A state vector algebra for algorithmic implementation of second-order logic

D. Lesnik, T. Schaefer

December 10, 2013

Abstract

We present a mathematical framework for mapping second-order logic relations onto a simple state vector algebra. Using this algebra, basic theorems of set theory can be proven in an algorithmic way, hence by an expert system. We illustrate the use of the algebra with simple examples and show that, in principle, all theorems of basic set theory can be recovered in an elementary way. The developed technique can be used for an automated theorem proving in the 1st and 2nd order logic.

Keywords: algorithmic implementation of second-order logic; expert systems; computational set theory; automated theorem proving

1 Introduction

The aim of this work is develop a method to implement an automated theorem proving for higher-order logic in an algorithmic way. For this purpose, we construct a so-called logical S-algebra that consists of events, associated state-vectors, and theorems. Events can be true or false and we say that they have the value 1 if they are true, and 0 if they are false. For clarity, in the text below, we put events in a box. For instance, $E \equiv [x \in A]$ denotes the event that $x$ is an element of $A$. If $x \in A$ is true, we say $E$ has the value 1, if $x \in A$ is false, the value of $E$ is 0 (we write $E = 1$ or $E = 0$ accordingly). Note, that it is also possible to develop a probabilistic version of the algorithm described below. In this work, however, we restrict ourselves to the deterministic version.

We (i.e. the expert system) might not know a priori whether an event is true or false and, in this case, we say that it has the value “-1”. We can then reformulate our aim as follows: We will describe an algorithm that, under certain assumptions that will be clarified below, can lead to a change of the value of an event from -1 to either 1 (true) or 0 (false). Assume for instance, we would like a computer to give an abstract proof of the statement that $A \subseteq B$ and $B \subseteq C$ implies $A \subseteq C$. Then, the computer needs to ‘understand’ the concept $\subseteq$ and must be able to apply this...
understanding to events in the following way: Assuming that the event \( a \in A \) is true, and that the value of the event \( a \in C \) is not known (hence -1 for now), it needs to check whether there is sufficient information available to deduce that the event \( a \in C \) is true (for any \( a \) that is an element of the set \( A \)). We emphasize that this checking should not happen elementwise but in a rather abstract way, in fact the same way our reasoning works in order to prove the above statement. In other words: in an appropriate framework, our reasoning of carrying out mathematical proof can be mapped onto an algorithm in a way that an appropriately instructed machine can carry out exactly the same proof.

The main tool to implement this idea will be the construction of a state vector, which represents the collection of values for a set of events. From there, it is straightforward to define appropriate operations (e.g. a conflict operation) and propositional logic. The implementation of first and second-order logic requires the construction of a so-called virtual point that allows the algorithm to deal with statements that use quantifiers \( \forall \) and \( \exists \). The virtual points will operate along with new objects of the S-algebra – triggers and terminators. The main idea is, in some sense, to mimic our own thinking in the following way: If we are asked to prove a statement, we will check the statement against our knowledge. For a statement of the type above involving the quantifier \( \forall x \), we will check whether, within our known universe, we can find an \( x \) for which the statement is not true. If we can say that such an \( x \) does not exist, we will assume the above statement to be true.

The main purpose of any abstract proof is to not have to check the statement for all \( x \) individually but rather applying known theorems instead if possible. Our claim is that this can be done in an algorithmic way. This, still, entails that whether we decide if a statement is true or false depends on our current knowledge and we even might decide that we are not able to determine whether an event is true or false and that we need additional information in order to make this decision.

We also should remark that, in the present work, we are not interested to characterize the set of statements that are provable in the framework of our algorithm. In fact, this set will depend on the known universe and is subject to change while the algorithm is learning. Nor we discuss any problems that arise from computational issues due to combinatorial growth of the size of the problem. Our aim is merely to provide a simple and efficient practical algorithm that mimics, to a certain extent, our own thinking at the most basic level.

It is worth emphasising that S-algebra provides a computational engine for a 1st and 2nd order logic, but it is not an implementation of the logic itself. In this paper we are not focusing on any particular implementation of the logical system. We only suppose that it is rich enough to handle the statements of higher order logic.

