}{2a} \).

**Definition 27** Let \( M \) be an ER machine over the reals with input set \( Y \subseteq \mathbb{R}^\infty \).

(i) For such a modified BSS machine \( M \), the function computed by \( M \) is defined by the partial function \( \Phi_M : Y \rightarrow 2^{\mathbb{R}^\infty} \) which is obtained when performing the computation of \( M \) on an input \( y \) and taking all the possible outputs as the result set (according to all the possible values obtained after pick instructions).

(ii) \((x, y)\) is an Input-Output pair, if \( y \in \Phi_M(x) \), i.e. given the input \( x \), there exists an execution for which the relevant part of the machine’s output will be \( y \).
Example 28 An example for an ER machine $M$ can be given by the following set of instructions:

1. if $x_1 \geq 0$ goto 3 else goto 2
2. $x_4 \leftarrow$ choose $x$ with $(0 \leq x \leq 10.6)$
3. $x_2 \leftarrow 2.5 \times x_1$
4. $x_3 \leftarrow x_2 + 5$

We can see that this machine has two computation paths which the computation can follow, depending on the result of the branch condition in the first instruction. The two possible paths would be: $(1 \rightarrow 2 \rightarrow 3 \rightarrow 4)$ and $(1 \rightarrow 3 \rightarrow 4)$. The input set $Y$ is the set of all inputs with a single non-zero value (at the first location).

The function computed by this machine:

$$\phi_M((x_1,0,\ldots)) = \begin{cases} 
\{(x_1,2.5 \times x_1,2.5 \times x_1 + 5,0,\ldots)\} & \text{if } x_1 \geq 0 \\
\{(x_1,2.5 \times x_1,2.5 \times x_1 + 5,y,0,\ldots)\} & \text{otherwise}
\end{cases}$$

Examples for input-output pairs would be:

- $((4,0,\ldots),(4,10,15,0,\ldots))$
- $((-4,0,\ldots),(-4,-10,-5,3.4,0,\ldots))$
- $((-4,0,\ldots),(-4,-10,-5,9.8,0,\ldots))$

3.4 The Path Decomposition Theorem For the ER Model

As for the BSS Model, the following theorem characterizes the ER machines input sets and computed functions. However, since for the ER model we are limiting the discussion to ER Circuits, we will not be concerned with the machines halting sets, since for this subset of programs the machines always halt. We will use the same notations as before.

Theorem 29 (Path Decomposition) Let $M$ be an ER-Machine.

a) There exist only finitely many computation paths for $M$.

b) Each $V_\gamma$ is a semi-algebraic subset of the space $R^n$ (where $m$ is $M$’s input dimension). For each input $x$ in $V_\gamma$, $\Phi_M(x)$ is either a semi-algebraic set, or the special "No Output" value (Furthermore, if $\Phi_M(x)$ equals the "No Output" value for some $x$ in $V_\gamma$, then for every $x$ in $V_\gamma$, $\Phi_M(x)$ equals the "No Output" value since the same computation path is followed).

Proof:
a) The finite number of computation paths follows from the restriction to ER circuits.

b) We use the definitions and construction of $R_{\gamma}$ and $T_{\gamma}$ as shown in the path decomposition theorem for the BSS model. However, this time $T_{\gamma}$ will represent a set of possible output values instead of a single value. Actually, $T_{\gamma}$ will be a formula describing the output set for an input $x$, i.e. $\phi_M(x)$ when restricted to $\gamma$ will be $\{y T_{\gamma}(y)\}$. We need to add to the inductive construction handling of pick instructions as follows:

$$R^n_{\gamma}(x) = R^{n+1}_{\gamma}(x)$$ and

$$T^n_{\gamma}(x) = T^{n+1}_{\gamma}(x) \land (x_s \geq x_t \land x_s \leq x_k)$$

for interval-pick instructions, and:

$$R^n_{\gamma}(x) = R^{n+1}_{\gamma}(x)$$ and

$$T^n_{\gamma}(x) = T^{n+1}_{\gamma}(x) \land (\forall i, x_s = x_{t_i})$$

for the discrete pick instruction.

Therefore, we can see that without the pick instructions $T_{\gamma}$ would be a rational function as before. The pick instructions make it a semi-algebraic set because as argumented for $R_{\gamma}$ in the BSS case, all conditions can be brought to the required normal form either with equalities (for the computation expressions) or inequalities (for the pick intervals). Under similar arguments $R_{\gamma}$ remains a semi-algebraic set for these machines as well.

Note that $R_{\gamma}$ is not affected by pick instructions, because we do not allow branch instructions to use tainted variables, and therefore $R_{\gamma}$ will not contain terms containing $x_s$.

\[\Box\]

**Observation 30** Looking at the constructions presented in section b) of the proof we can see that again for linear computation paths $\gamma$ (in a set of variables), the representations of the set $V_{\gamma}$, and of the computed function $\Phi_M$ are also linear in these variables.

### 3.5 Possible Variations over the ER Computation Model

When defining the ER model we defined a new pick instruction for the machine, at which a value is uniformly chosen from an interval (or a singleton set) and is assigned to the required register.

However, we can consider using other Choice Operators as alternative pick instructions for this model. For example:

- **linear - eq - pick** \( x \leftarrow \text{choose } z \text{ with } Mz = b \)
- **linear - ineq - pick** \( x \leftarrow \text{choose } z \text{ with } Mz \leq b \)
- **algebraic - pick** \( x \leftarrow \text{choose } z \text{ with } \bigwedge_i V_j \varphi_j(z) \Delta 0, \Delta \in \{=, <, \leq, >, \geq\} \)
- **witness - pick** \( x \leftarrow \text{choose } z \text{ with } \varphi(z) \)
The third operator is a generalization of the other choice operators defined. The last operator is the most general form of choice operator. These operators can either be defined over vectors or single variable values.

From the proof of the path decomposition theorem presented in the last subsection, we can see that the theorem would still hold if we used the linear-ineq-pick or linear-eq-pick operators instead of the interval-pick instruction. (We would only need to replace the condition \((x_s \geq x_0 \land x_s \leq x_k)\) by \((Mx \leq b)\) or \((Mx = b)\) accordingly which are also polynomial conditions).

In later sections (4.4, 5.6) we will discuss questions such as how using different choice operators affects the power of the computation model (i.e. whether a powerful choice operator can be implemented using a simpler one and computation instructions).
4 Algorithms and Linear Algorithms

An Algorithm $A$ is a tuple $(A, x, y)$ where:

- $A$ is an ER-Machine as defined in the previous section.
- $x$ is the predefined ordered tuple of the machine’s input which is copied to the Input Registers at the beginning of the machine’s execution.
- $y$ is the predefined ordered tuple of the machine’s output which is copied from the Registers at the end of the machine’s execution.

Linear Algorithms:
An Algorithm $(A, x, y)$ is a Linear Algorithm if all of the machine’s computation paths are linear computation paths in the set of Registers variables $(R_i)$ and in the output variables.

Example 31 An algorithm for solving a linear equation: $ax + b = c$: $(M, (a, b, c), x)$.

$M$: Input $a, b, c$ into $IR_1, IR_2, IR_3$

if $IR_1 = 0 \land (IR_3 - IR_2) \neq 0$ Output “No output”

if $IR_1 = 0 \land (IR_3 - IR_2) = 0$ Output choose $z$ with $-\infty \leq z \leq \infty$ into $x$

$R_1 \leftarrow IR_3 - IR_2$

$R_2 \leftarrow R_1 \div IR_1$

Output $R_2$ into $x$

This is a linear algorithm since all the instructions are linear in $x$ and in the register variables $(R_1, R_2)$.

Example 32 For the machine $A$:

\[
\begin{align*}
&i: y \leftarrow x \cdot R_i \\
&\ldots
\end{align*}
\]

$(A, (x), (y))$ is a linear algorithm

$(A, (x), (x, y))$ is not a linear algorithm

4.1 Linear Problems Vs. Linear Algorithms

Problems and Algorithms Equivalency:

Linear Problems are Equivalent if they have the same data - solution pairs.

Linear Algorithms are Equivalent if they have the same input - output pairs.

Matching Algorithms and Problems: A problem and an algorithm are defined to match if the data and solution variables of the problem coincide with the input and output variables of the algorithm respectively.

A Problem Solved By an Algorithm: A Problem $P$ is solved by an algorithm $A$ if $P$ and $A$ match, and for every data assignment $\bar{x}$ in $\mathbb{R}$ of $P$, and every execution of $(A, \bar{x}, y)$ either
a) The output value \( \tilde{y} \) is valid (not the "No Output" special value) and \( \tilde{y} \) solves \((P, \tilde{x}, y)\) over \( \mathbb{R} \), or

b) The machine outputs the special "No Output" value and \((P, \tilde{x}, y)\) over \( \mathbb{R} \) has no solution.

