Informatica — Universiteit van Amsterdam

Bachelor Informatica

University of Amsterdam

Thread-oriented program algebra

S.D. Melzer

June 8, 2018

Supervisor(s): dr. I. Bethke and dr. A. Ponse

Signed:

Abstract

This thesis introduces TOP (Thread-Oriented Program algebra) and TOP_2. TOP is an alternative to PGA (ProGram Algebra) and the semigroup C. TOP_2 is a 2-dimensional variant of TOP.

In this thesis several properties of TOP and TOP_2 are studied. All TOP instruction sequences model regular threads and all regular threads can be modelled by TOP instruction sequences. The behaviour expressed by a TOP instruction sequence of length n can be defined by a linear specification of $n + 1$ equations. The behaviour defined by a linear specification of m equations can be expressed by a instruction sequence of length $3m - \lceil \frac{m}{2} \rceil$.

Analogous to instruction sequences, TOP_2 uses instruction planes to model sequential programs. TOP_2 is equally expressive as TOP but does not require arbitrarily large jumps to express all regular threads. TOP on the other hand does require arbitrarily large jumps in both directions.

Contents

CHAPTER 1

Introduction

Bergstra and Loots presented ProGram Algebra (PGA) in [2]. PGA is an algebra of programs which is supposed to capture the core of imperative sequential programming languages. The syntax of PGA serves as a very simple program notation. The primitives of this notation are designed to enable single pass execution of instruction sequences. This design choice resulted in a directional bias, instruction sequences are always executed in a left-to-right manner. While this bias is clearly existing in most practical imperative programming languages, the assumption that this is a core property of imperative programming languages should not be made cautiously. It is likely that this is merely a result of the natural language of the designers of programming languages.

Bergstra and Ponse introduced C as an alternative to PGA in [3]. C is a semigroup of finite instruction sequences in which programs can be represented without directional bias. C has both forward and backward instructions and a C-expression can be interpreted starting from any instruction. Properties of C are studied by Bergstra and Ponse in [3], and Schroevers in [10]. It is proven that C requires arbitrarily large jumps in both directions to model all regular threads.

The semantics of C and PGA are defined using Basic Thread Algebra (BTA). BTA is a set of axioms used to describe the behaviour of sequential programs. BTA expressions, called threads, can describe finite and infinite behaviour. PGA models infinite threads with infinite instruction sequences. In contrast C models infinite threads with finite instruction sequences. In practice any program must be finite, thus C appears to be a more practical approach than PGA.

Schroevers introduced a variant of C in [10]. This variant is denoted TOP and is the main topic of this thesis. The semantics and syntax of TOP are almost identical to C. Only the test instructions are defined slightly different. Concretely, the test instructions of TOP are chosen to have a direct correlation with BTA, while the test instructions of C are more or less based on PGA. This correlation might be advantageous to TOP since programs can therefore closely match the BTA thread they model.

In this thesis some properties of TOP are studied. First, the semantics of BTA and TOP are described in more detail in Chapter 2. Second, in Chapter 3 the expressiveness of TOP is explored. Third, Chapter 4 discusses the length of TOP instruction sequences.

Finally, in Chapter 5 the dimensional bias of TOP is investigated by introducing a two-dimensional interpretation of TOP. Dimensional bias is the fact that the control can move in only one dimension. Similarly to directional bias, dimensional bias might be a side effect of the natural language used by the designers of programming languages. On the other hand, the dimensionality might be related to the fact that these types of languages are based on a Boolean system. Every test has only two responses, which correlates with going either forwards or backwards in TOP.

CHAPTER 2

Preliminaries

2.1 Basic Thread Algebra

Most of the text in this section is based on the work of Ponse and van der Zwaag in [8].

Basic Thread Algebra (BTA) is a form of process algebra used to describe the behaviour of sequential programs. BTA is based on an arbitrary set of basic actions $\mathcal{A}$. On execution each action yields a Boolean value `true` or `false`. The set of all BTA expressions is denoted BTA.

BTA expressions are called threads, which are built with two constants and a single ternary operator.

- The *termination* constant is denoted by $\mathsf{S} \in \text{BTA}$, and describes termination of a program.
- The *deadlock* or *inaction* constant is denoted by $\mathsf{D} \in \text{BTA}$. This behaviour indicates that the program is in a state where no more actions can be executed.
- The *postconditional composition* operator $P \trianglelefteq a \trianglerighteq Q : \text{BTA} \times \mathcal{A} \times \text{BTA} \to \text{BTA}$. This operator describes the behaviour that executes some action a. If a generates `true` the execution continues with P and otherwise with Q.

Additionally there are some convenience notations. These serve to simplify equations but do not add any functionality.

1. First, there is the *action prefix operator* $a \circ P : \mathcal{A} \times \text{BTA} \to \text{BTA}$. It denotes the behaviour $P \trianglelefteq a \trianglerighteq P$, i.e., the result of a has no influence on the control flow. The action prefix operator binds stronger than the postconditional composition operator.
2. Second, there is the abbreviation a^n. It describes that action a is repeated n times, regardless of the Boolean responses. It is defined recursively: $a^1 = a$ and $a^{n+1} = a \circ a^n$.

Every thread in BTA has an upper bound to the number of actions it can perform. Thus each closed thread models finite behaviour which either ends with deadlock or termination. The *approximation* operator $\pi(n, P) : \mathbb{N} \times \text{BTA} \to \text{BTA}$ binds the number of actions performed by P to n. The operator is defined as follows:

$$\begin{aligned} \pi(0, P) &= \mathsf{D}, \\ \pi(n+1, \mathsf{S}) &= \mathsf{S}, \\ \pi(n+1, \mathsf{D}) &= \mathsf{D}, \\ \pi(n+1, P \trianglelefteq a \trianglerighteq Q) &= \pi(n, P) \trianglelefteq a \trianglerighteq \pi(n, Q), \end{aligned}$$

for $P, Q \in \text{BTA}$ and $n \in \mathbb{N}$. Since all threads are finite each thread P has some upper bound $n \in \mathbb{N}$ such that for all $m \geq n$:

$$\pi(m, P) = P.$$

BTA^∞ is the complete partial order consisting of all projective sequences:

$$\mathrm{BTA}^\infty = \{(P_n)_{n\in\mathbb{N}} \mid \forall n \in \mathbb{N}(P_n \in \mathrm{BTA} \wedge \pi(n, P_{n+1}) = P_n)\}.$$

Hence BTA^∞ also contains the infinite threads. In BTA^∞ equality is defined componentwise: $(P_n)_{n\in\mathbb{N}} = (Q_n)_{n\in\mathbb{N}}$ if $P_n = Q_n$ for all $n \in \mathbb{N}$.

The constants and operators of BTA are defined for BTA^∞ by:

$$\begin{aligned}
\mathsf{D} &= (\mathsf{D}, \mathsf{D}, \dots),\\
\mathsf{S} &= (\mathsf{D}, \mathsf{S}, \mathsf{S}, \dots),\\
(P_n)_{n\in\mathbb{N}} \trianglelefteq a \trianglerighteq (Q_n)_{n\in\mathbb{N}} &= (R_n)_{n\in\mathbb{N}} \text{ with } R_0 = \mathsf{D} \text{ and } R_{n+1} = P_n \trianglelefteq a \trianglerighteq Q_n,\\
\pi(n, (P_m)_{m\in\mathbb{N}}) &= (P_0, \dots, P_{n-1}, P_n, P_n, P_n, \dots).
\end{aligned}$$

The set of *residual threads* of P is defined as $\mathsf{Res}(P)$:

$$\begin{aligned}
&P \in \mathsf{Res}(P),\\
&Q \trianglelefteq a \trianglerighteq R \in \mathsf{Res}(P) \implies Q \in \mathsf{Res}(P) \wedge R \in \mathsf{Res}(P).
\end{aligned}$$

A thread P is *regular* if and only if $\mathsf{Res}(P)$ is finite. Furthermore, a thread Q is a 0-*residual* of thread P if $P = Q$, and an $n+1$-*residual* of P if for some $a \in \mathcal{A}, P = P_1 \trianglelefteq a \trianglerighteq P_2$ and Q is an n-*residual* of P_1 or P_2. A *finite linear recursive specification* over BTA^∞ is a set of equations

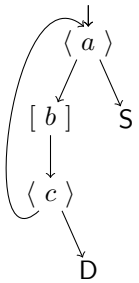
$$x_i = t_i$$

for $i \in I$ with I some finite index set, variables x_i, and all t_i terms of the form S, D, or $x_j \trianglelefteq a \trianglerighteq x_k$ with $j, k \in I$. Observe that P is regular if and only if P is the solution of a finite linear recursive specification.

Threads can be represented graphically by directed graphs. In these illustrations angular brackets and square brackets represent the postconditional composition operator and the action prefix operator, respectively. The left descent of a postconditional composition node denotes a `true` yield from the execution and the right descent the opposite. Finally, the initial state is indicated by an arc without a source.

Example 2.1.1. The specification and graphically representation of some thread P_1.

$$\begin{aligned}
P_1 &= P_2 \trianglelefteq a \trianglerighteq P_3\\
P_2 &= b \circ P_4\\
P_3 &= \mathsf{S}\\
P_4 &= P_1 \trianglelefteq c \trianglerighteq P_5\\
P_5 &= \mathsf{D}
\end{aligned}$$



2.2 TOP instruction sequences

This section describes the syntax and semantics of Thread-Oriented Program algebra (TOP). It is briefly introduced in [10] as a variant of C. Like C, TOP does not have a directional bias; all directional instructions exist in a forward and backward flavour and execution can start at an arbitrary position in an instruction sequence.