We use the set-theoretical terminology to demonstrate the mechanics of the S-algebra. This language allows to express an arbitrary logic theory using the standard correspondence: for any predicate \( P(x) \) we can introduce a space \( A_P \), such that for any point \( x \) the statement \( P(x) \) can be equivalently expressed as \( x \in A_P \), where \( A_P \) is a set of all points \( x \) for which \( P(x) \) is true.
2 Basic definitions

First we introduce a space of events. An event is anything that can be assigned a truth value: 1 (true) or 0 (false). Events represent propositional symbols, logical formulas, sentences, etc.

Let $\mathcal{E} = \{E_1, \ldots, E_N\}$ be a finite system of events consisting of $N$ elements. Let us assign a particular value to every event, and let $e_i$ be a value of $i$th event. An array

$$ s = \{e_1, \ldots, e_N\}, \quad e_i \in \{1, 0\} $$

represents a state of $\mathcal{E}$. Obviously $\mathcal{E}$ can have $2^N$ different states.

A matrix whose rows are interpreted as states we call state vector or shorter – $S$-vector:

$$ s = \begin{pmatrix} e_{11} & e_{12} & \cdots & e_{1N} \\ e_{21} & e_{22} & \cdots & e_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \cdots & e_{mN} \end{pmatrix}. $$

By definition there are no identical states in the state vector. The number of states $m$ in the state vector satisfies $0 \leq m \leq 2^N$. If $m = 0$ we say that the state vector is empty. If $m = 2^N$ we say that the state vector is trivial (the reason for this name will become clear later) and designate it as $t$.

We refer to the number of rows $m$, as the the size of the state vector.

Let us consider a subset $\mathcal{K}$ of the system of events

$$ \mathcal{K} = \{E_{i_1}, \ldots, E_{i_m}\} \subset \mathcal{E}. $$

The state vector obtained by selecting the columns $i_1, \ldots, i_m$ of the original state vector is called a state vector constrained to the subset $\mathcal{K}$.

An event is said to be identically true, if the corresponding column in the state vector only contains value 1 (this corresponds to the tautology in propositional logic). Similarly an event is said to be identically false, if the corresponding column in the state vector only contains value 0 (this corresponds to the unsatisfiable sentence in propositional logic). If a column contains both values 0 and 1, then the corresponding even is said to have an indefinite value.

The purpose of the theorem proving is usually reduced to proving some target event true or false. In terms of $S$-algebraic representation this implies that the theorem is proven when the the target event becomes identically true or false.

2.1 Information state vector

If there are some relations imposed on the events, then not every state becomes possible. We say that a state $s$ is allowed by the system, if it does not contradict any relation between the events. For instance, if $E_1 = E_2$ then the state

$$ s = \{1 \ 0 \ e_3 \ \cdots \ e_N\}. $$
is not allowed by the system.

Relations impose restriction on the possible combinations between event values. Every relation taken into account makes the number of allowed states smaller: the states which previously were allowed may become prohibited. The opposite is impossible: any state which is prohibited can never become allowed due to any new relation taken onto account. We thus can say that the information implied in a relation between events is contained in the prohibited states.

If \( s \) is a state vector, then the vector

\[
\bar{s} = t \setminus s
\]  

(2.1)

is called information state vector corresponding to \( s \). If \( s \) contains allowed states, then \( \bar{s} \) contains all states that are prohibited by \( s \).

The trivial state vector does not prohibit any state, and thus contains no information about the system. The empty state vector prohibits all states. An empty state vector indicates a logical infeasibility.

### 2.2 System state vector

A state vector, which contains all states that do not contradict any of the relations between system events, we call an allowed state space. The allowed state space is unique, however it is not always easy (or even possible) to be identified, especially if the size of the system becomes infinite. If the system of events becomes infinite, we only consider finite-size subsets of it.

In the propositional logic the number of events and the relations between events is finite, and all of them can be taken into account. Thus for the propositional logic the allowed state space can be calculated exactly. It simply represents the truth table for the logical system.

For the first and higher order logic it is usually not possible to take all relations into account, since the number of events and relations may become infinite, or just too large to be processed numerically in a reasonable time.

For the infinite system we can only take a finite subset of the relations into account. Let \( \aleph \) be some finite subset of all possible relations in the system. A state vector, which contains all states allowed by \( \aleph \) is called system state vector.