**The function Completely Solved By**

\((P, x, y)\) is *completely solved by* \((A, x, y)\) if \(P\) is solved over by \(A\) and for every input assignment \(\tilde{x}\), \(\Phi_A(\tilde{x}) = \text{The Solution Set of } (P, \tilde{x}, y)\).

If \(P\) is completely solved by \(A\), then the set of data - solution pairs of \(P\) over \( \mathbb{R} \) equals the set of input - output pairs of \(A\) over \( \mathbb{R} \).

**Observation 33** The relation *Completely Solved By on problem - algorithm pairs*, is invariant over equivalence of problems and equivalence of algorithms. I.e., if a problem is completely solved by an algorithm \(A\), then all equivalent problems are also solved by \(A\), and all equivalent algorithms \(A'\) completely solve the problem.

This immediately follows from the definitions due to the equality of the data - solution pairs and the input - output pairs sets when a problem is completely solved by an algorithm, and the equality of these sets between equivalent problems and equivalent algorithms.

### 4.2 Problem Construction Given an Algorithm

**Theorem 34** For each linear algorithm \(A\), there exists a linear problem \(P\), which is completely solved by \(A\).

**Proof:**

Given a linear algorithm \((A, \tilde{x}, \tilde{y})\), we construct a linear problem \((P, \tilde{x}, \tilde{y})\).

The set \(\{(\tilde{x}, \tilde{y})|\tilde{y} \in \Phi_A(\tilde{x})\}\) in \(\mathbb{R}^{n+m}\) is the set of \(A\)'s input-output pairs. We need to show that it is FOL definable and therefore a problem which is completely solved by \(A\) (because of the sets equality). We also need to show that it has a linear representation.

For every computation path \(\gamma\) in \(A\) we construct a formula \(p_\gamma\):

For each computation path \(\gamma\) in \(A\), \(V_\gamma\) is semi-algebraic according to the path decomposition theorem (see section 3.4).

Therefore it is definable by a quantifier-free FOL formula, \(f_{V_\gamma}(\tilde{x})\).

Also, for each path \(\gamma\), the set

\[
\{\tilde{y}|\tilde{y} \in \Phi_A(\tilde{x}) \text{ and } \tilde{x} \in V_\gamma\}
\]

is semi-algebraic, and therefore definable by a quantifier-free FOL formula, \(f_{\Phi_A(\tilde{x})}(\tilde{x}, \tilde{y})\).

Next, define:

\[
p_\gamma(\tilde{x}, \tilde{y}) = f_{V_\gamma}(\tilde{x}) \land f_{\Phi_A(\tilde{x})}(\tilde{x}, \tilde{y})
\]

26
Now, 
\[ P(\bar{x}, \bar{y}) = \bigvee_{\gamma} p_\gamma(\bar{x}, \bar{y}) \]

Since the number of computation paths \( \gamma \) is finite according to the path decomposition theorem, \( P \) is well defined.

According to the path decomposition theorem, since the sets \( V_\gamma \) are a partition of the machine’s input, then the sets \( (V_\gamma, \Phi_A|V_\gamma) \) fully describe the machine’s input-output pairs. Therefore, \( P(x \leftarrow \bar{x}, y \leftarrow \bar{y}) \) is true iff \( (\bar{x}, \bar{y}) \) is an input-output pair for \( A \). Therefore, \( P \) is completely solved by \( A \).

Also, we note that according to observation 30, for a Linear Algorithm’s machine, all the polynomials describing the semi-algebraic sets of \( V_\gamma \) and \( \Phi_M(\bar{x}) \) are linear in the registers variables and in the output variables, since all the computation paths are linear computation paths in these variables.

Therefore, each \( p_\gamma \) is linear in the set of output variables, therefore \( P \) is linear in its solution variables. Since \( P \) is unquantified, \( (P, x, y) \) is a linear problem.

\( \square \)

**Example 35** For the following algorithm:

\[ 
\begin{align*}
\text{Input } x \\
\text{if } x > 0 \text{ then} \\
\quad \text{yes} & \text{ no} \\
\quad r & \leftarrow x \cdot x \\
\quad y & \leftarrow x \cdot x \\
\quad \text{choose } z \text{ with } (0 \leq z \leq r) \\
\text{output } y
\end{align*}
\]

The constructed formula will be:

\[ P = ((x > 0) \land (y = x^2)) \lor ((x \leq 0) \land (0 \leq y \leq x^3)) \]

And \((P, x, y)\) is a linear problem which is completely solved by the algorithm.

### 4.3 Algorithm Construction Given a Problem

**Theorem 36** For each linear problem \( P \), there exists a linear algorithm \( A \), such that \( P \) is completely solved by \( A \).

**Proof:**
Given a linear problem \((P, \bar{x}, \bar{y})\), we construct a linear algorithm \((A, \bar{x}, \bar{y})\).

First we construct a quantifier free formula \( P' \) which is equivalent to \( P \), using the linear Quantifier Elimination process as shown in Subsection 2.4.2. As shown, \( P' \) is also linear in the solution variables, therefore \((P', \bar{x}, \bar{y})\) is a linear problem which is equivalent to \((P, \bar{x}, \bar{y})\).

Next we will construct a linear algorithm which completely solves \((P', \bar{x}, \bar{y})\), and therefore completely solves \((P, \bar{x}, \bar{y})\) according to Observation 33 in Subsection 4.1.
$P'$ is of the form $\text{BooleanCombination}(f_{ij}(\bar{x}, \bar{y}) \geq 0)$ where each $f_{ij}$ is linear in $\bar{y}$, i.e. it can be brought to the form:

$$P' = \bigwedge_{i} \bigvee_{j} f_{ij}(\bar{x}, \bar{y})\Delta 0, \Delta \in \{\geq, =\}$$

The following algorithm construction uses the Simplex algorithm for solving linear programs, as presented in [PS98]. An outline of the algorithm for solving $(P', \bar{x}, \bar{y})$ would be:

- Use discrete-pick to choose one of the clause in $P' - C_i$.
- Use Simplex algorithm to solve the linear program $\text{min}(y_1)$ such that $C_i$.
- Use Simplex algorithm to solve the linear program $\text{max}(y_1)$ such that $C_i$.
- $R_1 \leftarrow$ choose $z$ with $(\text{min}(y_1) \leq z \leq \text{max}(y_1))$ (pick a value from the interval and assign to $R_1$). Any value in this interval is a valid solution value for $y_1$ since the solution space of a linear program is a polyhedron and therefore there aren’t any discontinuities, i.e. $y_1$ values within this interval for which there is no solution to $C_i$.
- Use Simplex algorithm to solve the linear program $\text{min}(y_2)$ such that $C_i(y_1 \leftarrow R_1)$.
- Use Simplex algorithm to solve the linear program $\text{max}(y_2)$ such that $C_i(y_1 \leftarrow R_1)$.
- $R_2 \leftarrow$ choose $z$ with $(\text{min}(y_2) \leq z \leq \text{max}(y_2))$.
- Again, as explained above, any value in this interval is a valid solution value for $y_2$.
- Repeat the above for each coordinate of $\bar{y}$.
- For each $i$, $y_i \leftarrow R_i$.

This algorithm completely solves $(P', \bar{x}, \bar{y})$ since according to the result of the pick instructions, for each value of $x$, each possible solution of the clause $C_i$ is in the output set of the algorithm. Also, according to the result of the first discrete-pick instruction, each clause of $P'$ can be chosen to be satisfied, therefore each possible solution of $P'$ is in the output set of the algorithm. Note that there are no branch instructions which depend on the results of pick instructions, and therefore the constructed algorithm is valid in our model. What is left is to show that the Simplex algorithm is linear in $\bar{y}$ in order to prove that the above algorithm is a linear algorithm.

**An Outline of the Simplex Algorithm:**

The Simplex algorithm gets a linear system as an input (min $c \cdot x$ such that $Ax = b, x \geq 0$ is the standard representation. Any system can be transformed
to an equivalent system in the standard representation. The transformation is
linear in \( x \), and finds the optimal solution.

The first stage of the Simplex algorithm consists of finding some basic feasible
solution (bfs) of the linear system. This is done by transforming the linear
system into a different system for which finding a bfs is easy and for which an
optimal solution is a bfs of the original system. I.e. we first find the optimum
for this system and start with that as an initial bfs for finding the optimum of
the original system. The transformation is simply adding a variable for each
constraint (matrix row) of the original system, and therefore a linear operation.

The core of the Simplex algorithm is the pivoting step which obtains a new
bfs out of the previous one until the optimum is reached (the solution to the
system is also a bfs).