TOP is based on a set of actions $\mathcal{A}$. This set is often kept implicit. For most proofs in this thesis it assumed that $|\mathcal{A}| > 1$. In equations, elements of $\mathcal{A}$ are written as lowercase letters $\{a, b, \dots\}$.

The primitives or instructions of TOP can be separated into two types.

1. *Directed* instructions continue execution in some direction. Instructions with a forward slash ($/$) are called forward instructions and instructions with a backward slash ($\backslash$) backward instructions. As the names imply, forward instructions move control forward and backward instructions move control backward. In this thesis sequences are written from left to right, thus forward corresponds with a left-to-right direction and backward with a right-to-left direction.
   - *Basic* instructions $/a$ and $\backslash a$ for $a \in \mathcal{A}$ execute action a and move control exactly one instruction forward or backward, respectively.
   - *Jump* instructions $/\#k$ and $\backslash\#k$ for $k \in \mathbb{N}^+$ move control k instructions forward or backward, respectively. A jump instruction $/\#k$ or $\backslash\#k$ has a *jump counter* k and jumps over $k-1$ instructions.
   - *Test* instructions $+a$ and $-a$ for $a \in \mathcal{A}$ execute action a and depending on the yield of a move control either forward or backward. The positive test instruction $+a$ moves control backward if a yields `true` and forward if a yields `false`. Execution of the negative test instruction $-a$ is the same, except that the directions are mirrored.
2. *Undirected* instructions indicate that no progress will be made after execution. These instructions do not move control in any way. After these instructions are executed, no other instruction will ever be executed. These instructions are also unique in the way that they do not have a mirrored counterpart. Hence they are x undirected.
   - The *termination* instruction ! terminates the execution of the program.
   - The *abort* instruction # indicates that the program can make no progress and models inaction or deadlock.

These primitives form the set of instructions $\mathcal{I}$. Formally,

$$\mathcal{I} = \bigcup_{a \in \mathcal{A}} \{/a, \backslash a, +a, -a\} \cup \bigcup_{k \in \mathbb{N}^+} \{/\#k, \backslash\#k\} \cup \{\#, !\}.$$

Instructions of the set $\mathcal{I}$ are concatenated to create *instruction sequences*. The concatenation operator $\_;\_ : \mathcal{I}^n \times \mathcal{I}^m \to \mathcal{I}^{n+m}$ is defined as follows:

$$\begin{aligned} \mathcal{I}^1 &= \mathcal{I}, \\ \mathcal{I}^{n+1} &= \{X; u \mid X \in \mathcal{I}^n, u \in \mathcal{I}^1\}. \end{aligned}$$

Here concatenation is associative $(X;Y);Z = X;(Y;Z)$ for any instruction sequences X, Y, Z. TOP is the union of all sets $\mathcal{I}^n$ for $n \in \mathbb{N}^+$:

$$\text{TOP} = \bigcup_{n \in \mathbb{N}^+} \mathcal{I}^n.$$

Therefore, TOP contains all finite, non-empty sequences of instructions in $\mathcal{I}$. X is called a TOP instruction sequence if and only if $X \in \text{TOP}$. The length of an instruction sequence X corresponds to the number of instructions in the sequence, i.e., the length of $X \in \mathcal{I}^n$ is n.

The semantics of TOP are defined using BTA. For some TOP instruction sequence X each position $i \in \mathbb{Z}$ is assigned a thread in BTA^∞ using the *thread extraction operator* $|\_|\_ : \mathrm{TOP} \times \mathbb{Z} \to \mathrm{BTA}^\infty$

$$|X|_i = \begin{cases} a \circ |X|_{i+1} & \text{if } \sigma(X,i) = /a, \\ a \circ |X|_{i-1} & \text{if } \sigma(X,i) = \backslash a, \\ |X|_{i+k} & \text{if } \sigma(X,i) = /\#k, \\ |X|_{i-k} & \text{if } \sigma(X,i) = \backslash\#k, \\ |X|_{i-1} \trianglelefteq a \trianglerighteq |X|_{i+1} & \text{if } \sigma(X,i) = +a, \\ |X|_{i+1} \trianglelefteq a \trianglerighteq |X|_{i-1} & \text{if } \sigma(X,i) = -a, \\ \mathsf{D} & \text{if } \sigma(X,i) = \#, \\ \mathsf{S} & \text{if } \sigma(X,i) = !, \\ \\ \mathsf{D} & \text{if } \sigma(X,i) = \bigcirc, \end{cases} \tag{2.1}$$

where $a \in \mathcal{A}$, $k \in \mathbb{N}$, and $\sigma(X,i) : \mathrm{TOP} \times \mathbb{Z} \to \mathcal{I} \cup \{\bigcirc\}$ is given by

$$\sigma(X,i) = \begin{cases} u_i & \text{if } 0 < i \leq n, \\ \bigcirc & \text{otherwise,} \end{cases} \tag{2.2}$$

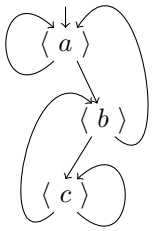
for $X = u_1, \ldots, u_n$.

A TOP instruction sequence $X = u_1; \ldots; u_n$ is said to model a thread $P \in \mathrm{BTA}^\infty$ if there is a starting position $i \in \{1, \ldots, n\}$ for which $|X|_i = P$. Likewise, an instruction sequence X is said to express the behaviour defined by a linear specification $\{P_1{=}t_1, \ldots, P_m{=}t_m\}$ if there is some i for which $|X|_i = P_1$.

Observe that the thread extraction operator allows thread extraction from any position $i \in \mathbb{Z}$ for an instruction sequence, including positions outside of the sequence. Contrarily, an instruction sequence only models a thread if behaviour extraction is started from inside of the sequence $1 \leq i \leq n$ for $X = u_1; \ldots; u_n$. By this fact the deadlock behaviour (D) can be extracted from each $X \in \mathrm{TOP}$ but not all X's can model a thread defined by only the deadlock behaviour.

Example 2.2.1. Let $X = /\#1; +a; -b; +c; \backslash\#1$ then $|X|_1 = P_1$ where P_1 is defined as follows:

$$\begin{aligned} P_1 &= P_1 \trianglelefteq a \trianglerighteq P_2 \\ P_2 &= P_3 \trianglelefteq b \trianglerighteq P_1 \\ P_3 &= P_2 \trianglelefteq c \trianglerighteq P_3 \end{aligned}$$



X can not model deadlock since there is no abort instruction and no movement of control outside of the sequence. However, deadlock can still be extracted from X by extracting from outside of the sequence: $|X|_n = \mathsf{D}$ for $n \in \mathbb{Z}$ and $n \notin \{1, 2, 3, 4, 5\}$.

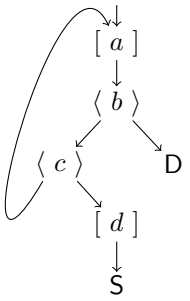
In the case that application of these actions results in a loop without actions the extracted thread is defined as D, e.g., $|/\#1; \backslash\#1|_1 = \mathsf{D}$. Such an occurrence is called a loop without activity.

> If Equation 2.1 can be applied infinitely many times from left to right without ever yielding an action, then the extracted thread is D. (2.3)

Some position j in an instruction sequence $X = u_1; \ldots; u_n$ is *directly reachable* from position i if and only if applying Equation 2.1 exactly once to $|X|_i$ has an occurrence of $|X|_j$ on the right hand side of the equation. A position j is *reachable* from i if and only if applying Equation 2.1 arbitrarily many times to $|X|_i$ has an occurrence of $|X|_j$ on the right hand side of the equation.

Example 2.2.2. The behaviour $P_1 = |X|_5$ for $X = \#; -b; /\#3; \backslash\#2; \backslash a; +c; /d; !; \backslash e$ is defined as follows:

$$
\begin{aligned}
P_1 &= a \circ P_2 \\
P_2 &= P_3 \trianglelefteq b \trianglerighteq P_4 \\
P_3 &= P_1 \trianglelefteq c \trianglerighteq P_5 \\
P_4 &= \mathsf{D} \\
P_5 &= d \circ P_6 \\
P_6 &= \mathsf{S}
\end{aligned}
$$



In X the instruction at position 4 ($\backslash\#2$) is directly reachable from the instruction at position 5 ($\backslash a$), the other way around position 5 is reachable from position 4 but not directly reachable. Finally, position 9 ($\backslash e$) is not reachable from any other position.

CHAPTER 3

On the expressiveness of TOP

In this chapter the expressiveness of TOP is discussed. All regular threads can be modelled by C instruction sequences and all C instruction sequences model regular threads [3]. Since TOP is closely related to C, it would be surprising if the same would not hold for TOP. In the first section of this chapter it is proven that this also holds for TOP. In [10] it is already shown that this must be the case by presenting behaviour preserving homomorphisms from C to TOP and vice versa, nevertheless a formal proof is included in this thesis for completeness.

In the second section of this chapter reduced instruction sets of TOP are studied. Analogous to the findings of Schroevers [10] for C, it is found that jump counters need to be arbitrarily large in both directions for TOP instruction sequences to model all regular threads.

3.1 Regular threads

In this section it is proven that the threads extracted from a TOP instruction sequence are regular and that each regular thread can be modelled by a TOP instruction sequence.