In the process of logical inference new relations are generated and added to the system. This new information leads to a consecutive elimination of allowed states, and the system state vector approaches the allowed state space. The system state vector equals the allowed state space only if all possible relations are taken into account. In general case the system state vector contains an upper bound of the allowed state space.
2.3 Unknown event value

Here we introduce a useful notation which allows to significantly simplify the state
vector representation. Let some event (let it for example be the first event \( E_1 \)) has
unknown value. This means that it can be either true or false. If all other events
have certain values, then the system state vector contains two states:

\[
s = \begin{cases} 
0 & e_2 \ldots e_N \\
1 & e_2 \ldots e_N 
\end{cases}.
\]

We introduce the “unknown” event value, which we designate as “-1”. By definition
if a state in the state vector contains value -1 at the \( i \)-th position, then it is equivalent
to two states, having 1 and 0 in the \( i \)-th position, and identical otherwise. For
instance, the latter state vector can be represented as

\[
s = \{ -1 \ e_2 \ e_3 \ldots e_N \} = \begin{cases} 
0 & e_2 \ e_3 \ldots e_N \\
1 & e_2 \ e_3 \ldots e_N 
\end{cases}.
\]

For two unknown events \( e_1 = -1 \) and \( e_2 = -1 \) we have

\[
s = \{ -1 \ -1 \ e_3 \ldots e_N \} = \begin{cases} 
0 & 0 \ e_3 \ldots e_N \\
0 & 1 \ e_3 \ldots e_N \\
1 & 0 \ e_3 \ldots e_N \\
1 & 1 \ e_3 \ldots e_N 
\end{cases}.
\]

This rule is straightforward to generalise for the case of arbitrary number of “-1”
entries. If a row in a state vector contains \( m \) entries “-1”, then it is equivalent to
\( 2^m \) different states.

With the introduced notation the trivial state vector can be written in a form

\[
t = \{ -1 \ -1 \ldots -1 \}.
\]

We can now introduce two transformations which can be performed with a state
vector – expansion and reduction. We say that the state vector is expanded, if it
contains no “-1” values (and no duplicated rows). We say that the state vector is
reduced if the number of rows is reduced to minimum by using the “-1” values. If
the state vector is reduced, then no two rows in it can be replaced by an equivalent
one row using “-1” value. We point out that the reduced form of a state vector is
generally not unique. For instance

\[
\begin{pmatrix} 
1 & 1 \\
1 & 0 \\
0 & 1 
\end{pmatrix} = \begin{pmatrix} 
1 & -1 \\
0 & 1 
\end{pmatrix} = \begin{pmatrix} 
-1 & 1 \\
1 & 0 
\end{pmatrix}
\]

In the reduced form some columns of the state vector may contain only “-1” values.
If this is the case for the \( i \)-th event, we say that the corresponding column is trivial.
Though the reduced form is usually not unique, in any reduced state all trivial
columns are the same.
Those columns which are not trivial, are called *pivot*. Identifying pivot columns will be useful for the optimisation of the algebraic transformations of S-vectors.

The size (the number of states) of the S-vector is independent on whether the vector is in reduced or expanded form, since the number of the states is calculated as a number of rows in expanded representation.

### 2.4 Logical operators

State vectors allow to express logical relations between events. We consider below the basic logical relations, and show how they can be expressed by means of the state vectors.

Let for instance the three events $E_1$, $E_2$ and $E_3$ obey a logical conjunction relation

$$E_1 \equiv (E_2 \land E_3).$$

(2.2)

where $\equiv$ means “is by definition”. In terms of the state vector this relation can be expressed as

$$\begin{cases}
1 & 1 & 1 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0
\end{cases}
\quad \text{or equivalently} \quad
\begin{cases}
1 & 1 & 1 \\
0 & 1 & 0 \\
0 & 0 & -1
\end{cases}.

(2.3)

If we have the system of $N$ events $\{E_1, \ldots, E_N\}$, then the latter state vector should be extended to the remaining events using “-1” (trivial) columns:

$$\begin{cases}
1 & 1 & 1 & -1 & \ldots & -1 \\
0 & 1 & 0 & -1 & \ldots & -1 \\
0 & 0 & -1 & -1 & \ldots & -1
\end{cases}.