The pivoting operation starts with a bfs \( x \) with basis \( B \) (of the constraints
matrix \( A \)), and an index \( j \) not in \( B \) and outputs a new bfs \( x' \) with a new basis
\( B' \), as follows:

1. Compute the coefficients \( y_k \) such that:

\[
A_j = \sum_{k=1}^{m} y_k \cdot A_{B(k)}
\]

Note that the coefficients do not depend on \( x \), but only on the matrix \( A \),
therefore this step is linear in \( x \).

2. Compute the value of \( \theta \):

\[
\theta = \min_{k, y_k \geq 0} \left\{ \frac{x_B(k)}{y_k} \right\}
\]

Again since \( y_k \) do not depend on \( x \), this step is linear in \( x \).

3. Compute the value of \( x' \) and the new basis \( B' \):

\[
x'_l = \begin{cases} 
\theta & l = j \\
0 & l \neq j, l \text{ not in the basis } B \\
x_{B(k)} - \theta y_k & l = B(k) 
\end{cases}
\]

The new basis: let \( i \) be some index such that \( y_i > 0 \) and \( \theta = \frac{x_B(i)}{y_i} \). Then
\( A_j \) enters the basis instead of \( A_{B(i)} \):

\[
B'(k) = \begin{cases} 
B(k) & k \neq i \\
j & k = i
\end{cases}
\]

Again we see that this stage is linear in \( x \).

Therefore the pivoting stage is linear in \( x \). Therefore the Simplex algorithm is
linear in \( x \). In our case all the usage of Simplex will be linear in \( y \) and therefore
the algorithm constructed above is a linear algorithm.
The time complexity for the Simplex algorithm is in the worst case exponential in the size of the system, although it is known to be efficient in practice. Other known algorithms for solving linear programs, such as the Ellipsoid method and Karmarkar’s method may improve the complexity but are not linear so they could not be used in this case. □

4.4 Choice Operators for the ER-Model (Revisited)

If we take another look at the algorithm construction presented in the previous subsection, we can see that the construction very much depends on the choice operator which was chosen for the model. For instance, if our model had used the linear-ineq-pick instruction instead of the interval-pick instruction then the algorithm construction would be very simple:

\[ y \leftarrow \text{choose } z \text{ with } P'(x, z) \]

Since we have shown that an algorithm can be constructed using the interval-pick instruction, then this means that the linear-ineq-pick choice operator can be implemented by means of the interval-pick operator, using the Simplex algorithm. Given an algorithm (ER-Machine) which uses linear-ineq-pick instructions, we can directly construct an algorithm which uses only interval-pick instruction, by replacing each linear-ineq-pick instruction by the subroutine described in the previous proof as the constructed algorithm’s outline. Therefore using linear-ineq-pick instructions would not add expressive power to our model.
5 Generalizing the Computational Model - Polynomial Problems and Algorithms

Under the definitions of the previous sections, let us now consider problems and algorithms, i.e., without the restrictions of linearity, which we will now refer to as *polynomial problems and algorithms* respectively. Is there a polynomial algorithm $A$ for each polynomial problem $P$, so that $P$ is completely solved by $A$? Is there a polynomial problem $P$ for each polynomial algorithm $A$, so that $A$ completely solves $P$?

It is easy to see that the answer to the second question is positive. The same construction using the path decomposition theorem shown in the previous section (4) will apply in this case as well, and the problem polynomiality is immediate from the path decomposition theorem.

However, the first question is more complicated. For the first part of the construction - finding an equivalent problem which is quantifier free, instead of the Linear Quantifier Elimination method introduced in the previous sections, we can use the algorithm of quantifier elimination by cylindrical algebraic decomposition, as introduced in section 2.4.3. For constructing an algorithm, given a quantifier free polynomial problem, the construction shown in section 4.3 will not hold in this case since it highly relies on the problem’s linearity in its solution variables (the problem is viewed as a linear program and the Simplex algorithm for solving linear programs is used in the construction).

In fact, the answer to this question is negative. Using the ER computation model presented in section 3.3, not for every polynomial problem there is a polynomial algorithm which completely solves it. Consider the following polynomial problem for example: $((y^2 = x), x, y)$. Now assume for contradiction that there exists a polynomial algorithm which completely solves it. Then the algorithm’s ER machine computes the function $x \rightarrow \sqrt{x}$. In [MM02, p. 12] it is proved that this function is not BSS computable. The same arguments which are used in the proof would still hold for ER machines since they are not affected by the additional non-determinism of the *pick* instructions.

Therefore, we will have to further enhance our model in order to achieve a generalization to non-linear problems and algorithms.

The enhancement will consist of adding “black box” operators as computation instructions, according to the skolem functions for $Th(\mathbb{R})$, as explained below.

5.1 Skolem Functions

Skolem functions are used for eliminating existential quantifiers to produce a formula equivalent to the original. Skolemization is an application of the (second-order) equisatisfiability

$$\forall x \exists y R(x, y) \iff \exists f \forall x R(x, f(x))$$

i.e., $\forall x \exists y R(x, y)$ is satisfiable iff there is a function $f(x)$ such that $\forall x R(x, f(x))$ is satisfiable. This holds due to the observation that if a formula in the form
\( \forall x \exists_2 y R(x, y) \) is satisfiable in some model, then there must be some point in the model for every \( x \) which makes \( R(x, y) \) true, and using the axiom of choice there exists some function \( y = f(x) \) which makes the formula \( \forall x R(x, f(x)) \) true. The function \( f \) is called a **Skolem Function**.

Skolem functions are used in second order logic to move all existential quantifiers outside the scope of first order universal quantifiers. This can be done since second order quantifiers can quantify over functions.

### 5.2 Using Skolem Functions for \( Th(\mathbb{R}) \)

Let \( \mathbb{R} = (\mathbb{R}, +, \cdot, <, 0, 1, -) \).

Assume we have \( \varphi(\vec{x}, y) \) such that

\[
\mathbb{R} \models \forall \vec{x} \exists y \varphi(\vec{x}, y)
\]

or equivalently,

\[
RCF \models \forall \vec{x} \exists y \varphi(\vec{x}, y)
\]

Since \( \mathbb{R} \) admits quantifier elimination we can assume without loss of generality that \( \varphi(\vec{x}, y) \) is quantifier free, and therefore, without loss of generality, it suffices to discuss the case of a single existentially quantified variable.

**Example 37**

For the formula

\[
\forall x_1 \forall x_2 \forall x_3 \exists y_1 \exists y_2 (y_1 y_2 = x_1 x_2 x_3)
\]

we can consider the equivalent formula

\[
\forall x_1 \forall x_2 \forall x_3 \exists y_1 (\neg (x_1 \neq 0 \land x_2 \neq 0 \land x_3 \neq 0 \land y_1 = 0))
\]

which has only one existentially quantified variable.

We define a function \( \mathbb{R}^n \to \mathbb{R} \) (where \( n \) is the arity of \( \vec{x} \)) in the following way: \( f_\varphi(\vec{a}) \mapsto \beta \) such that \( \mathbb{R} = \varphi(\vec{a}, \beta) \), and \( \beta \) is uniquely defined as follows:

As mentioned above since \( \mathbb{R} \) admits quantifier elimination, we can assume without loss of generality that \( \varphi(\vec{x}, y) \) is quantifier free. Hence, \( \varphi(\vec{a}, y) \) is quantifier free and is of the following form:

\[
\varphi(\vec{a}, y) = \bigvee_{i} \bigwedge_{j} p_{ij}(\vec{a}, y) \Delta_{ij} 0
\]

where \( \Delta_{ij} \in \{<, =, \geq, \ldots \} \). Therefore the set \( D(\vec{a}) = \{ z | \varphi(\vec{a}, z) \} \) is a definable union of finitely many singletons (isolated points) and intervals \((a, b)\) where \(-\infty \leq a < b \leq +\infty\). If \( D(\vec{a}) \) has a least element we define \( \beta \) to be that element. Otherwise, take the lowest maximal component of \( D(\vec{a}) \), an interval \((a(\vec{a}), b(\vec{a}))\) such that \( a(\vec{a}) \in \mathbb{R} \cup \{-\infty\} \) and \( b(\vec{a}) \in \mathbb{R} \cup \{\infty\} \) and \((a(\vec{a}), b(\vec{a})) \subset D(\vec{a}) \). Next, \( \beta \) is chosen as follows:

(i) \( a(\vec{a}) = -\infty \) and \( b(\vec{a}) = +\infty \) then we choose 0.
(ii) \( a(\bar{a}) = -\infty \) and \( b(\bar{a}) \in \mathbb{R} \) then we choose \( b(\bar{a}) - 1 \).