Theorem 3.1.1. *If X is a* TOP *instruction sequence and $i \in \mathbb{Z}$ then $|X|_i$ defines a regular thread.*

Proof. Let $X = u_1; \ldots; u_n$ be a TOP instruction sequence. With Equation 2.1 a linear specification can be created. All threads extracted from outside of the sequence ($\sigma(X, i) = \bigcirc$) give raise to a single equation $|X|_i = \mathsf{D}$, and thus are regular threads. For each $i \in \{1, \ldots, n\}$ an equation of the form

$$|X|_i = |X|_j \trianglelefteq a \trianglerighteq |X|_k \text{ or } |X|_i = |X|_j \text{ or } |X|_i = \mathsf{D} \text{ or } |X|_i = \mathsf{S}$$

is found. Therefore, a finite set of linear equation can be created

$$\{P_i = |X|_i \mid i \in \{1, \ldots, n\}\} \cup \{P_\mathsf{D} = \mathsf{D}\}.$$

Any thread extraction on the right side of equations that extracts from outside the sequence is replaced with D. The remaining extractions $|X|_j$ and $|X|_k$ on the right side of the equations are replaced with P_j and P_k respectively. This results in a linear specification. □

Conversely, TOP instruction sequences can model the behaviour of all regular threads. Given a linear specification $\{P_i{=}t_i \mid i \in \{1, \ldots, n\}\}$ a TOP instruction sequence $X = X_1; \cdots; X_n$ of $3n$ instructions can be constructed such that $|X|_{3i-1} = P_i$.

$$X_i = \begin{cases} \#;!;\#; & \text{if } P_i = \mathsf{S}, \\ \#;\#;\#; & \text{if } P_i = \mathsf{D}, \\ \mathcal{J}(3(j-i)+1);+a;\mathcal{J}(3(k-i)-1); & \text{if } P_i = P_j \trianglelefteq a \trianglerighteq P_k, \end{cases} \tag{3.1}$$

where $\mathcal{J}(i)$ is a relative jump in either the forward or backward direction

$$\mathcal{J}(i) = \begin{cases} /\#i & \text{if } i > 0, \\ \backslash\# - i & \text{if } i < 0. \end{cases} \tag{3.2}$$

Example 3.1.1. If Equation 3.1 is applied to the linear specification

$$\{P_1{=}a \circ P_2, \quad P_2{=}P_1 \trianglelefteq b \trianglerighteq P_3, \quad P_3{=}\mathsf{S}\}$$

an instruction sequence $X \in \mathrm{TOP}$ is obtained,

$$X = /\#4; +a; /\#2; \backslash\#2; +b; /\#2; \#; !; \#.$$

While this is subjectively not the most natural instruction sequence, e.g., the action prefix operator is modelled as a test instruction instead of a basic instruction, it expresses the behaviour defined by all equations in the specification for some position in the sequence.

Theorem 3.1.2. *Each regular thread is modelled by some* TOP *instruction sequence.*

Proof. Let P be a regular thread. There must be a finite linear specification that defines this thread and an instruction sequence that models this thread can be found with Equation 3.1-3.2. □

3.2 Reduced instruction sets

In the previous section it was indirectly shown that not all instructions of TOP are required to model all regular threads. For instance, backward basic instructions or negative test instructions are not present in Equation 3.1 which clearly can create an instruction sequence for any regular thread. This observation leads to the question what instructions in TOP are truly required to model all regular threads. As was stated in Section 2.2, the thread extraction for TOP is defined so that the abort instruction is not needed to express the deadlock behaviour. Deadlock can also be expressed by moving control out of the sequence or by a loop without activity. This leads to an alternative equation which can creates instruction sequences that model all regular threads without the abort instruction.

Theorem 3.2.1. *Let* TOP^- *be defined by allowing only instructions from the set*

$$\{+a; /\#k; \backslash\#k; ! \mid a \in \mathcal{A}, k \in \mathbb{N}^+\}.$$

Each regular thread can be modelled by a TOP^- *instruction sequence.*

Proof. Let P be a regular thread defined by the linear specification $\{P_i{=}t_i \mid i \in \{1, \ldots, n\}\}$. Equation 3.1 is adjusted by modelling deadlock with a loop without activity to construct a TOP^- instruction sequence $X = X_1; \cdots; X_n$ of $3n$ instructions such that $|X|_{3i-1} = P_i$

$$X_i = \begin{cases} !;!;! & \text{if } P_i = \mathsf{S}, \\ /\#1; /\#1; \backslash\#1 & \text{if } P_i = \mathsf{D}, \\ \mathcal{J}(3(j-i)+1); +a; \mathcal{J}(3(k-i)-1) & \text{if } P_i = P_j \trianglelefteq a \trianglerighteq P_k, \end{cases}$$

where $\mathcal{J}(i)$ is a relative jump in either the forward or backward direction

$$\mathcal{J}(i) = \begin{cases} /\#i & \text{if } i > 0, \\ \backslash\# - i & \text{if } i < 0. \end{cases}$$

□

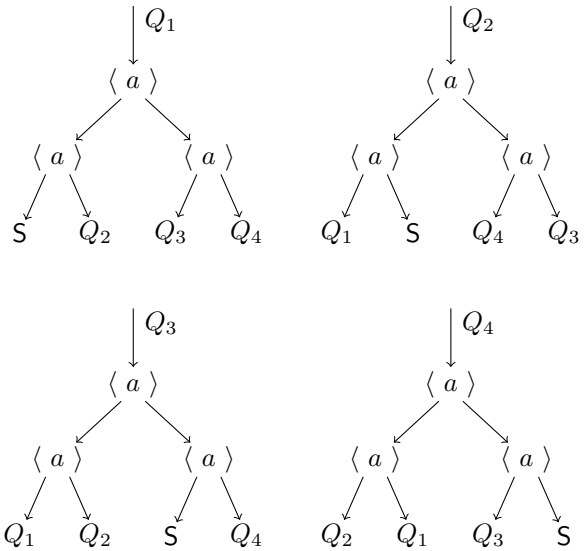
All remaining instructions in TOP^- seem necessary to model all regular threads. For example, it can not be done without the positive test instruction ($+a$ for $a \in \mathcal{A}$) or the termination instruction (!) since that would result in the post conditional composition or termination not being expressible. Furthermore, arbitrarily large jumps in both directions are required to model all regular threads. Consider the following definition:

Let $a \in \mathcal{A}$ and $n \in \mathbb{N}^+$, thread $P = Q_1$ has the $(a,n)^\omega$-property if $\pi_n(P) = a^n \circ \mathsf{D}$ and P has 2^n distinct n-residuals $\{\mathsf{S}, Q_2, \ldots, Q_{2^n}\}$, where each residual Q_i also has 2^n distinct n-residuals $\{Q_j \mid j \in \{1,\ldots,2^n\} \wedge i \neq j\} \cup \{\mathsf{S}\}$. Lastly, no $(n-1)$-residuals of distinct Q_i's may be identical.

Threads with this property can only be modelled with arbitrarily large jumps in both directions. Therefore, such jumps are required to express all regular threads.

Example 3.2.1. Q_1 has the $(a,2)^\omega$-property (and so do Q_2, Q_3 and Q_4).

$Q_1 = Q_{\mathsf{S},2} \trianglelefteq a \trianglerighteq Q_{3,4}$
$Q_2 = Q_{1,\mathsf{S}} \trianglelefteq a \trianglerighteq Q_{4,3}$
$Q_3 = Q_{1,2} \trianglelefteq a \trianglerighteq Q_{\mathsf{S},4}$
$Q_4 = Q_{2,1} \trianglelefteq a \trianglerighteq Q_{3,\mathsf{S}}$
$Q_{1,\mathsf{S}} = Q_1 \trianglelefteq a \trianglerighteq \mathsf{S}$
$Q_{\mathsf{S},2} = \mathsf{S} \trianglelefteq a \trianglerighteq Q_2$
$Q_{\mathsf{S},4} = \mathsf{S} \trianglelefteq a \trianglerighteq Q_4$
$Q_{3,\mathsf{S}} = Q_3 \trianglelefteq a \trianglerighteq \mathsf{S}$
$Q_{1,2} = Q_1 \trianglelefteq a \trianglerighteq Q_2$
$Q_{2,1} = Q_1 \trianglelefteq a \trianglerighteq Q_2$
$Q_{3,4} = Q_3 \trianglelefteq a \trianglerighteq Q_4$
$Q_{4,3} = Q_4 \trianglelefteq a \trianglerighteq Q_3$



Observe the relation of the threads Q_1 to Q_4. From each Q_i each other Q_j is reachable. Yet there is no possibility for the $(n-1)$-residuals for some Q_i to overlap with the $(n-1)$-residuals of some other Q_j because of the $(a,n)^\omega$-property.

Theorem 3.2.2. *Let* $\text{TOP}^{\leq k}$ *be the subset of* TOP *that includes all instructions except forward jump instructions with a jump counter greater than* $k \in \mathbb{N}^+$.

$\text{TOP}^{\leq k}$ *instruction sequences cannot model all regular threads for any* k.