(2.3)

It is straightforward to find the state vectors for all basic logical connectives:

- **Negation:**
  $$E_1 \equiv \neg E_2; \quad \begin{cases}
1 & 0 & -1 & \ldots & -1 \\
0 & 1 & -1 & \ldots & -1
\end{cases};

(2.4)

- **Conjunction:**
  $$E_1 \equiv (E_2 \land E_3); \quad \begin{cases}
1 & 1 & 1 & -1 & \ldots & -1 \\
0 & 1 & 0 & -1 & \ldots & -1 \\
0 & 0 & -1 & -1 & \ldots & -1
\end{cases}

(2.5)

- **Disjunction:**
  $$E_1 \equiv (E_2 \lor E_3); \quad \begin{cases}
1 & 1 & -1 & -1 & \ldots & -1 \\
0 & 1 & -1 & -1 & \ldots & -1 \\
0 & 0 & 0 & -1 & \ldots & -1
\end{cases}

(2.6)
• Material implication:

\[ E_1 \equiv (E_2 \rightarrow E_3) ; \begin{cases} 1 & 1 & 1 & -1 & \ldots & -1 \\ 1 & 0 & -1 & -1 & \ldots & -1 \\ 0 & 1 & 0 & -1 & \ldots & -1 \end{cases} \] (2.7)

• Biconditional:

\[ E_1 \equiv (E_2 \leftrightarrow E_3) ; \begin{cases} 1 & 1 & 1 & -1 & \ldots & -1 \\ 1 & 0 & 0 & -1 & \ldots & -1 \\ 0 & 1 & 0 & -1 & \ldots & -1 \\ 0 & 0 & 1 & -1 & \ldots & -1 \end{cases} \] (2.8)

If events involved into the relations have indices other than in the examples above, the pivot columns should be placed in the corresponding positions.

Notice that in the propositional logic one usually considers such logical connectives, that are boolean functions of the involved events, ie, the value of \( E_1 \) is always uniquely defined by the values of \( E_2 \) and \( E_3 \). In the state vector representation this requirement is irrelevant. A state vector simply imposes some restriction on the possible combinations of all three involved events.

### 2.5 Supplementary events

Compound logical formulas can be represented by introduction of the supplementary events. Consider for instance the following formula

\[ E_1 \equiv (E_2 \land (E_3 \lor E_4)). \]

For the inner bracket we can introduce a supplementary event \( E_s \), such that

\[ E_s \equiv (E_3 \lor E_4) ; \]

\[ E_1 \equiv (E_2 \land E_s). \]

For an arbitrary formula we can introduce one supplementary event for each bracket and possibly for the negation operator. This allows us to express any formula by means of a small number of predefined state vectors, which represent the standard logical connectives.

The supplementary events can be removed from the system at some stage, as described at the end of Sec. 3.4.

### 2.6 Operations on state vectors

If we consider a state vector as a set of states, then the state vectors allow standard operations, eg, union, intersection, subtraction, etc.

\[ s \cup r, \quad s \cap r, \quad s \setminus r. \]
The intersection operation is of a special importance in the S-algebra. We refer to the intersection of state vectors as conflict. It is considered in more details in Sec. 2.6.1.

Next we define a projection operation, which defines a sub-vector corresponding to some event value. Let $s$ be some state vector. We designate $s(i)$ the sub-vector of $s$ which corresponds to the event value $E_i = 1$. Similarly $s(i)$ corresponds to $E_i = 0$. For instance, if

$$s = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

then

$$s(1) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}; \quad s(1) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix};$$

More generally $s_{(i_1, \ldots, i_m)}$ means a sub-vector of $s$ which corresponds to $E_{i_1} = \ldots = E_{i_m} = 1$; $E_{j_1} = \ldots = E_{j_n} = 0$.