(iii) \( a(\bar{a}) \in \mathbb{R} \) and \( b(\bar{a}) = +\infty \) then we choose \( a(\bar{a}) + 1 \).

(iv) \( a(\bar{a}) \in \mathbb{R} \) and \( b(\bar{a}) \in \mathbb{R} \) then we choose the midpoint \( (a(\bar{a}) + b(\bar{a}))/2 \).

The functions \( f_\varphi(\bar{x}) \) are called Skolem Functions.

Note that when \( \varphi \equiv \psi \) then \( f_\varphi(\bar{x}) = f_\psi(\bar{x}) \) (since we get the same set \( D \) for every value \( \bar{a} \) of \( \bar{x} \)).

Now, suppose we use these Skolem Functions to enhance our language, i.e. to create a “Fortran Library”, in the following recursive manner:

Let the initial vocabulary be \( \tau_0 = \tau_{RCF} \), and let the initial set of axioms be \( T_0 = T_{RCF} \). Next, define:

\[
\begin{align*}
\tau_{n+1} &= \text{Sk}_R(\tau_n) = \tau_0 \cup \{ f_\varphi[T_{RCF} \vdash \forall \bar{x} \exists y \varphi(\bar{x}, y)] \}, \\
T_{n+1} &= T_n \cup \{ \forall \bar{x} \varphi(\bar{x}, f_\varphi(\bar{x})) | \varphi \in \text{FOL}(\tau_0) \} \cup \{ \forall \bar{x} \forall y(\varphi(\bar{x}, y) \equiv \psi(\bar{x}, y) \rightarrow f_\varphi(\bar{x}) = f_\psi(\bar{x})) \}
\end{align*}
\]

(Where the axioms which are added to \( T_0 \) will be called Skolemization Axioms).

\[
\begin{align*}
\tau_* &= \bigcup_n \tau_n, \\
T_* &= \bigcup_n T_n.
\end{align*}
\]

We will see that using the above Skolem Functions added to our “Fortran Library” as additional instructions, will enable us to find an algorithm for solving each problem.

**Definition 38**

A Theory \( T \) admits definable Skolem Functions if for every formula \( \varphi \), such that \( T \vdash \forall \bar{x} \exists y \varphi(\bar{x}, y) \), there exists a definable function \( f \) for which \( T \vdash \forall \bar{x} \varphi(\bar{x}, f(\bar{x})) \). \( f \) is definable over \( T \) iff there is a term \( t(\bar{x}) \) such that \( T \vdash t(\bar{x}) = b \iff f(\bar{x}) = b \).

The above construction shows that \( Th(\mathbb{R}) \) admits definable Skolem Functions.

For the general case, in [Sco88], the following theorem is presented stating conditions under which a theory \( T \) admits definable Skolem Functions:

**Theorem 39** Let \( L \) be a first order language with at least one constant symbol \( c \). Let \( T \) be a model complete \( L \)-theory and \( A \) a set of \( \pi_2 \) sentences which axiomatizes \( T \). Assume that \( T \) Skolemizes \( A \), i.e. for every \( \forall \bar{x} \exists \bar{y} \phi(\bar{x}, \bar{y}) \) in \( A \) there is a definable function \( f \) for which \( T \vdash \forall \bar{x} \phi(\bar{x}, f(\bar{x})) \). Then \( T \) admits definable Skolem functions.

A model complete theory is a theory which admits quantifier elimination up to an existential prefix.

\( \pi_2 \) formulas are formulas of the form \( \forall \bar{x} \exists \bar{y} B(\bar{x}, \bar{y}) \), where \( B \) is a quantifier free formula.
Example 40 Consider the theory of dense order over $\mathbb{R}$. This theory has a $\pi_2$ axiom:

$$\forall x \forall y \exists z (x \neq y \rightarrow ((x < y \land x < z < y) \lor (y < x \land y < z < x)))$$

For the language $\langle \mathbb{R}, <, \text{mid} \rangle$ where mid is a function symbol representing $\mid x \mid = \frac{x + y}{2}$, then the axiom can be skolemized in the following way:

$$\forall x \forall y (x \neq y \rightarrow ((x < y \land x < \text{mid}(x, y) < y) \lor (y < x \land y < \text{mid}(x, y) < x)))$$

and therefore the theory of dense order over $\mathbb{R}$ with the function symbol mid admits definable Skolem functions.

This theorem is applied to show that $Th(\mathbb{R})$ admits definable Skolem Functions. Since the skolem functions are all constructed using the axioms' skolem functions it is enough to use these functions to enrich our model. We will use these results to further enhance our model.

5.3 Model Enhancements using Skolem Functions for $Th(\mathbb{R})$

The set of axioms which axiomatizes $Th(\mathbb{R})$ is obtained by adding to the universal axioms for commutative rings with nontrivial unit, the following formulas:

(i) $\forall x \exists y (x \neq 0 \rightarrow xy = 1)$

(ii) $\forall x_1 \forall x_2 \ldots \forall x_{2n+1} \exists y (x_{2n+1} = 0 \lor \sum_{i=0}^{2n+1} x_i y^i = 0)$ for $n \geq 1$.

(iii) $\forall x \exists y (x + y^2 = 0 \lor x = y^2)$

(iv) $\forall x_1 \ldots \forall x_n (\sum_{i=1}^{n} x_i^2 = 0 \rightarrow \bigwedge_{i=1}^{n} x_i = 0)$ for $n \geq 1$

(v) $\forall x \exists y (x + y = 0)$

The skolem function for axiom v is the unary $'-'$ operator which we already have as a part of our language (the computation instructions in our model are all binary but the unary $'-'$ operator can easily be implemented using $x_s \leftarrow 0 - x_i$). The following functions are the skolem functions for the rest of the $\pi_2$ axioms of $Th(\mathbb{R})$:

$$(\cdot)^1$$

$$x \rightarrow \begin{cases} x^1 & \text{if } x \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$\text{leastroot}_{2n+1}$

$$(x_0, \ldots, x_{2n+1}) \rightarrow \begin{cases} \text{the least root of } \sum_{i=0}^{2n+1} x_i y^i & \text{if } x_{2n+1} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

34
\[ x \rightarrow +\sqrt{|x|} \]

The \((\cdot)^1\) function corresponds to axiom (i), the \(\text{leastroot}_{2n+1}\) function corresponds to axiom (ii) and the \(+\sqrt{|\cdot|}\) function corresponds to axiom (iii).

Next we will define

\[ \mathbb{R}^* = \langle \mathbb{R}, +, -, \cdot, 0, 1, <, (\cdot)^1, \text{leastroot}_{2n+1}, +\sqrt{|\cdot|} \rangle = \tau_{RCF} \]

Denote the polynomial problems which were defined earlier as \(\mathbb{R}\) Problems. Now define,

**Definition 41** An \(\mathbb{R}^*\) Problem is a problem definable by a prenex formula \(\varphi(\bar{x}, \bar{y}) \in FOL(\tau_{RCF}^*)\). \(\bar{x}, \bar{y}\) are the problem’s data and solution variables accordingly.

We would like to show that this new theory, \(Th(\mathbb{R}^*)\), admits quantifier elimination. In order to do that we will first prove the following proposition, stating that every \(\mathbb{R}^*\) formula has an equivalent \(\mathbb{R}\) formula, which is also computable from the given formula. Thus since \(Th(\mathbb{R})\) admits quantifier elimination we will conclude that \(Th(\mathbb{R}^*)\) also admits quantifier elimination.

**Proposition 42** Every \(\mathbb{R}^*\) formula is also an \(\mathbb{R}\) formula. More precisely, for every \(\varphi \in FOL(\tau_{RCF}^*)\) there exists \(\psi \in FOL(\tau_{RCF})\) such that

(i) \(\varphi, \psi\) have the same free variables

(ii) \(RCF + \text{definitions of the Skolem functions} \models \forall \bar{x}\bar{y}(\varphi(\bar{x}, \bar{y}) \leftrightarrow \psi(\bar{x}, \bar{y}))\)

(iii) \(\psi\) is computable from \(\varphi\)

**Proof:** Given an \(\mathbb{R}^*\) formula, i.e. a formula \(\varphi \in FOL(\tau_{RCF}^*)\) we will recursively construct an equivalent formula \(\psi \in FOL(\tau_{RCF})\), i.e. an \(\mathbb{R}\) formula.

Let us consider the atomic subformulas of \(\varphi\). Without loss of generality, we can assume that each such subformula is of the form: \(P(\bar{x}) < 0\), where \(P\) is recursively constructed using \(\tau_{RCF}\)'s operators and function symbols.