Proof. Let P be a regular thread with the $(a,n)^\omega$-property where $2^n > 2k+3$. P contains the residual threads $Q_1, \ldots, Q_{2^n}$ (note that $Q_1 = P$ and that each $Q \in \{Q_1, \ldots, Q_{2^n}\}$ must have the $(a,n)^\omega$-property). Assume $X = u_1; \ldots; u_m$ is an instruction sequence that models P without forward jump counters greater than k. There must be an index set I where for each $i \in I$ there must be some $j \in \{1,\ldots,2^n\}$ so it is true that $|X|_i = Q_j$. Assume that I is chosen so that $\sum_{i \in I} i$ is minimal and all behaviours are represented by some index in I, e.g., exactly one and only the lowest index for which each Q_j can be extracted from X is in I.

Consider I as an ordered sequence of integers $i_1 < i_2 < \cdots < i_{n-1} < i_{2^n}$. The instruction at position i_1 in X is the first instruction in the sequence that allows extraction of some Q_j. All other indices in I extract some other Q_j. Note that all other Q_j's must be n-residuals of Q_j. It is assumed without loss of generality that for $i \in \{i_2, \ldots, i_{2^n}\}$, i must be reachable from i_1. If this is not the case some indices in I can be replaced by larger indices to make this true.

Observe that there are at least $|q - p| - 1$ instructions between position i_p and position i_q for $p \neq q$. Furthermore, i_1 must be able to reach all instructions $i_2, \ldots, i_{2^n}$ with forward jumps since it is the minimum position in X that extracts some Q_j.

Since there are at least $k + 1$ instructions between i_1 and i_j for $j > k + 3$ and no forward jumps greater than k, i_j can only be reached with chained forward jumps from i_1. Moreover, such a chain is needed for *each* $j > k + 2$ and there are exactly k such j's. Now some forward jump in the chain corresponding to the path from i_1 to i_n must be able to jump over at least one jump instruction of each other chain. This is a contradiction since this jump would need to jump over at least k instructions. □

A similar argument can be made for limiting jump counters of backward jump instructions. This proves that arbitrarily large jumps in both directions are needed to model all regular threads with TOP instruction sequences.

CHAPTER 4

On the length of TOP instruction sequences

This chapter explores the relation of the length of TOP instruction sequences and the number of equations in corresponding linear specifications. In the previous chapter it was shown that for any $X \in \mathrm{TOP}$ a regular thread that is modelled by X can be found and vice versa.

First, it is explored how many equations are needed in a linear specification to define the behaviour modelled by a TOP instruction sequence of length n. An observation can be made in relation to the proof of Theorem 3.1.1 where a linear specification is created given a TOP instruction sequence. For each instruction at most a single equation is added to the specification, and in addition an extra equation defining deadlock is added. Evidently the minimum number of equations should be $n + 1$. This would match the result of Redder for PGA in [9].

Second, the opposite is considered. That is to say, the length of a TOP instruction sequence that expresses the behaviour defined by a linear specifications of n equations. In [3] bounds on the length of C instruction sequences are formulated. An upper bound for C is found to be $3n$. In other words, instruction sequences of minimally length $3n$ are required to model all regular threads of n states. Bouber lowered this upper bound to $3n - \lfloor (n+2)/3 \rfloor$ in [4].

4.1 Instruction sequence to thread

Considering the number of equations needed in a linear specification to define the behaviour expressed by a TOP instruction sequence of length n it is found that generally each instruction adds at most a single equation. The only exception occurs in the case that some non-jump instruction causes implicit deadlock by moving control outside of the instruction sequence. For instance, take the instruction sequence

$$X = +a; /b.$$

This sequence requires three equations to define the extracted behaviour $|X|_1 = P_1$.

$$\begin{aligned} P_1 &= P_2 \trianglelefteq a \trianglerighteq P_3 \\ P_2 &= \mathsf{D} \\ P_3 &= b \circ P_2 \end{aligned}$$

While both instructions can move control outside of the sequence the extra equation is required at most a single time since all occurrences of deadlock can be defined by a single equation.

Theorem 4.1.1. *The behaviour expressed by a* TOP *instruction sequence of length n can be defined by a linear specification of $n+1$ equations.*

Proof. The shortest possible instruction sequence $X \in \mathcal{I}^1$ contains only one instruction. Such an instruction sequence can give rise to three different linear specifications:

1. $\{P_1{=}P_{\mathsf{D}} \trianglelefteq a \trianglerighteq P_{\mathsf{D}}, \quad P_{\mathsf{D}}{=}\mathsf{D}\}$ defines the behaviour expressed by a basic or test instruction.
2. $\{P_1{=}\mathsf{D}\}$ defines the behaviour expressed by a jump or abort instruction.
3. $\{P_1{=}\mathsf{S}\}$ defines the behaviour expressed by a termination instruction.

In each case the set either contains a deadlock state or just a single state. It is assumed without loss of generality that each specification contains the deadlock state and at most two states.

Each longer instruction sequence can be seen as $u_{n+1}; X$ with $X = u_n; \ldots; u_1$ with the associated linear specification $P = \{P_{\mathsf{D}}{=}\mathsf{D}, P_1{=}t_1, \ldots, P_n{=}t_n\}$. The prepended instruction u_{n+1} requires at most one extra equation:

$$\{P_{n+1}{=}\phi(u_{n+1})\} \cup \{\psi(P_i{=}t_i) \mid P_i{=}t_i \in E\},$$

where $\phi(\_) : \mathcal{I} \to \mathrm{BTA}^{\infty}$

$$\phi(u_{n+1}) = \begin{cases} a \circ P_n & \text{if } u_{n+1} = /a, \\ a \circ P_{\mathsf{D}} & \text{if } u_{n+1} = \backslash a, \\ P_{n-k+1} & \text{if } u_{n+1} = /\#k \text{ and } n > k, \\ P_{\mathsf{D}} & \text{if } u_{n+1} = /\#k \text{ and } n \leq k, \\ P_{\mathsf{D}} & \text{if } u_{n+1} = \backslash\#k, \\ P_{\mathsf{D}} \trianglelefteq a \trianglerighteq P_n & \text{if } u_{n+1} = +a, \\ P_n \trianglelefteq a \trianglerighteq P_{\mathsf{D}} & \text{if } u_{n+1} = -a, \\ P_{\mathsf{D}} & \text{if } u_{n+1} = \#, \\ \mathsf{S} & \text{if } u_{n+1} = !, \end{cases}$$

and ψ is a transformation defined as

$$\psi(P_i{=}t_i) = \begin{cases} P_i = P_{n+1} & \text{if } u_i = \backslash\#n+1-i, \\ P_i = a \circ P_{n+1} & \text{if } i = n \text{ and } u_i = \backslash a, \\ P_i = P_{n+1} \trianglelefteq a \trianglerighteq P_{n-1} & \text{if } i = n \text{ and } u_i = +a, \\ P_i = P_{n-1} \trianglelefteq a \trianglerighteq P_{n+1} & \text{if } i = n \text{ and } u_i = -a, \\ P_i = t_i & \text{otherwise.} \end{cases}$$

As such the behaviour expressed by an instruction sequence of length n requires a linear specification of at most $n+1$ equations to be defined. □

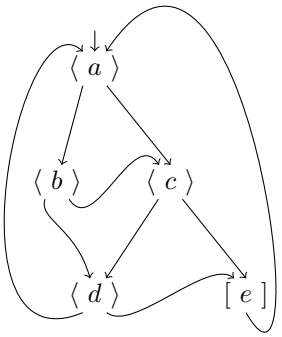
4.2 Thread to instruction sequence

Given a linear specification $P = \{P_i{=}t_i \mid i \in \{1, \ldots, n\}\}$ an instruction sequence $X = u_1; \ldots; u_{3n}$ that expresses the behaviour defined by P_i can be created with Equation 3.1. Since this construct models each state with 3 instructions, it is clear that an instruction sequence of at most $3n$ instructions is needed to express the behaviour defined by a linear specification of n equations.

If an instruction is directly reachable from 2 jump instructions the sequence can be reordered to use a single jump instruction. In other words, when two jumps target the same instruction it is possible to combine them to a single jump instruction. On a thread level this only occurs when two equations in a linear specification have a shared term in the right hand side of the equation.

Example 4.2.1. Consider the following linear specification:

$$
\begin{aligned}
P_1 &= P_2 \trianglelefteq a \trianglerighteq P_3\\
P_2 &= P_4 \trianglelefteq b \trianglerighteq P_3\\
P_3 &= P_4 \trianglelefteq c \trianglerighteq P_5\\
P_4 &= P_1 \trianglelefteq d \trianglerighteq P_5\\
P_5 &= e \circ P_1
\end{aligned}
$$



Applying the construct an instruction sequence of length 15 is found:

$$X = /\#4; +a; /\#5; /\#7; +b; /\#2; /\#4; +c; /\#5; \backslash\#8; +d; /\#2; \backslash\#11; +e; \backslash\#13.$$

Combining the jumps with shared targets a sequence of 11 instructions is obtained. Observe that even when the two jumps are generated by a single equation the combination is possible by using a basic instruction.

$$X = /\#3; +a; /\#3; -b; /\#3; +c; /\#4; -d; \backslash\#7; \backslash\#8; \backslash e$$

This is not the shortest instruction sequence that expresses the behaviour defined by this specification, e.g., removing the second last instruction and replacing /#4 with /#3 expresses the same behaviour.

Let $\mathsf{deg}^-_{\mathsf{P}}(P_i)$ be the number of equations in some linear specification P that have P_i on the right hand side of the equation. $\mathsf{deg}^-_{\mathsf{P}}(P_i)$ is called the indegree of P_i.