Notice that in the vector $t_{(i_1, \ldots, i_m)}^{(j_1, \ldots, j_n)}$ the entries $i_1, \ldots, i_m$ equal 1, entries $j_1, \ldots, j_n$ equal 0, all other entries equal $-1$, eg

$$t_{(1,2)}^{(3,4)} = \{1 \ 1 \ 0 \ 0 \ -1 \ \ldots \ -1\}$$

2.6.1 Operation conflict

If two state vectors are provided for the same set of events, this leads a logical conflict: states allowed by one vector may not be allowed by another and vice versa. The conflict is resolved by selecting a set of states contained in both state vectors, ie, by the intersection of the state vectors. We refer to this operation as conflict of state vectors, and designate it as

$$s = r \cap q.$$ 

Formally, to calculate the conflict of two state vectors, we need to expand both of them (ie, get rid of -1 entries), and then select only those rows, which are present in both vectors.

Example 1.

$$r = \{0 \ -1\} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix};$$

$$q = \{-1 \ 0\} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix};$$

$$r \cap q = \{0 \ 0\};$$
Example 2.

\[
\begin{align*}
 r &= \begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \end{pmatrix} ; \\
 q &= \begin{pmatrix} 0 & 0 & -1 \\ -1 & 1 & -1 \end{pmatrix} \\
 r \cap q &= \begin{pmatrix} 0 & 1 & -1 \end{pmatrix} ;
\end{align*}
\]

Note that in the latter example the 3rd column is trivial in both vectors, and it can be ignored when calculating the conflict. In the resulting state vector the 3rd column is also trivial.

We say that the state vector \( r \) is conflicted by the state vector \( q \), if \( r \) is assigned the result of the conflict:

\[
r := r \cap q .
\]  \hspace{1cm} (2.9)

Because the operation conflict corresponds to the logical conjunction, we refer to this class of the state vectors as *conjunction* state vector, or shorter *c-vector*. Later we will introduce two more classes of S-vectors: *e-vector* and *q-vector*. These two types of S-vectors have a different rules of passing their information to the system state vector.

**3 Propositional Logic (PL)**

As was mentioned before, in the propositional logic the system state vector represents the truth table of the system of events. Here we formalise the way how the system S-vector can be calculated in an algorithmic way.

**3.1 Conjunction state vector life cycle**

In the propositional logical system we have a finite number of relations represented as state vectors. Our goal is to find a system state vector, which contains the entire information from all relations. Such a state vector, like a truth table, allows to make a conclusion about the truth value of any event.

As we know the information can be merged by means of the operation conflict. Hence the ultimate system state vector results as a conflict of all state vectors representing relations.

As we know the information can be merged by means of the operation conflict. Hence the ultimate system state vector results as a conflict of all state vectors representing relations.

Practically we could proceed as follows. At the beginning of the logical inference process we introduce a trivial system state vector. As we remember, it contains no information. Next the system state vector has to be conflicted one by one by the S-vectors corresponding to the logical relations. Each time the system state vector is conflicted by a S-vector, it absorbs its information, and thus the S-vector can be removed. The order in which the information is passed to the system state vector is irrelevant (it might be relevant from the performance point of view though).
3.2 State matrix

During the process of theorem proving any new information is added to the system in the form of state vectors. The system state vector must be conflicted by the new state vectors in order to absorb their information. From numerical efficiency perspective it is however not always reasonable to evaluate the conflict between the state vectors immediately. Sometimes instead of calculating the conflict of the state vectors straight away, we put the vectors in a table

$$ S = \begin{bmatrix} \{s\} \\ \{q\} \end{bmatrix} $$

where we used brackets [ ] with the rule that all S-vectors inside must be conflicted. We refer to such a collection of state vectors as state matrix or shorter S-matrix.

The context of the state matrix changes during the life cycle of the S-vectors. At the beginning of the logical inference process the state matrix contains a set of S-vectors representing the logical relations. All S-vectors in the matrix are equivalent, but practically it is convenient to select one of the state vectors in the matrix as the main state vector. The main S-vector is going to represent the system state vector, which accumulates the entire information about the system. The main S-vector should be conflicted by other state vectors to absorb their information. Any S-vector which has passed its information to the main S-vector, can be removed from the S-matrix.

In the process of logical inference a new information may become available, and new state vectors added to the state matrix.

The ultimate system state vector is represented by the main state vector after it is conflicted by all vectors in the S-matrix.

3.3 Example proof of an inference rule

An inference rule in propositional logic can be verified by the truth table. Thus using a state vector algebra we can demonstrate how any inference rule can be automatically verified.