Now, for each such subformula, going from inward to outward, according to the recursive construction of \(P\), each time perform one of the following as appropriate:

- For an appearance of \((g(\bar{x}))^1\) in \(P\), replace the subformula by

\[ \exists y [y : g(\bar{x}) = 1 \wedge P(\bar{x})^{g(\bar{x}) \downarrow -1} < 0] \]

where \(y\) is a new variable which does not appear in \(\varphi\) and \(P(\bar{x})^{g(\bar{x}) \downarrow -1}\) stands for \(P(\bar{x})\) after replacing each occurrence of \((g(\bar{x}))^1\) by \(y\).
• For an appearance of $\sqrt{|g(\overline{x})|}$ in $P$, replace the subformula by

$$\exists y[(g(\overline{x}) < 0 \land y^2 = -g(\overline{x})) \lor (g(\overline{x}) \geq 0 \land y^2 = g(\overline{x}))] \land P(\overline{x})^y_{\sqrt{|g(\overline{x})|}} < 0$$

where $y$ is a new variable which does not appear in $\varphi$ and $P(\overline{x})^y_{\sqrt{|g(\overline{x})|}}$ stands for $P(\overline{x})$ after replacing each occurrence of $\sqrt{|g(\overline{x})|}$ by $y$.

• For an appearance of $\text{leastroot}_{2n+1}(\overline{a})$, where each $a_i$ is some $g_k(\overline{x})$ in $P$, replace the subformula by

$$\exists y[\sum_{i=0}^{2n+1} a_i y^i = 0 \land$$

$$\forall z(\sum_{i=0}^{2n+1} a_i z^i = 0 \rightarrow z \geq y) \land$$

$$P(\overline{x})^y_{\text{leastroot}_{2n+1}(\overline{a})} < 0]$$

where $y$ is a new variable which does not appear in $\varphi$ and $P(\overline{x})^y_{\text{leastroot}_{2n+1}(\overline{a})}$ stands for $P(\overline{x})$ after replacing each occurrence of $\text{leastroot}_{2n+1}(\overline{a})$ by $y$.

Let $\psi$ be the resulting formula, after all the above replacements. We can see that $\varphi$’s free variables have been preserved, and no new free variables were added, therefore $\varphi$ and $\psi$ have the same free variables. These formulas are equivalent under the above theory by definition, since all the replacements are according to the functions definitions. The proof of the existence of an $\mathbb{R}$ formula $\psi$ equivalent to $\varphi$ was shown in a constructive manner, therefore $\psi$ is computable from $\varphi$. □

Note:
Each such replacement step linearly increases the formula’s size, therefore the size of $\psi$ is exponential in the size of $\varphi$.

Example 43
For the $\mathbb{R}^*$ formula: $\sqrt{|x|} + \text{leastroot}((1,0,1,-1)) < 0$, we get:

$$\sqrt{|x|} + \text{leastroot}((1,0,1,-1)) < 0 \equiv$$

$$\exists y[(x < 0 \land y^2 = -x) \lor (x \geq 0 \land y^2 = x)] \land y + \text{leastroot}((1,0,1,-1) < 0) \equiv$$

$$\exists y \exists z[((x < 0 \land y^2 = -x) \lor (x \geq 0 \land y^2 = x)) \land (z^3 + z - 1 = 0) \land w(w^3 + w - 1 = 0 \rightarrow w \geq z) \land y + z < 0]$$

Which is an equivalent $\mathbb{R}$ formula.

Corollary 44 $\text{Th}(\mathbb{R}^*)$ admits QE
Proof: Given a formula $\varphi \in FOL(\tau_{RCF})$, we can construct a formula $\psi \in FOL(\tau_{RCF})$, as shown in the proof of Proposition 42, such that $\psi$ is equivalent to $\varphi$. Now, since $RCF$ admits $QE$, we can construct a quantifier free formula equivalent to $\psi$, therefore also equivalent to $\varphi$ by transitivity.

As shown in section 2.4, this can be done in double exponential time in the size of $\psi$. Therefore, the upper bound of the time complexity of $QE$ for $Th(\mathbb{R}^*)$ is triple exponential. □

5.4 The $ER^*$ Model and $\mathbb{R}^*$ Algorithms

We will further enhance the ER-Model as was defined in Section 3. An $ER^*$-Machine is an ER machine with the following modifications:

- Remove the division operator ($\div$)
- Add the following computation instructions:
  - $(\) \uparrow : n : x_1 \leftarrow (x_1)^1$
    The register $x_1$ will get the function’s value (as defined above) for $x_1$. All other register entries remain unchanged. The next instruction will be $n + 1$. This instruction was actually already added to the ER model but is listed again here for completeness.
  - $+\sqrt{\cdot}\downarrow : n : x_1 \leftarrow +\sqrt{|x_1|}$ The register $x_1$ will get the function’s value (as defined above) for $x_1$. All other register entries remain unchanged. The next instruction will be $n + 1$.
  - $\text{leastroot}_{2m+1} : n : x_1 \leftarrow \text{leastroot}_{2m+1}(x_0, \ldots, x_{2m+1})$ The register $x_1$ will get the function’s value (as defined above) for $x_0, \ldots, x_{2m+1}$. All other register entries remain unchanged. The next instruction will be $n + 1$.

Note that the above operations use the skolem functions defined above as “black box” operators.

- Add the following pick instruction:
  root-pick-in-interval:
  $n : x_1 \leftarrow \text{choose } z \text{ with } (x_{b_1} < z < x_{b_2} \land \sum_{i=0}^{n} x_i, z^i = 0)$
  The register $x_1$ will get one of the polynomial’s roots within the given interval. All other register entries remain unchanged. The next instruction will be $n + 1$

An $\mathbb{R}^*$ Algorithm is an Algorithm using an $ER^*$-Machine.
5.4.1 The Path Decomposition Theorem for the $ER^*$ Model

The following theorem characterizes the $ER^*$ machines input sets and computed functions. We will use the same notations as used in Sections 3.2 and 3.4.

**Theorem 45 (Path Decomposition)** Let $M$ be an $ER^*$-Machine.

a) There exist only finite many computation paths for $M$.

b) Each $V_{\gamma}$ is a semi-algebraic subset of the space $R^n$ (where $n$ is $M$’s input dimension). For each input $x$ in $V_{\gamma}$, $\Phi_M(x)$ is either a semi-algebraic set, or the special "No Output" value. (Furthermore, if $\Phi_M(x)$ equals the "No Output" value for some $x$ in $V_{\gamma}$, then for every $x$ in $V_{\gamma}$, $\Phi_M(x)$ equals the "No Output" value since the same computation path is followed).

**Proof:**

a) As before, the finite number of computation paths follows from the restriction to $ER^*$ circuits.

b) We use the definitions and construction of $R_\gamma$ and $T_\gamma$ as shown in the path decomposition theorem for the ER model (section 3.4). As before the inductive construction of $T_\gamma$ and $R_\gamma$ for computation instructions is done by substitution of the register variable by the computation expression in the instruction. Since we added instructions using the $\tau_{RCF}$ functions, the resulting formulas are in $FOL(\tau_{RCF}^*)$.

For the new root-pick-in-interval instruction, $u : x_s \leftarrow z$ with $(x_k < z < x_{k_2} \land \sum_{i=0}^{m} x_i < z = 0)$, the construction would be as follows:

$$R_{\gamma}^m(x) = R_{\gamma}^{m+1}(x) \text{ and } T_{\gamma}^m(x) = T_{\gamma}^{m+1}(x) \land (x_{k_1} < x_s < x_{k_2} \land \sum_{i=0}^{m} x_i = 0)$$

Therefore both $R_\gamma$ and $T_\gamma$ are definable in $FOL(\tau_{RCF}^*)$, hence $V_{\gamma}$ and $\Phi_M(x)$ when restricted to $\gamma$ are definable in $FOL(\tau_{RCF}^*)$. Therefore, according to proposition 42, $V_{\gamma}$ and $\Phi_M(x)$ when restricted to $\gamma$ are definable in $FOL(\tau_{RCF}^*)$, and since $RCF$ admits quantifier elimination, they are definable by quantifier free $FOL$ formulas, and therefore they are both semi-algebraic.

$\square$

5.5 $\mathbb{R}^*$ Problems Vs. $\mathbb{R}^*$ Algorithms

In this subsection, we would finally like to show, that for each $\mathbb{R}^*$ problem there exists an $\mathbb{R}^*$ algorithm which completely solves the problem, and vice versa, i.e., for each $\mathbb{R}^*$ algorithm there exists an $\mathbb{R}^*$ problem which is completely solved by it.
5.5.1 \( \mathbb{R}^* \) Problem Construction Given an \( \mathbb{R}^* \) Algorithm

**Theorem 46** For each \( \mathbb{R}^* \) algorithm \( A \), there exists an \( \mathbb{R}^* \) problem \( P \), which is completely solved by \( A \).