Theorem 4.2.1. *The minimum length of* TOP *instruction sequences required to model all regular threads of n states is $3n - \lceil \frac{n}{2} \rceil$.*

Proof. Given the construct of Equation 3.1 it is obvious that each state can be modelled by three instructions. Essentially if it possible to show that each pair of states can be modelled with at most five instructions the proof is complete. For a pair including a deadlock or termination constant this is trivial. Clearly, these states can be modelled by a single instruction and thus a pair including such a state needs at most four instructions.

Therefore, to find the upper bound only threads without constant states need to be considered. Given some thread with a linear specification $P = \{P_1{=}t_1, \ldots, P_n{=}t_n\}$. It is shown that when a state P_i is on the right hand side of two equations in the linear specification an instruction can be saved. The number of pairs that have this property for some equation P_i is the number of saved instructions for that equation. Thus the number k of saved instructions for the whole thread P is

$$k = \sum_{P_i = t_i \in P} \left\lfloor \frac{\mathsf{deg}^-_P(P_i)}{2} \right\rfloor.$$

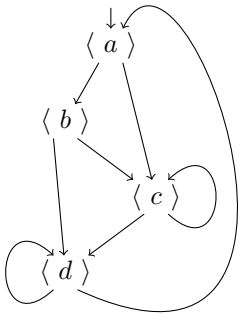
In order to find the upper bound on needed instructions, k needs to be minimised for an arbitrary thread. This is done by finding an integer partition of $2n$ into n parts with the maximum number of odd parts. This is trivial. For an even number of equations the indegree for each P_i can be odd. For an odd number of equations one indegree must be even. Therefore,

$$k = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even,} \\ \frac{n+1}{2} & \text{if } n \text{ is odd,} \end{cases} \quad \text{or} \quad k = \left\lceil \frac{n}{2} \right\rceil .$$

When including the constant states this still holds. Assume there is a thread of n states with $m \in \{0, 1, 2\}$ constant states. The thread can be coded in $3(n-m) - \lceil (n-m)/2 \rceil + m$ instructions. □

It can be shown that this is tight. Consider the regular thread defined by the linear specification below.

$$\begin{aligned} P_1 &= P_2 \trianglelefteq a \trianglerighteq P_3 \\ P_2 &= P_4 \trianglelefteq b \trianglerighteq P_3 \\ P_3 &= P_4 \trianglelefteq c \trianglerighteq P_3 \\ P_4 &= P_4 \trianglelefteq d \trianglerighteq P_1 \end{aligned}$$



A shortest instruction sequence that models this thread is of length $10 = 3 \cdot 4 - \lceil \frac{4}{2} \rceil$:

$$X = /\#3; +a; /\#3; -b; /\#4; +c; \backslash\#1; /\#1; +d; \backslash\#8$$

with $|X|_2 = P_1$.

A linear specification $P = \{P_i{=}t_i \mid i \in \{1, \ldots, n\}\}$ that defines a behaviour that requires at least $3n - \lceil \frac{n}{2} \rceil$ instructions to be expressed can be created for any $n \in \mathbb{N}^+$ where $n \neq 2$ as follows:

- For $n = 1$:

  $$P_1 = a \circ P_1$$

  The behaviour defined by this specification requires an instruction sequence of at least 2 instructions to be expressed, e.g., $|/a; \backslash\#1|_1 = P_1$.

- For $n = 3$:

  $$\begin{aligned} P_1 &= P_1 \trianglelefteq a \trianglerighteq P_2 \\ P_2 &= P_2 \trianglelefteq b \trianglerighteq P_3 \\ P_3 &= P_3 \trianglelefteq a \trianglerighteq P_1 \end{aligned}$$

  A shortest instruction sequence X that expresses this behaviour is

  $$X = /\#1; +a; /\#1; +b; /\#1; +a; \backslash\#4.$$

- For $n > 2$ the linear specification can be defined as $P = \{P_1{=}t_1, \ldots, P_n{=}t_n\}$ with

$$P_i = \begin{cases} P_1 \trianglelefteq a \trianglerighteq P_n & \text{if } i = 1, \\ \phi_n(i) & \text{if } 1 < i \quad \text{and } n = 0 \mod 4, \\ \phi_{n-1}(i) & \text{if } 1 < i < n \text{ and } n = 1 \mod 4, \\ \psi_n(i) & \text{if } 1 < i \quad \text{and } n = 2 \mod 4, \\ \psi_{n-1}(i) & \text{if } 1 < i < n \text{ and } n = 3 \mod 4, \\ P_n \trianglelefteq \chi(i) \trianglerighteq P_{n-1} & \text{otherwise,} \end{cases}$$

  where $\phi_n(\_), \psi_n(\_) : \mathbb{N}^+ \to \mathrm{BTA}^\infty$ is defined as

$$\phi_n(i) = \begin{cases} P_{2\lceil \frac{i}{3} \rceil - 1} \trianglelefteq \chi(i) \trianglerighteq P_{2\lceil \frac{i-1}{3} \rceil} & \text{if } i < \frac{3n}{4} + 1, \\ P_{2i-n-1} \trianglelefteq \chi(i) \trianglerighteq P_{2\lceil \frac{i-1}{3} \rceil} & \text{if } i = \frac{3n}{4} + 1 \\ P_{2i-n-1} \trianglelefteq \chi(i) \trianglerighteq P_{2i-n-2} & \text{otherwise,} \end{cases}$$

$$\psi_n(i) = \begin{cases} P_{2\lceil \frac{i}{3} \rceil - 1} \trianglelefteq \chi(i) \trianglerighteq P_{2\lceil \frac{i-1}{3} \rceil} & \text{if } i < \frac{3n-6}{4} + 2, \\ P_{2\lceil \frac{i}{3} \rceil - 1} \trianglelefteq \chi(i) \trianglerighteq P_{2i-n} & \text{if } i = \frac{3n+6}{4} + 2, \\ P_{2\lceil \frac{i}{3} \rceil - 1} \trianglelefteq \chi(i) \trianglerighteq P_{2i-n-1} & \text{if } i = \frac{3n+6}{4} + 3, \\ P_{2i-n-2} \trianglelefteq \chi(i) \trianglerighteq P_{2i-n-1} & \text{otherwise,} \end{cases}$$

  and $\chi(\_) : \mathbb{N}^+ \to \mathcal{A}$ is defined as

$$\chi(i) = \begin{cases} a & \text{if } i \text{ is even,} \\ b & \text{otherwise.} \end{cases}$$

Example 4.2.2. The linear specifications $\{P_1{=}t_1, \ldots, P_n{=}t_n\}$ for $n \in \{8, 9, 10, 11\}$ created with the equations above are defined below. Observe that each thread is created by distributing the arcs in such a way that the number of states with an odd indegree is maximised and that the indegree of all states is in $\{1, 2, 3\}$. Furthermore, all states in the threads defined by these specifications are unique and reachable from any other state in the specification.

$n = 8$	$n = 9$	$n = 10$	$n = 11$
$P_1 = P_1 \trianglelefteq a \trianglerighteq P_8$	$P_1 = P_1 \trianglelefteq a \trianglerighteq P_9$	$P_1 = P_1 \trianglelefteq a \trianglerighteq P_{10}$	$P_1 = P_1 \trianglelefteq a \trianglerighteq P_{11}$
$P_2 = P_1 \trianglelefteq a \trianglerighteq P_2$	$P_2 = P_1 \trianglelefteq a \trianglerighteq P_2$	$P_2 = P_1 \trianglelefteq a \trianglerighteq P_2$	$P_2 = P_1 \trianglelefteq a \trianglerighteq P_2$
$P_3 = P_1 \trianglelefteq b \trianglerighteq P_2$	$P_3 = P_1 \trianglelefteq b \trianglerighteq P_2$	$P_3 = P_1 \trianglelefteq b \trianglerighteq P_2$	$P_3 = P_1 \trianglelefteq b \trianglerighteq P_2$
$P_4 = P_3 \trianglelefteq a \trianglerighteq P_2$	$P_4 = P_3 \trianglelefteq a \trianglerighteq P_2$	$P_4 = P_3 \trianglelefteq a \trianglerighteq P_2$	$P_4 = P_3 \trianglelefteq a \trianglerighteq P_2$
$P_5 = P_3 \trianglelefteq b \trianglerighteq P_4$	$P_5 = P_3 \trianglelefteq b \trianglerighteq P_4$	$P_5 = P_3 \trianglelefteq b \trianglerighteq P_4$	$P_5 = P_3 \trianglelefteq b \trianglerighteq P_4$
$P_6 = P_3 \trianglelefteq a \trianglerighteq P_4$	$P_6 = P_3 \trianglelefteq a \trianglerighteq P_4$	$P_6 = P_3 \trianglelefteq a \trianglerighteq P_4$	$P_6 = P_3 \trianglelefteq a \trianglerighteq P_4$
$P_7 = P_5 \trianglelefteq b \trianglerighteq P_4$	$P_7 = P_5 \trianglelefteq b \trianglerighteq P_4$	$P_7 = P_5 \trianglelefteq b \trianglerighteq P_4$	$P_7 = P_5 \trianglelefteq b \trianglerighteq P_4$
$P_8 = P_7 \trianglelefteq a \trianglerighteq P_6$	$P_8 = P_7 \trianglelefteq a \trianglerighteq P_6$	$P_8 = P_5 \trianglelefteq a \trianglerighteq P_6$	$P_8 = P_5 \trianglelefteq a \trianglerighteq P_6$
	$P_9 = P_9 \trianglelefteq b \trianglerighteq P_8$	$P_9 = P_5 \trianglelefteq b \trianglerighteq P_7$	$P_9 = P_5 \trianglelefteq b \trianglerighteq P_7$
		$P_{10} = P_8 \trianglelefteq a \trianglerighteq P_9$	$P_{10} = P_8 \trianglelefteq a \trianglerighteq P_9$
			$P_{11} = P_{10} \trianglelefteq b \trianglerighteq P_{11}$

It can be shown that threads modelled by instruction sequences constructed as described above require at least $3n - \lceil n/2 \rceil$ instructions. Each state is unique and has two distinct directly reachable other states. Therefore, each state must be coded with a test instruction. Furthermore, there is no pair of two states where both states can directly reach the other state in the pair. Therefore, no two test instructions can be placed directly next to each other.