As an example we consider the “Importation” argument form

$$(p \rightarrow (q \rightarrow r)) \vdash ((p \land q) \rightarrow r)$$

We introduce the following 7 events (of which events 4 through 7 are supplementary):

$$ E_1 \equiv p; \quad E_2 \equiv q; \quad E_3 \equiv r; \quad (3.1) $$
$$ E_4 \equiv (q \rightarrow r) \equiv (E_2 \rightarrow E_3); \quad (3.2) $$
$$ E_5 \equiv (p \rightarrow E_4) \equiv (E_1 \rightarrow E_4); \quad (3.3) $$
$$ E_6 \equiv (p \land q) \equiv (E_1 \land E_2); \quad (3.4) $$
$$ E_7 \equiv (E_6 \rightarrow r) \equiv (E_6 \rightarrow E_3); \quad (3.5) $$
We need to show that if $E_5$ is true (premise), then $E_7$ is also true (conclusion). The state vectors corresponding to the relations (3.2 - 3.5) are

(3.2) : $s_4 = \begin{bmatrix} -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & 1 & -1 & -1 & -1 \end{bmatrix}$

(3.3) : $s_5 = \begin{bmatrix} 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 0 & -1 & -1 & -1 & 1 & -1 & -1 \end{bmatrix}$

(3.4) : $s_6 = \begin{bmatrix} 1 & 0 & -1 & -1 & -1 & 0 & -1 \\ 0 & -1 & -1 & -1 & -1 & 0 & -1 \end{bmatrix}$

(3.5) : $s_7 = \begin{bmatrix} -1 & -1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & 0 & -1 & -1 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & 0 & 1 \end{bmatrix}$

Here the pivot columns are highlighted in red. One more state vector is required to express that the premise ($E_5$) is true. Obviously it is

$s_p = \{-1 \ -1 \ -1 \ -1 \ 1 \ -1 \ -1\} \tag{3.10}$

The state matrix now consists of the 5 state vectors:

$S = \begin{bmatrix} \{s_p\} \\ \{s_4\} \\ \{s_5\} \\ \{s_6\} \\ \{s_7\} \end{bmatrix} \tag{3.11}$

After conflicting all state vectors, the resulting state vector becomes:

$s_{\text{term}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ -1 & 0 & -1 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{3.12}$

The terminal state vector contains only values 1 in the 7th column, i.e., $E_7$ is identically true. This implies that from the assumptions of the argument form the conclusion follows.

### 3.4 Summary propositional logic

There is still nothing interesting provided by the state vectors. The calculation of the truth table is rather straightforward task, if the number of events is reasonably small. The advantages of the state vectors approach becomes significant if the number of events becomes bigger. Here are the most important advantages:
• Usage of “unknown” event value -1 makes the representation of logical formulas very compact. Every logical connective involves up to 3 events one of which is a boolean function of the two others, and hence the corresponding state vector contains no more than 4 rows. The entire logical formula is then represented as a set of compact state vectors.

• The evaluation of the final system state vector results as a conflict between all state vectors in the system S-Matrix. The actual order in which the conflict is evaluated is completely irrelevant. It means, that the information contained in every logical sub-formula can be added in arbitrary order, and there is no need for a sequential “derivation” of intermediate formulas.

• There are many other numerical tricks which can improve performance of the S-vector calculation. For instance the following:
  
  – The calculation of a particular event value does not require a complete evaluation of the final system S-vector. Usually it can be done in a more efficient way.
  
  – When evaluating conflict of two S-vectors, only pivot columns have to be taken into account.
  
  – From the performance perspective the order in which the S-vectors are conflicted is important. We say that the state vectors are decoupled if they have all different pivot columns. A degree of coupling between S-vectors is measured as a relative number of common pivot columns. During the operation conflict the pivot columns of each of the conflicted vectors become pivot in the resulting vector. Thus for the computational efficiency reason, strongly coupled S-vectors should be conflicted before weakly coupled or decoupled vectors, in order to minimise the number of pivot columns of the main state vector at every stage.

  – If all state vectors can be split into completely decoupled groups, their truth tables can be calculated independently.