**Proof:** As in the previous sections the construction of the problem is based on the results of the Path Decomposition Theorem.

Given an \( \mathbb{R}^* \) algorithm \( (A, \bar{x}, \tilde{y}) \), we construct an \( \mathbb{R}^* \) problem \( (P, \bar{x}, \tilde{y}) \).

The set \( \{(\bar{x}, \tilde{y})|\tilde{y} \in \Phi_A(\bar{x})\} \) in \( \mathbb{R}^{n+m} \) is the set of \( A \)'s input-output pairs. We need to show that it is FOL definable and therefore a problem which is completely solved by \( A \) (because of the sets equality).

For every computation path \( \gamma \) in \( A \) we construct a formula \( p_\gamma \):

For each computation path \( \gamma \) in \( A \), \( V_\gamma \) is semi-algebraic.

Therefore it is definable by a quantifier-free FOL formula, \( f_{V_\gamma} (\bar{x}) \).

Also, for each path \( \gamma \), the set

\[ \{\tilde{y}|\tilde{y} \in \Phi_A(\bar{x}) \text{ and } \bar{x} \in V_\gamma\} \]

is semi-algebraic, and therefore definable by a quantifier-free FOL formula, \( f_{\Phi_A|V_\gamma} (\bar{x}, \tilde{y}) \).

Next, define:

\[ p_\gamma (\bar{x}, \tilde{y}) = f_{V_\gamma} (\bar{x}) \land f_{\Phi_A|V_\gamma} (\bar{x}, \tilde{y}) \]

Now,

\[ P(\bar{x}, \tilde{y}) = \bigvee_\gamma p_\gamma (\bar{x}, \tilde{y}) \]

Since the number of computation paths \( \gamma \) is finite according to the path decomposition theorem, \( P \) is well defined.

According to the path decomposition theorem, since the sets \( V_\gamma \) are a partition of the machine's input, then the sets \( (V_\gamma, \Phi_A|V_\gamma) \) fully describe the machine's input-output pairs. Therefore, \( P(x \leftarrow \bar{x}, y \leftarrow \tilde{y}) \) is true iff \( (\bar{x}, \tilde{y}) \) is an input-output pair for \( A \). Therefore, \( P \) is completely solved by \( A \). Note that the constructed problem is even an \( \mathbb{R} \) problem, due to the construction shown in the proof for the path decomposition theorem, the \( \mathbb{R}^* \) function symbols needn't be used for constructing the problem definition \( P \).

\[ \square \]

5.5.2 \( \mathbb{R}^* \) Algorithm Construction Given an \( \mathbb{R}^* \) Problem

**Theorem 47** For each \( \mathbb{R}^* \) problem \( P \), there exists an \( \mathbb{R}^* \) algorithm \( A \), such that \( P \) is completely solved by \( A \).

In order to prove this theorem we will use the technique of cylindrical algebraic decomposition as presented in [CJ98].

This method was already introduced in section 2.4.3 for the purpose of quantifier elimination of non-linear formulas. However, the CAD algorithm can also be applied to solve systems of polynomial equalities and inequalities.
Definition 48
Cells over the ordered field of real numbers $\mathbb{R}$ are defined inductively:

(i) Singletons are 0-cells.

(ii) Open intervals are 1-cells.

(iii) If $C$ is an $m$-cell in $\mathbb{R}^n$ and $f : C \to \mathbb{R}$ is a continuous function definable in $\mathbb{R}$, then its graph

$$\{(\xi^m, y) \in C \times \mathbb{R} : y = f(\xi^m)\}$$

is an $m$-cell in $\mathbb{R}^{n+1}$.

(iv) If $C$ is an $m$-cell in $\mathbb{R}^n$ and $f, g : C \to \mathbb{R}$ is a continuous function definable in $\mathbb{R}$, or are the constant functions $\pm \infty$ and $f(\xi^m) < g(\xi^m)$ for every $\xi^m \in C$, then

$$\{(\xi^m, y) \in C \times \mathbb{R} : f(\xi^m) < y < g(\xi^m)\}$$

is an $(m + 1)$-cell in $\mathbb{R}^{n+1}$.

The input to the CAD algorithm is a set of polynomials in $r$ variables. The output of the CAD algorithm is a cylindrical algebraic decomposition of $\mathbb{R}^r$ into finitely many disjoint semi-algebraic connected subsets, $\{C_i\}$, called cells (defined above) such that:

- Each cell $C_i$ is homeomorphic to $\mathbb{R}^{\delta_i}, 0 \leq \delta_i \leq r$.
- Closure of each cell $C_i$, $\bar{C}_i$, is a union of some cells $C_j$'s:

$$\bar{C}_i = \bigcup_j C_j$$

- In each cell $C_i$, each input polynomial is sign invariant.

The CAD algorithm also produces sample points for each cell.

After applying the CAD algorithm to the set of polynomials in the system, the true cells of the CAD (i.e., cells in which the input equalities and/or inequalities hold, tested by using the sample points of each cell) contain all solutions of the system. If each true cell is zero-dimensional then the number of solutions is finite. Otherwise an algebraic-geometric description is obtained for each component of the solution set.

5.5.3 Cylindrical Algebraic Decomposition (CAD) Algorithm Outline

The CAD algorithm has two phases. In the first phase a projection operation is repeatedly applied, each time eliminating one variable, until polynomials in only one variable are produced. At each step, given polynomials in $k$ variables,
a set of polynomials in \(k-1\) variables is computed. The roots of the computed polynomials constitute a “silhouette” of the given polynomials, i.e., the roots of the computed polynomials are the projections onto \(\mathbb{R}^{k-1}\) of the “significant points” of the curve represented by the given polynomials (i.e., self-crossings, isolated points, points at which its tangent is vertical, etc.).

The second phase is called lifting or stack construction. This phase starts with the univariate polynomials produced by the last projection, producing a CAD of one-dimensional space, such that in each of its cells all these polynomials are sign-invariant. The cells of this CAD are the real zeros of these polynomials (these cells will be called sections) and the open intervals between consecutive zeros (including the semi-infinite open intervals preceding and following all zeros), which will be called sectors. These sections and sectors in their natural order are also called a stack.

The CAD of \(\mathbb{R}^1\) is then extended to a CAD of \(\mathbb{R}^2\) by constructing a stack over each cell in the \(\mathbb{R}^1\) CAD, using the set of bivariate polynomials which were produced by projection (in phase 1). By the properties of the projection operation, and by the underlying theory, the real zeros of these bivariate polynomials intersected with the cylinder whose base is any cell of 1-space, are linearly ordered continuous functions defined on the base, called sections. The connected sets between successive sections (also below all sections and above all sections) are called sectors. These sections and sectors together comprise a stack and all such stacks comprise a sign-invariant CAD of \(\mathbb{R}^2\).

Next, this CAD of \(\mathbb{R}^2\) is extended to a CAD of \(\mathbb{R}^3\) using the same technique, i.e., using the set of polynomials in 3 variables produced at the projection phase to construct a stack over each cell in the \(\mathbb{R}^2\) CAD. These liftings or stack constructions continue until we obtain the required CAD for \(\mathbb{R}^n\).

Following are a few examples for applying the CAD algorithm to solving polynomial inequalities systems.

**Example 49**
Consider the inequality: \(x^2 + y^2 - 1 < 0\) (representing all the points \((x, y)\) within the unit circle, as illustrated in figure 1).

The input to the CAD algorithm is the polynomial \(x^2 + y^2 - 1\).

The significant points of this polynomial are \((-1,0), (1,0)\) since at these points the tangent of the curve represented by this polynomial is vertical. The projection of these points onto the x-axis (\(\mathbb{R}^1\)) are the points \(-1, 1, 1, 1\). Therefore the projection phase produces the polynomial: \(x^2 - 1\)

The CAD for \(\mathbb{R}^1\):

\[ C_1 = (-\infty, -1), C_2 = [-1], C_3 = (-1, 1), C_4 = [1], C_5 = (1, \infty) \]

This CAD is now extended to a CAD of \(\mathbb{R}^2\) by constructing a stack over each of these cells.
Figure 1: Figure for Example 49

Constructing a stack over $C_3$ would yield:

$$C_{3,1} = \{-1 < x < 1, -\infty < y < -\sqrt{1-x^2}\}$$
$$C_{3,2} = \{-1 < x < 1, y = -\sqrt{1-x^2}\}$$
$$C_{3,3} = \{-1 < x < 1, -\sqrt{1-x^2} < y < \sqrt{1-x^2}\}$$
$$C_{3,4} = \{-1 < x < 1, y = \sqrt{1-x^2}\}$$
$$C_{3,5} = \{-1 < x < 1, \sqrt{1-x^2} < y < \infty\}$$

Constructing a stack over each of the cells $C_1, C_2, C_4, C_5$ would each yield a single cell:

$$C_{1,1} = \{-\infty < x < -1, -\infty < y < \infty\}$$
$$C_{2,1} = \{x = -1, -\infty < y < \infty\}$$
$$C_{4,1} = \{x = 1, -\infty < y < \infty\}$$
$$C_{5,1} = \{1 < x < \infty, -\infty < y < \infty\}$$

For the given inequality, $C_{3,3}$ is the only cell in this CAD for which the inequality holds (i.e. it is the only true cell of this CAD).