Now assume X is a minimal instruction sequence that expresses some behaviour as defined above for some arbitrary n.

- At least n instructions must be test instructions. If there are less than n, not all n states can be modelled by the sequence.
- Exactly $\lfloor n/2 \rfloor$ states are directly reachable from 1 other state. Consequently, these states require at least 1 jump instruction to be reached from the test instruction that models that other state.
- Furthermore, there are exactly $\lfloor n/2 \rfloor$ states that are directly reachable from 3 other states. These require at least 2 jump instructions to be directly reached. One pair of tests can share a jump instruction, the remaining test needs an additional jump instruction.
- Lastly, $(n \bmod 2)$ states are directly reachable from 2 other states. This can be modelled by a single jump instruction.

Summing up the individual instruction counts of each component the following equation is obtained

$$n + \left\lfloor \frac{n}{2} \right\rfloor + 2 \cdot \left\lfloor \frac{n}{2} \right\rfloor + (n \bmod 2)$$

which is equal to

$$3n - \left\lceil \frac{n}{2} \right\rceil .$$

Thus Theorem 4.2.1 is tight for at least $n \in \{4, 5, 6, \dots\}$. Proving tightness for $n = 1$ and $n = 3$ is trivial, but for $n = 2$ the bound does not seem reachable.

CHAPTER 5

Multidimensional programming languages

In previous chapters programs are considered to be sequences of instructions, i.e., 1-dimensional objects. Empirically, this might be a result of natural languages, which can also be considered 1-dimensional, e.g., written and spoken text can be represented as a 1-dimensional sequence of ASCII characters.

Although just because a language can be represented in 1-dimension, this does not implicate that 1-dimension is the natural representation for that language. Turing observed in [11] that the use of a second dimension is always avoidable and as such not essential for computation. This is found to be true in this chapter regarding the expressiveness of TOP and TOP_2. Nonetheless, Dershowitz and Dowek observed the relatively naturalness two-dimensional programming delivers in [6].

Ultimately, 2-dimensionality seems to come natural with human thinking [1]. Consider the illustrations used for the behaviour of regular threads. The intuitive nature of these images implies that behaviour might be easier to interpret in a 2-dimensional setting. Perhaps programming languages should make use of this feat.

Conceptually 2-dimensional programs are not a new idea. There are several approaches which can be separated into three main categories:

1. Block based languages,
    - Maloney et al. describe a visual programming environment where users can learn computer programming while working on personally meaningful projects [7]. The syntax and semantics of programs in the corresponding language are defined by a visual grammar of block shapes and their combination rules. Programs are made out of blocks that indicate control structure by their shape and colour.
2. Grid based languages,
    - Befunge[1] is a grid based language where specific characters change the direction of control flow. Programs were originally written on a bounded grid (Befunge-93) but in a later specification[2] this bound was removed making the language Turing complete.
    - Piet[3] is a programming language inspired by Piet Mondrian. The programs are defined by bitmaps, which are separated into colour blocks, connected pixels with the same colour. Execution starts with the colour block in the upper left corner. Transition

[1] C. Pressey. Befunge-93 documentation. `https://github.com/catseye/Befunge-93/blob/master/doc/Befunge-93.markdown`, [1993] 2012. Accessed: 2018-05-31

[2] C. Pressey. Funge-98 specification. `https://github.com/catseye/Funge-98/blob/master/doc/funge98.markdown`, [1998] 2018. Accessed: 2018-05-31.

[3] D. Morgan-Mar. Piet. `http://www.dangermouse.net/esoteric/piet.html`, 2008. Accessed: 2018-05-31.

rules between colours define which command to execute and the properties of the current colour block provide the arguments of executed commands.

3. Graph based languages.
   - Denert et al. proposed in [5] the notion that 2-dimensional programs are not written, but drawn. While the language uses blocks to define programs it is based on graph rewriting systems. Each block in this language contains a declarative and an operational section. The declarative part declares the structure of the data type. The operational part is drawn arbitrarily in 2-dimensional space. Control flow is defined by rules which are represented by edges.

Since most of these languages run on common hardware they must ultimately be translated into 1-dimensional machine code, indicating that the extra dimension does indeed not add any expressiveness to the languages in comparison to more traditional 1-dimensional languages. Furthermore, these language types are actually not that different. Ultimately, blocks are simply limitations to some grid and each graph could be structured in a gridlike layout.

This chapter introduces a 2-dimensional variant of TOP and studies the impact of this dimensional enrichment on the expressiveness of this language. This variant is called Thread-Program algebra in 2 dimensions (TOP_2).

5.1 Instruction planes

In a 2-dimensional setting programs cannot be represented with instruction sequences, thus the notion of *instruction planes* is introduced. The gridlike approach is chosen because it is closely related to instruction sequences. It allows a direct translation from instruction sequences to instruction planes, while such a translation would be much more complex for another approach.

Like instruction sequences these instruction planes are constructed with a set of instructions. The instructions of TOP_2 are analogous to the instructions of TOP. For each directed instruction of TOP there is a counterpart in the second dimension.

- Basic instructions: $/a, \backslash a, \uparrow a, \downarrow a$ for $a \in \mathcal{A}$,
- Jump instructions: $/\#k, \backslash\#k, \uparrow\#k, \downarrow\#k$ for $k \in \mathbb{N}^+$,
- Test instructions: $+a, -a, \updownarrow{+}a, \updownarrow{-}a$ for $a \in \mathcal{A}$.

The undirected instructions are equivalent to those in TOP.

- The termination instruction !,
- The abort instruction #.

Let $\mathcal{I}_2$ be the set of all instructions in TOP_2. Formally,

$$\mathcal{I}_2 = \bigcup_{a\in\mathcal{A}} \{/a, \backslash a, \uparrow a, \downarrow a, +a, -a, \updownarrow{+}a, \updownarrow{-}a\} \cup \bigcup_{k\in\mathbb{N}^+} \{/\#k, \backslash\#k, \uparrow\#k, \downarrow\#k\} \cup \{\#, !\}.$$

Instruction planes $\mathcal{I}_2^{m,n}$ are defined similar to instruction sequences in 1-dimension. An instruction plane $\mathcal{I}_2^{m,n}$ can be seen as matrix of order $m \times n$ where each element is an instruction.

$$\begin{aligned}
\mathcal{I}_2^{1,1} &= \mathcal{I}_2 \\
\mathcal{I}_2^{m,n+1} &= \{\left[X \mid Y\right] \mid X \in \mathcal{I}_2^{m,n}, Y \in \mathcal{I}_2^{m,1}\} \\
\mathcal{I}_2^{m+1,n} &= \{\begin{bmatrix} X \\ \hline Y \end{bmatrix} \mid X \in \mathcal{I}_2^{m,n}, Y \in \mathcal{I}_2^{1,n}\}
\end{aligned}$$

TOP_2 is defined as the union over $\mathcal{I}_2^{m,n}$ for all $m, n \in \mathbb{N}^+$

$$\mathrm{TOP}_2 = \bigcup_{n,m \in \mathbb{N}^+} \mathcal{I}_2^{m,n}.$$

Unlike instruction sequences, the instructions in instruction planes are not concatenated with some operator. Instruction planes are denoted with square brackets and instructions must be vertically and horizontally aligned by their indices. Extraction can be done at an arbitrary position $n, m \in \mathbb{Z}$. Each directed instruction moves the control by adjusting the position in a single dimension. If control moves outside of the instruction plane deadlock occurs.