  – Supplementary events can be discarded (and the corresponding columns removed) after all relevant S-vectors are conflicted. The relevant S-vectors are those for which the supplementary event is a pivot column. We do not have to remember which events are supplementary. Simply if some column is not a pivot column in any of the S-vectors (apart from the main system S-vector), it can be removed.

4 Formal S-algebra

As has been already mentioned, the space of S-vectors is equipped with a number of natural operations: union, subtraction, intersection, etc. In this section only we
adopt the following notation (below $a$ and $b$ are some state vectors):

\begin{align*}
  a + b & \quad \text{union} \\  a - b & \quad \text{subtraction} \\  a \cdot b & \quad \text{intersection}
\end{align*}

(We sometimes omit the dot in the intersection.) We also introduce two constant S-vectors: 1 represents the trivial S-vector, and 0 represents empty S-vector.

By definition a state vector should not contain duplicated entries, ie identical states. Any introduced constant S-vector by definition has no duplicates. However the expressions with state vectors may have ambiguous interpretation due to the no-duplicates requirement. For instance, the meaning of the expression $a + b - b$ depends on where we put the brackets:

\[
 a + (b - b) = a \\
(a + b) - b = a - b
\]

In the latter example the union $a + b$ considered as a sum of the states contained in the vectors $a$ and $b$ would have double count of the states belonging to both S-vectors. The duplicates should be removed in order to obey the no-duplicates requirement. Due to the “no duplicates” requirement, the interpretation and rules of transformation of expressions with S-vectors become rather unusual.

The rules of algebraic transformations could be much simpler and unambiguous if we allowed the state duplicates (or better to say – an arbitrary multiplicity). Below we introduce the S-algebra with duplicates. Later we will need to define a rule how to interpret the state vectors with multiple entries.

For the S-algebra with duplicates we consider the state vectors as vectors in multi-dimensional linear space spanned by different states. For the system of $N$ events there are $2^N$ different states. Thus a S-vector can be considered as a vector in $2^N$-dimensional space, having coordinates 1 for those states which are present in the S-vector, and 0 otherwise.

The S-algebra becomes an abstract commutative algebra over a field of integer numbers $\mathbb{Z}$. By other words, we allow linear combinations

\[
 m a + n b
\]

where $a$ and $b$ are S-vectors, and $m$ and $n$ are integer numbers (positive, negative or zero). All operations – union, subtraction and intersection become \textit{element-wise} operations of addition, subtraction and multiplication. All rules of a simple vector algebra apply, eg, commutativity, associativity, bilinearity, etc.

Because the multiplication is element-wise, any vector which has zero coordinates has no inverse. Thus S-algebra is not equipped with inverse elements with respect to the multiplication.

Next we point out that not only the S-vectors themselves, but every state within S-vector can have an individual integer multiplier. For instance, in the union $a + b$ those states which belong to the intersection between $a$ and $b$ are counted twice.
A state vector with duplicates can be converted to the interpretable S-vector without duplicates. We designate an operation of projection onto the space without duplicates by embracing a state vector or an expression in angular brackets, e.g., \([a + b]\).

The projection operator \([\ ]\) works by a simple rule: for every S-vector \(a\) without duplicates

\[
[a] = \begin{cases} 
  a, & n \geq 1 \\
  0, & n < 1 
\end{cases}
\]

The same rule applies to individual states within S-vector: if a state has multiplier greater or equal 1, then it becomes 1, and 0 otherwise.

Now we can formulate a few rules of transformations. Let \(a, b, c\) be some state vectors without duplicates, i.e.

\[
[a] = a; \quad [b] = b; \quad [c] = c;
\]

Then obviously

\[
\begin{align*}
[1 \cdot a] &= a \quad (4.4) \\
[1 + a] &= 1 \quad (4.5) \\
[0 \cdot a] &= 0 \quad (4.6) \\
[0 + a] &= a \quad (4.7) \\
[0 \cdot 1] &= 0 \quad (4.8) \\
[a \cdot a] &= a \quad (4.9) \\
[a + b] &= a + b - ab \quad (4.10) \\
[a - b] &= a - ab \quad (4.11)
\end{align*}
\]

Based on these rules, one can derive transformation rules for more complex expressions.