Now we will show an algorithm $A$, for solving the problem $(x^2 + y^2 - 1 < 0, (x, y))$, which relies on the above CAD (note that computation instructions are grouped together into a single instruction, for readability reasons):

1: $r_1 \leftarrow \text{choose } x \text{ with } (-1 < x < 1)$
2: $r_2 \leftarrow +\sqrt{1-r_1^2}$
3: $r_3 \leftarrow 0 - r_2$
4: $r_4 \leftarrow \text{choose } x \text{ with } (r_3 < x < r_2)$
5: $\text{Output } r_1, r_4 \text{ as } x, y$
Example 50
Consider the following (linear) inequalities system as illustrated in figure 2:
\[(x > 0) \land \]
\[(y > 0) \land \]
\[(y + x - 3 < 0)\]

The input to the CAD algorithm are the polynomials \(x, y, y + x - 3\).
The significant points of these polynomials are \((3,0), (0,0), (0,3)\) since these are the intersection points for each 2 of the 3 lines represented by these polynomials. The projection phase defines the points onto the x-axis \(\mathbb{R}^1\) are the points 0, 3.
Therefore the projection phase produces the polynomials: \(x - 3, x\).

The CAD for \(\mathbb{R}^1\):
\[C_1 = (-\infty, -0), C_2 = [0], C_3 = (0, 3), C_4 = [3], C_5 = (3, \infty)\]

This CAD is now extended to a CAD of \(\mathbb{R}^2\) by constructing a stack over each of these cells.
Constructing a stack over \(C_1\) would yield:
\[C_{1,1} = \{-\infty < x < 0, -\infty < y < 0\}\]
\[C_{1,2} = \{-\infty < x < 0, y = 0\}\]
\[C_{1,3} = \{-\infty < x < 0, 0 < y < 3 - x\}\]
\[C_{1,4} = \{-\infty < x < 0, y = 3 - x\}\]
\[C_{1,5} = \{-\infty < x < 0, 3 - x < y < \infty\}\]

Constructing a stack over \(C_2\) would yield:
\[C_{2,1} = \{x = 0, -\infty < y < 0\}\]
\[C_{2,2} = \{x = 0, y = 0\}\]
\[C_{2,3} = \{x = 0, 0 < y < 3\}\]
\[C_{2,4} = \{x = 0, y = 3\}\]
\[C_{2,5} = \{x = 0, 3 < y < \infty\}\]
Constructing a stack over $C_3$ would yield:

$$C_{3,1} = \{0 < x < 3, -\infty < y < 0\}$$
$$C_{3,2} = \{0 < x < 3, y = 0\}$$
$$C_{3,3} = \{0 < x < 3, 0 < y < 3 - x\}$$
$$C_{3,4} = \{0 < x < 3, y = 3 - x\}$$
$$C_{3,5} = \{0 < x < 3, 3 - x < y < \infty\}$$

Constructing a stack over $C_4$ would yield:

$$C_{4,1} = \{x = 3, -\infty < y < 0\}$$
$$C_{4,2} = \{x = 3, y = 0\}$$
$$C_{4,3} = \{x = 3, 0 < y < \infty\}$$

Constructing a stack over $C_5$ would yield:

$$C_{5,1} = \{3 < x < \infty, -\infty < y < 3 - x\}$$
$$C_{5,2} = \{3 < x < \infty, y = 3 - x\}$$
$$C_{5,3} = \{3 < x < \infty, 3 - x < y < 0\}$$
$$C_{5,4} = \{3 < x < \infty, y = 0\}$$
$$C_{5,5} = \{3 < x < \infty, 0 < y < \infty\}$$

For the given inequalities system, $C_{3,3}$ is the only cell in this CAD for which the inequalities hold, (i.e. it is the only true cell of this CAD).

Now we will show an algorithm $A$, for solving the problem $(x > 0 \land y > 0 \land y + x - 3 < 0)$, which relies on the above CAD (note that computation instructions are grouped together into a single instruction, and branch instructions contain a complex condition, for readability reasons):

1: if $x \leq 0 \lor x \geq 3$ Output “No Output” and end
2: $r_1 \leftarrow 3 - x$
3: $r_2 \leftarrow$ choose $x$ with $(0 < x < r_1)$
4: Output $r_2$ as $y$

Example 51
Consider the following equality (see figure 3):

$$y^4 - 2y^3 + y^2 - 3x^2y + 2x^4 = 0$$

The input to the CAD algorithm is the polynomial $y^4 - 2y^3 + y^2 - 3x^2y + 2x^4$. All the significant points are points in which the polynomial has a multiple zero. Therefore we can use the polynomial’s discriminant as the projected polynomial, since a polynomial’s discriminant vanishes iff the polynomial has a multiple zero.

If we consider the polynomial as a polynomial in $y$, then:

$$a_0 = 2x^4, a_1 = -3x^2, a_2 = 1, a_3 = -2, a_4 = 1$$
and the discriminant of a quartic equation
\[ a_4 y^4 + a_3 y^3 + a_2 y^2 + a_1 y + a_0 \]
is
\[ D_4 = [(a_1^2 a_2^2 a_3^2 - 4 a_1^3 a_3^3 - 4 a_1^2 a_2^2 a_4 + 18 a_1 a_2 a_3 a_4 - 27 a_1^4 a_3^2 + 256 a_0^3 a_4) \\
+ a_0(-4 a_2^3 a_3^3 + 18 a_1 a_2 a_3^3 + 16 a_1^2 a_3 a_4 - 80 a_1 a_2^2 a_3 a_4 - 6 a_1^2 a_3^2 a_4 + 144 a_1^2 a_2 a_3^2) \\
+ a_2^2(-27 a_3^4 + 144 a_2 a_3^2 a_4 - 128 a_2^2 a_4^2 - 192 a_1 a_3 a_4^2)] \]

So the projection polynomial is:
\[ 2048x^{12} - 4608x^{10} + 37x^8 + 12x^6 \]

Its zeros are at approximately: \(-1.49, -0.23, 0, 0.23, 1.49\). These points provide the decomposition of \(\mathbb{R}^1\) into cells (containing the points themselves and the open intervals between them and below and above all points). The resulting CAD after the extension phase is illustrated in Figure 4.

The examples help illustrate that we can look at the CADs obtained throughout the CAD algorithm execution as a collection of cell trees.

Formally let us define a cells graph to be a directed graph such that the set of nodes is the set of all cells obtained throughout the CAD algorithm (i.e. the cells comprising the CAD for \(\mathbb{R}^1\), the cells comprising the CAD for \(\mathbb{R}^2\), \ldots, the cells comprising the CAD for \(\mathbb{R}^r\)). There is an edge from node (cell) \(C\) to \(C'\) if \(C'\) belongs to the stack constructed over \(C\) in the second phase of the algorithm.

We can see that the structure of the above graph is a collection of disjoint trees, the root of each is a cell in the \(\mathbb{R}^1\) CAD, and for each cell (node) its direct successors are the cells in the stack constructed over it.

The examples also help understand how the cell trees can be used to construct an algorithm for solving a given problem.
5.5.4 Back to Theorem 47

Now we are ready to prove Theorem 47.

Proof: Given an $\mathbb{R}^*$ problem $P$, we first construct an equivalent quantifier free $\mathbb{R}$ problem $P'$ (as shown in Propositions 42, 44). Then, we will construct an $\mathbb{R}^*$ algorithm which completely solves $P'$ (and therefore completely solves $P$ according to Observation 33 shown in Section 4.1).

$P'$ is a boolean combination of polynomial equalities and inequalities, in $n + m$ variables (where $n$ and $m$ are the dimensions of the solution and data variables of $P'$ respectively), i.e.

$$P' = \bigwedge_i \bigvee_j q_{ij}(\tilde{x}, \tilde{y}) \triangleq 0$$

where $\triangle \in \{=, <, \leq, >, \geq\}$.