The extraction operator of TOP_2 is defined as $|\_|_{i,j} : \mathrm{TOP}_2 \times \mathbb{Z} \times \mathbb{Z} \to \mathrm{BTA}^\infty$

$$|X|_{i,j} = \begin{cases} a \circ |X|_{i,j+1} & \text{if } \sigma_2(X,i,j) = /a, \\ a \circ |X|_{i,j-1} & \text{if } \sigma_2(X,i,j) = \backslash a, \\ a \circ |X|_{i+1,j} & \text{if } \sigma_2(X,i,j) = \uparrow a, \\ a \circ |X|_{i-1,j} & \text{if } \sigma_2(X,i,j) = \downarrow a, \\ |X|_{i,j+k} & \text{if } \sigma_2(X,i,j) = /\#k, \\ |X|_{i,j-k} & \text{if } \sigma_2(X,i,j) = \backslash \#k, \\ |X|_{i+k,j} & \text{if } \sigma_2(X,i,j) = \uparrow \#k, \\ |X|_{i-k,j} & \text{if } \sigma_2(X,i,j) = \downarrow \#k, \\ |X|_{i,j-1} \trianglelefteq a \trianglerighteq |X|_{i,j+1} & \text{if } \sigma_2(X,i,j) = +a, \\ |X|_{i,j+1} \trianglelefteq a \trianglerighteq |X|_{i,j-1} & \text{if } \sigma_2(X,i,j) = -a, \\ |X|_{i-1,j} \trianglelefteq a \trianglerighteq |X|_{i+1,j} & \text{if } \sigma_2(X,i,j) = \updownarrow +a, \\ |X|_{i+1,j} \trianglelefteq a \trianglerighteq |X|_{i-1,j} & \text{if } \sigma_2(X,i,j) = \updownarrow -a, \\ \mathsf{D} & \text{if } \sigma_2(X,i,j) = \#, \\ \mathsf{S} & \text{if } \sigma_2(X,i,j) = !, \\ & \\ \mathsf{D} & \text{if } \sigma_2(X,i,j) = \bigcirc, \end{cases}$$

where $a \in \mathcal{A}$, $k \in \mathbb{N}^+$, and $\sigma_2(\_,\_,\_) : \mathrm{TOP}_2 \times \mathbb{Z} \times \mathbb{Z} \to \mathcal{I} \cup \{\bigcirc\}$ is given by

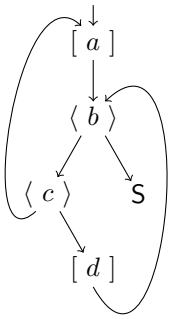
$$\sigma_2(X,i,j) = \begin{cases} u_{i,j} & \text{if } 0 < i \leq m \text{ and } 0 < j \leq n, \\ \bigcirc & \text{otherwise,} \end{cases}$$

for $X = \begin{bmatrix} u_{1,1} & \cdots & u_{1,n} \\ \vdots & \ddots & \vdots \\ u_{m,1} & \cdots & u_{m,n} \end{bmatrix}$.

Formally, instruction planes cannot contain empty entries. For convenience cells of the plane that contain abort instructions that are not reachable from any other instruction are not filled in. Here reachability is defined as expected, equivalent to the definition for instruction sequences.

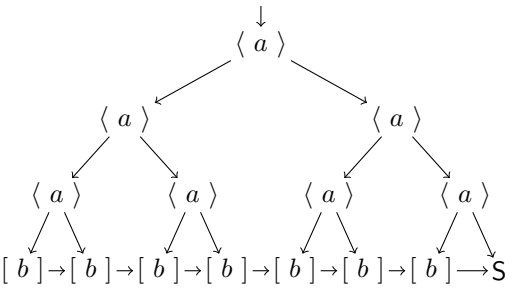
Example 5.1.1. Let $X = \begin{bmatrix} /a & \downarrow\#1 \\ \updownarrow -c & +b \quad ! \\ /d & \uparrow\#1 \end{bmatrix}$. The behaviour extracted with $|X|_{1,1} = P_{1,1}$ is defined by the linear specification below.

$$
\begin{aligned}
P_{1,1} &= a \circ P_{2,2} \\
P_{2,1} &= P_{1,1} \trianglelefteq c \trianglerighteq P_{3,1} \\
P_{2,2} &= P_{2,1} \trianglelefteq b \trianglerighteq P_{2,3} \\
P_{2,3} &= \mathsf{S} \\
P_{3,1} &= d \circ P_{2,2}
\end{aligned}
$$



Example 5.1.2.
Let $X = \begin{bmatrix} & & \downarrow\#1 & & \backslash\#2 & +a & /\#2 & & \downarrow\#1 & & & & \\ & \downarrow\#1 & +a & /\#1 & \downarrow\#1 & & & \downarrow\#1 & +a & /\#1 & \downarrow\#1 & & \\ \downarrow\#1 & +a & \downarrow\#1 & \downarrow\#1 & +a & \downarrow\#1 & \downarrow\#1 & +a & \downarrow\#1 & \downarrow\#1 & +a & \downarrow\#1 \\ /b & /\#1 & /b & /b & /\#1 & /b & /b & /\#1 & /b & /b & /\#1 & ! \end{bmatrix}$.
The extracted thread $|X|_{1,6} = P$ is defined as:

$$
\begin{aligned}
P &= P_{\mathtt{t}} \trianglelefteq a \trianglerighteq P_{\mathtt{f}} \\
P_{\mathtt{t}} &= P_{\mathtt{tt}} \trianglelefteq a \trianglerighteq P_{\mathtt{tf}} \\
P_{\mathtt{f}} &= P_{\mathtt{ft}} \trianglelefteq a \trianglerighteq P_{\mathtt{ff}} \\
P_{\mathtt{tt}} &= P_7 \trianglelefteq a \trianglerighteq P_6 \\
P_{\mathtt{ft}} &= P_5 \trianglelefteq a \trianglerighteq P_4 \\
P_{\mathtt{tf}} &= P_3 \trianglelefteq a \trianglerighteq P \\
P_{\mathtt{ff}} &= P_1 \trianglelefteq a \trianglerighteq P_0 \\
P_i &= b^i \circ \mathsf{S} \quad \text{for } i \in \{0, \ldots, 7\}
\end{aligned}
$$



It is striking how structurally similar an instruction plane can be to the graphical representation of the thread it models.

5.2 Expressiveness results

It is unsurprising that TOP_2 instruction planes must at least be equally expressive as TOP instruction sequences since TOP_2 has a corresponding instruction plane to all instruction sequences contained in TOP. Formally, there is an injection from $\mathcal{I}^n$ to $\mathcal{I}^{1,n}$ for all $n \in \mathbb{N}^+$.

Given an instruction sequence $X = u_1; \ldots; u_n$ an instruction plane Y can be constructed with $Y = \begin{bmatrix} I(u_1) & \cdots & I(u_n) \end{bmatrix}$ where I is the identity function.

The opposite is done by concatenating the individual rows of some matrix into a single sequence

while transforming 2-dimensional instructions. Given an instruction plane $X = \begin{bmatrix} u_{1,1} & \cdots & u_{1,n} \\ \vdots & \ddots & \vdots \\ u_{m,1} & \cdots & u_{m,n} \end{bmatrix}$.
An instruction sequence $Y = Y_{1,1}; Y_{1,2}; \ldots; Y_{m,n}$ can be constructed as follows:

$$Y_{i,j} = \begin{cases} \#; /a; \mathcal{J}(3n-1) & \text{if } u_{i,j} = \uparrow a, \\ \mathcal{J}(-3n+1); \backslash a; \# & \text{if } u_{i,j} = \downarrow a, \\ \mathcal{J}(-3n+1); +a; \mathcal{J}(3n-1) & \text{if } u_{i,j} = \updownarrow +a, \\ \mathcal{J}(3n+1); -a; \mathcal{J}(-3n-1) & \text{if } u_{i,j} = \updownarrow -a, \\ \#; \mathcal{J}(3kn); \# & \text{if } u_{i,j} = \uparrow \#k, \\ \#; \mathcal{J}(-3kn); \# & \text{if } u_{i,j} = \downarrow \#k, \\ \#; \mathcal{J}(3k); \# & \text{if } u_{i,j} = /\#k \text{ and } j+k \leq n, \\ \#; \#; \# & \text{if } u_{i,j} = /\#k \text{ and } j+k > n, \\ \#; \mathcal{J}(-3k); \# & \text{if } u_{i,j} = \backslash\#k \text{ and } j-k > 0, \\ \#; \#; \# & \text{if } u_{i,j} = \backslash\#k \text{ and } j-k \leq 0, \\ \backslash\#2; I(u_i); /\#2; & \text{otherwise,} \end{cases}$$

where $I(\_) : \mathcal{I}_2 \to \mathcal{I}$ is the identity function and $\mathcal{J}(i)$ is a relative jump in either the forward or backward direction:

$$\mathcal{J}(i) = \begin{cases} /\#i & \text{if } i > 0, \\ \backslash\# - i & \text{if } i < 0. \end{cases}$$

Since TOP sequences can be translated into TOP_2 planes and vice versa TOP_2 and TOP are equally expressive. This confirms, at least for TOP and TOP_2, that adding another dimension does not increase expressiveness. Yet the additional dimension does have an effect on the expressiveness. While arbitrarily large jump counters are required in TOP to express all regular threads, this is not necessary in TOP_2.

Theorem 5.2.1. *Let* $\text{TOP}_2^{\leq 2}$ *be the subset of* TOP_2 *that does not contain jump instructions with a jump counter greater than 2.*

$\text{TOP}_2^{\leq 2}$ *instruction sequences can model all regular threads.*

Proof. Given the following construction any regular thread can be modelled in a TOP_2 instruction plane.

$$X_{1,i} = \begin{cases} \# \quad ! \quad \# & \text{if } P_i = \mathsf{S}, \\ \# \quad \# \quad \# & \text{if } P_i = \mathsf{D}, \\ \mathcal{J}(3(j-i)+1) \quad +a \quad \mathcal{J}(3(k-i)-1) & \text{if } P_i = P_j \trianglelefteq a \trianglerighteq P_k \end{cases} \tag{5.1}$$

where $\mathcal{J}(i)$ is a relative jump in either the forward or backward direction:

$$\mathcal{J}(i) = \begin{cases} /\#i & \text{if } i > 0, \\ \backslash\# - i & \text{if } i < 0. \end{cases} \tag{5.2}$$

Observe that this is almost equivalent to Equation 3.1, i.e., only a single row of the instruction plane is needed to model all regular threads. Other properties of this construct are that there are no chained jumps or jumps going outside of the instruction plane. Finally, there are no test instructions that directly reach another test instruction.

It is clear that any regular thread can be modelled by an instruction plane using only a single row. To complete the proof it needs to be shown that jumps with a jump counter greater than 2 can be replaced with a path of chained shorter jumps in other rows of the instruction plane.

Assume an instruction plane $X = \begin{bmatrix} u_1 & \dots & u_n \end{bmatrix}$ is created by Equation 5.1. An instruction plane $Y = \begin{bmatrix} Y_{1,1} & \dots & Y_{1,n} \\ \vdots & \ddots & \vdots \\ Y_{\frac{2}{3}n,1} & \dots & Y_{\frac{2}{3}n,1} \end{bmatrix}$ that uses no jump instructions with a jump counter greater than 2 can be created with the following equations.

$$Y_{i,j} = \begin{cases} \phi(i,j) & \text{if } i \text{ is odd,} \\ \psi(i,j) & \text{otherwise,} \end{cases} \tag{5.3}$$

with $\phi(\_,\_), \psi(\_,\_) : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathcal{I}_2$ defined as

$$\phi(i,j) = \begin{cases} \begin{cases} u_{i,j} & \text{if } i = 1, \\ \uparrow\#1 & \text{if } i > 1, \end{cases} & \text{if } j = 2 \mod 3, \\ \begin{cases} \mathcal{J}(i,j,k) & \text{if } \sigma(X,1,j) = /\#k, \\ \mathcal{J}(i,j,-k) & \text{if } \sigma(X,1,j) = \backslash\#k, \end{cases} & \text{otherwise,} \\ \# & \text{otherwise,} \end{cases} \tag{5.4}$$

$$\psi(i,j) = \begin{cases} \begin{cases} \backslash\#2 & \text{if } 3j < 2i, \\ \backslash\#1 & \text{if } 3j = 2i, \\ /\#1 & \text{if } 3j > 2i, \end{cases} & \text{if } j = 0 \mod 3, \\ \begin{cases} \backslash\#1 & \text{if } 3j < 2i + 2, \\ /\#1 & \text{if } 3j = 2i + 2, \\ /\#2 & \text{if } 3j > 2i + 2, \end{cases} & \text{if } j = 1 \mod 3, \\ \uparrow\#1 & \text{if } j = 2 \mod 3, \end{cases} \tag{5.5}$$

and $\mathcal{J}(\_,\_,\_) : \mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathcal{I}_2$

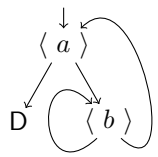
$$\mathcal{J}(i,j,k) = \begin{cases} \downarrow\#2 & \text{if } j + k > \frac{3}{2}(i+1) - 1, \\ \downarrow\#1 & \text{if } j + k = \frac{3}{2}(i+1) - 1, \\ \# & \text{otherwise.} \end{cases} \tag{5.6}$$

Using Equation 5.1-5.6 every regular thread can be modelled by $\mathrm{TOP}_2^{\leq 2}$ instruction plane. □

While the previous proof shows a systematic approach to construct an instruction plane without jump counters greater than 2 for any regular thread, it is not very intuitive. It might be more instructive to approach the problem in a different manner. It is evident that each regular thread can be represented as a directed graph in a 2-dimensional image. Now it is not hard to see that any graph can be discretised into an arbitrarily large but finite instruction plane by replacing each node with a test instruction and each edge with a series of jump instructions.

Example 5.2.1. Let a thread P_1 be defined with the following linear specification,

$$\begin{aligned} P_1 &= P_2 \trianglelefteq a \trianglerighteq P_3 \\ P_2 &= \mathsf{D} \\ P_3 &= P_3 \trianglelefteq b \trianglerighteq P_1 \end{aligned}$$



Applying Equations 5.1-5.2 a 1-dimensional instruction plane $X \in \mathcal{I}^{1,9}$ is obtained,

$$X = \begin{bmatrix} /\#4 & +a & /\#5 & \# & \# & \# & /\#1 & +b & \backslash\#7 \end{bmatrix}.$$

Applying Equations 5.3-5.6 to X gives $Y \in \mathcal{I}^{6,9}$,

$$Y = \begin{bmatrix} \downarrow\#2 & +a & \downarrow\#2 & & \# & & \downarrow\#2 & +b & \downarrow\#1 \\ /\#1 & \uparrow\#1 & \backslash\#1 & \backslash\#1 & \uparrow\#1 & \backslash\#2 & \backslash\#1 & \uparrow\#1 & \backslash\#2 \\ \downarrow\#1 & \uparrow\#1 & \downarrow\#2 & & \uparrow\#1 & & \downarrow\#2 & \uparrow\#1 & \\ /\#2 & \uparrow\#1 & /\#1 & /\#1 & \uparrow\#1 & \backslash\#1 & \backslash\#1 & \uparrow\#1 & \backslash\#2 \\ & \uparrow\#1 & \downarrow\#1 & & \uparrow\#1 & & \downarrow\#1 & \uparrow\#1 & \\ /\#2 & \uparrow\#1 & /\#1 & /\#2 & \uparrow\#1 & /\#1 & /\#1 & \uparrow\#1 & \backslash\#1 \end{bmatrix},$$

where $|Y|_{1,2} = P_1$. Compared to the graphical representation the behaviour is hard to interpret from this instruction plane. Informally, the graphical representation could simply be copied into an instruction plane presuming the plane is large enough.

$$Y' = \begin{bmatrix} & & \downarrow\#1 & \backslash\#1 & \backslash\#1 & \backslash\#1 \\ & \downarrow\#1 & +a & \downarrow\#1 & & \uparrow\#1 \\ \downarrow\#1 & \backslash\#1 & & /\#1 & \downarrow\#1 & \uparrow\#1 \\ \# & & & \uparrow\#1 & +b & \uparrow\#1 \end{bmatrix}$$

When considering multidimensional programming languages one might also study a 3-dimensional variant of TOP. It is easy to see that in this variant jump counters could be limited to 1 while still being able to represent all regular threads.

Finally, a language with an arbitrary amount of dimensions would probably not need jumps at all. For each state, control can be moved in a unique dimension.

CHAPTER 6

Conclusions

Several properties of TOP are studied in this thesis. All TOP instruction sequences model a regular thread and all regular threads can be modelled by a TOP instruction sequence. Only a subset of all instructions in TOP is required to model all regular threads but arbitrarily large jumps in both directions need to be in this set.

In also shown that the behaviour expressed by a TOP instruction sequence of length n can be defined by a linear specification of $n + 1$ equations. Conversely, a regular thread specified by a linear specification of n equations can be expressed by a TOP instruction sequence of length $3n - \lceil \frac{n}{2} \rceil$.

In Chapter 5 a 2-dimensional alternative to TOP is introduced. TOP_2 is equally expressive as TOP but does not require arbitrarily large jump counters to model all regular threads. TOP_2 could be an interesting field of study. It shows many levels of symmetry, e.g., one could find behaviour preserving automorphisms that mirror and rotate programs in TOP_2.

6.1 Acknowledgements

First and foremost, I wish to thank my supervisors, dr. I. Bethke and dr. A. Ponse. They have been supporting and advising me continuously throughout the works of this thesis. Their advice and feedback has been paramount in completing this thesis.

Second, I want to thank my family and friends for their unconditional support they have given me in the past years. Without those closest to me I would not be the person I am today.

Bibliography

[1] R. Arnheim. *Visual thinking*. University of California Press, 1969.

[2] J.A. Bergstra and M.E. Loots. Program algebra for component code. *Formal Aspects of Computing*, 12(1):1–17, 2000. ISSN 1433-299X. doi: 10.1007/PL00003928. URL `https://doi.org/10.1007/PL00003928`.

[3] J.A. Bergstra and A. Ponse. An instruction sequence semigroup with involutive anti-automorphisms. *Scientific Annals of Computer Science*, 19:57–92, 2009. ISSN 1843-8121.

[4] S. Bouber. On the length of instruction sequences for C. 2015. Bachelor thesis: `https://esc.fnwi.uva.nl/thesis/centraal/files/f881113803.pdf`.

[5] E. Denert, R. Franck, and W. Streng. Plan2d — towards a two-dimensional programming language. In D. Siefkes, editor, *Gl-4.Jahrestagung*, pages 202–213. Springer Berlin Heidelberg, 1975. ISBN 978-3-540-37424-4.

[6] N. Dershowitz and G. Dowek. Universality in two dimensions. *Journal of Logic and Computation*, 26(1):143–167, 2016.

[7] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *Trans. Comput. Educ.*, 10(4):16:1–16:15, November 2010. ISSN 1946-6226. doi: 10.1145/1868358.1868363. URL `http://doi.acm.org/10.1145/1868358.1868363`.

[8] A. Ponse and M.B. van der Zwaag. An introduction to program and thread algebra. In A. Beckmann, U. Berger, and J.V. Tucker, editors, *Logical Approaches to Computational Barriers: Proceedings CiE 2006*, pages 445–458. Springer-Verlag, 2006.

[9] M.G. Redder. On finite projections and program length in thread and program algebra. 2017. Bachelor thesis: `https://staff.fnwi.uva.nl/a.ponse/ThesisRedder.pdf`.

[10] S.H.P. Schroevers. Expressiveness and extensions of an instruction sequence semigroup. *arXiv preprint arXiv:1003.1572*, 2010.

[11] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(1):230–265, 1937.