We apply the CAD algorithm to the polynomials $q_{ij}$ in $P'$ in order to obtain a decomposition of the $(n + m)$-dimensional real space from which we determine which are the true cells, i.e. in which cells $P'$ holds (by using the sample points which are a part of the CAD algorithm's output). We will not only use the final CAD of $\mathbb{R}^{n+m}$, but also all the intermediate CADs obtained during the stack construction phase of the CAD algorithm, i.e. the whole cells graph which as mentioned above contains a collection of trees of cells (in example 49 we would use the cells $C_1, C_2, \ldots, C_5$ as well as $C_{1,1}, C_{2,1}, \ldots, C_{5,1}$). After marking which cells in the $\mathbb{R}^{n+m}$ CAD are true cells for $P'$, we backtrack the cells trees and mark the cells leading down to true cells, which we will also refer to as true cells.

Figure 5 illustrates the cells trees which are obtained by applying the CAD algorithm.

We will set the order of the variables in the polynomials of $P'$ such that all the solution variables are eliminated first during the projection phase, and

Figure 4: Resulting CAD for Example 51
Figure 5: CAD Algorithm Output - Cells Trees. Red cells denote true cells.

only then the data variables are eliminated, and therefore during the lifting phase the data variables are added first and only then the solution variables (In example 49 there are only solution variables so there is no significance to variables order. In example 50 y would be eliminated first in the projection phase). Hence, the first $m$ levels of the cells trees (starting from the roots) are for the data variables, and the next $n$ levels are for the solution variables.

We now describe the constructed algorithm (given by its outline rather than explicit machine instructions):

- Stroll down the first $m$ data levels of the cells graph, according to the (given) value of $\tilde{x}$ until the cell (in the $m$th level) corresponding to the input assignment, $\tilde{x}$, is reached. If this cell is not one of the true cells of the graph, then output the special “No Output” value and end.

- Consider the cell $C_i, \ldots , C_m$ within which $\tilde{x}$ lies (the cell we reached at the end of the previous phase). Its successors in the tree are the cells in the stack constructed over it during the second phase of the CAD algorithm. From the cells in this stack, select one of the section or sector cells which
are true cells (using the discrete-pick instruction over the set \( \{1, \ldots, k\} \) where \( k \) is the number of sections and sectors over the given cell which are true cells), \( C_{i_1, \ldots, i_m, i_{m+1}} \).

- Obtain a value for \( y_i \) within the selected section/sector in the following way:
  - If \( C_{i_1, \ldots, i_m, i_{m+1}} \) is a section - as mentioned in the CAD algorithm description, it is given by the real zeros of a polynomial \( p_f(\bar{x}, y_i) \) intersected with the base cell. Use the root-pick-in-interval instruction described above to select a value for \( y_i \).
  - If \( C_{i_1, \ldots, i_m, i_{m+1}} \) is a sector - as mentioned before the sector functions \( f \) and \( g \) (\( f < g \) within this sector) are given as \( p_f(\bar{x}, y_i) = 0, p_g(\bar{x}, y_i) = 0 \) intersected with the base cell.

\[
\begin{align*}
  y_i' &\leftarrow \text{root - pick - in - interval}(f(\bar{x}, y_i)) \\
  y_i'' &\leftarrow \text{root - pick - in - interval}(g(\bar{x}, y_i)) \\
  y_i &\leftarrow \text{choose } z \text{ with } \left( y_i' < z < y_i'' \right)
\end{align*}
\]

- Iteratively repeat the last 2 operations for \( i = 2, \ldots, n \), i.e. each time going down one more level in the cells tree, selecting one of the true sections/sectors from the stack constructed over the previous cell (using a discrete selection instruction), and obtain the value of \( y_i \) according to the section's/sector's polynomials in \( \bar{x}, y_1, \ldots, y_i \) (we use the values previously obtained for \( y_1, \ldots, y_{i-1} \) and the input assignment for \( \bar{x} \)).

- Output \( \bar{y} \) and end.

Note that for an input-free problem, the algorithm would start from selecting one of the true root cells, and going down its entire tree, i.e. start from a CAD for \( \mathbb{R}^1 \) and follow the entire lifting phase of the CAD algorithm throughout the CAD for \( \mathbb{R}^n \) in order to obtain the values for \( \bar{y} \). In example 49 the problem is input free, however \( C_3 \) is the only possible choice since it is the only true root, therefore the algorithm starts from choosing a value for \( x \) within \( C_3 \) and then choosing a value for \( y \) accordingly from within \( C_{3.3} \).

We can see that each output is a value within one of the true cells of the CAD for \( P' \), since it was constructed this way. Therefore the each possible output is a solution for \( P' \), hence the algorithm solves the problem \( P' \). We can also see that any value within the CAD true cells can be output, and since the CAD contains the entire solutions space of the problem, the algorithm completely solves \( P' \) (and therefore completely solves \( P \) as argued above).

Although our branch instructions depend on which cell is selected at each step, this dependency can be obtained by a value dependency, by keeping all the required values for each cell and using indirect addressing to access them as a dependency of the value of the chosen cell's index. Therefore the constructed algorithm will not contain branch instructions which depend on the results of pick instructions, and therefore the constructed algorithm is valid in our model.
Time Complexity for the Constructed Algorithm

The algorithm strolls along (using discrete selections among the successors at each step) a single path from one of the roots to one of the leaves of its tree. However, since each discrete selection “hides” looking at all the successors in order to prepare the set from which to choose, of true cells indices, an upper bound would be to say that the algorithm goes through all the cells in the graph, i.e. the algorithm’s time complexity depends on the size of the cells graph.

In [Var] it is shown that the number of cells in a CAD is bounded by \((l \cdot D)^O(n+m)\), where \(l\) is the number of input polynomials and \(D\) is a bound on the degrees of all input polynomials. Therefore the complexity is exponential.

Observation

In example 50, where the input polynomials were linear, the polynomials produced at the projection phase of the CAD algorithm were also linear. Therefore, the constructed algorithm for this example’s problem would be a linear algorithm.

Does this also hold in general? Could the CAD algorithm be used for the linear case as well, in order to obtain linear algorithms for linear problems? In fact, the answer is yes.

Consider a linear problem defined by a formula \(P(\vec{x}, \vec{\gamma})\), which is linear in \(\vec{\gamma}\) (without loss of generality we can assume that \(P\) is already quantifier free as was shown earlier), let \(A\) be the set of polynomials which appear in \(P\). Each \(A_i(\vec{x}, \vec{\gamma})\) is linear in \(\vec{\gamma}\).

We will focus on the first \(n\) steps of the projection phase of the CAD algorithm, where each time \(y_{n \ i+1}\) is eliminated, producing polynomials in the variables \(\vec{x}, y_1, \ldots, y_{n \ i}\), since the last \(m\) steps produce polynomials in which no variables from \(\vec{\gamma}\) appear, therefore they are linear in \(\vec{\gamma}\).

We will show by induction that all polynomials produced in the first \(n\) steps are linear in \(\vec{\gamma}\):

- Base: the input polynomials which are known to be linear in \(\vec{\gamma}\).

- Now consider the step of eliminating the variable \(y_{n \ i+1}\). By the induction assumption all the polynomials at the beginning of this step are linear in \(\vec{\gamma}\). Consider these polynomials as polynomials in the variable \(y_{n \ i+1}\) whose coefficients are polynomials in \(\vec{x}, y_1, \ldots, y_{n \ i}\). All polynomials in the projection result of this step are formed from these coefficients by addition, subtraction, and multiplication. Since the original polynomials are linear in \(\vec{\gamma}\), coefficients of \(y^n_{n \ i+1}\) where \(k \neq 0\) do not contain any elements of \(\vec{\gamma}\), therefore the resulting polynomials are also linear in \(\vec{\gamma}\).
5.6 Choice Operators for the $ER^*$-Model (Revisited)

Once again, taking another look at the algorithm construction presented in the previous subsection, we can see that the construction very much depends on the choice operator which was chosen for the model. For instance, if our model had used the algebraic-pick instruction instead of the interval-pick instruction (with the macros variations we defined over it) then the algorithm construction would be very simple:

\[ y \leftarrow \text{choose } z \text{ with } P'(x, z) \]

Since we have shown that an algorithm can be constructed using the interval-pick (and the additional macro instructions get-relevant-root, discrete-pick) instruction, then this means that the algebraic-pick choice operator can be implemented by means of the interval-pick operator, using the CAD algorithm. Given an algorithm ($ER^*$-Machine) which uses algebraic-pick instructions, we can directly construct an algorithm which uses only interval-pick instruction (and macros), by replacing each algebraic-pick instruction by the subroutine described in the previous proof as the constructed algorithm’s outline. Therefore using algebraic-pick instructions would not add expressive power to our model.
References