We can demonstrate the S-vector manipulation technique on example of the information S-vector. We remind the definition: if \(a\) is a S-vector, then 

\[
\bar{a} = 1 - a
\]

is called an information S-vector corresponding to \(a\). If \(a\) contains “allowed” states, then \(\bar{a}\) contains those states which are “prohibited” by \(a\). The operation conflict merges the information from two state vectors. Indeed, let

\[
a = b c
\]

Then obviously

\[
\begin{align*}
a &= (1 - \bar{b}) (1 - \bar{c}) = 1 - \bar{b} - \bar{c} + \bar{b} \bar{c} \quad (4.13) \\
\bar{a} &= 1 - a = \bar{b} + \bar{c} - \bar{b} \bar{c} = [\bar{b} + \bar{c}] \quad (4.14)
\end{align*}
\]

Thus the conflict of two S-vectors is equivalent to the union of the corresponding information vectors, which can be interpreted in terms of information: the information is merged by the operation conflict.
5 First and Higher Order Logic

The first and higher order logic have much richer syntax, and allow to express much more complex statements, than the propositional logic. In this chapter we show how the S-algebra formalism can be extended to work with the higher order logic sentences. The developed theory is applicable to 1st, 2nd and largely to the higher order logic. Below we write HOL to denote the 1st, 2nd and partially higher order logic.

HOL operates with terms \{x, y, z, \ldots\}. Events emerge as predicates evaluated on terms, like \(P(x)\) (unary predicate) or \(Q(x, y)\) (binary predicate), etc. If \(D\) is a definition domain, then predicate of arity \(n\) maps \(D^n \rightarrow B\), where \(B = \{1, 0\}\).

The main feature which makes HOL different from the propositional logic is a use of quantifiers \(\forall\) and \(\exists\). To express the statement \(\forall x P(x)\) in the propositional logic we would have to define a different statement for every point \(x\) from the domain (which would be impossible for an infinite domain).

In order to prove statements of HOL, we are going to use state vector formalism. However its machinery needs to be extended to deal with the quantifiers. We will introduce the so-called e-vectors to express the quantifier \(\forall\). The quantified \(\exists\) is expressed by means of the so-called q-vector (the existential quantifier is beyond the scope of this paper though).

As mentioned before, we use a set-theoretical terminology to describe the logical system, using standard mapping: a statement \(P(x)\) is replaced by an equivalent statement \(x \in A_P\), where \(A_P\) is a set of points for which \(P(x)\) is true. The set-theoretical representation is flexible enough to describe HOL.

Below we use small letters \(x, y, z\) to designate points (terms), big letters \(A, B, C, D\) to designate sets, letters \(P, Q, R\) as predicate symbols. Generally the points do not have to have the same definition domain. In order to make the theory more flexible, we may allow to have different spaces under the same framework. We also allow objects like sets of sets (SoS) in order to cover expressions of the HOL like \(\forall A\) and \(\exists A\), where a set \(A\) is considered as an element of a SoS. The S-algebra is formulated in such a way that it is independent of the concrete implementation of the logic itself, and for this reason we do not focus on it any longer.

5.1 Quantifier \(\forall\)

Unlike events involving constant terms, the events with quantified variables (below: quantified events) are strictly speaking not events in a usual PL sense, until some concrete constant term is substituted in place of the quantified variable. Below we use the following terminology: we say that the formula is applied to a term \(a\), if the point \(a\) is substituted in place of the quantified variable \(x\), and the corresponding event \(\boxed{a \in A_P}\) is added to the system.

An interpretation of the universal quantifier will require a few new definitions.
First we introduce a virtual point. The purpose of virtual point is to be a representative for an arbitrary point. We will give a formal definition, and then explain how it can be used in quantified expressions. For the definition of a virtual point it is essential that we use a set-theoretical representation.

**Definition: Virtual point.**

Let $D$ be some non-empty set. We consider a collection of sets $A_1, \ldots, A_n$, all being subsets of $D$

$$A_i \subseteq D, \quad i = 1, \ldots, n.$$  

(5.1)

Let the point $v$ be an element of $D$

$$v \in D = 1$$  

(5.2)

We say that the point $v$ is virtual for the collection of sets $\{A_i\}$ if no other information is provided about the point $v$. This means that the values of events

$$E_i = v \in A_i, \quad i = 1, \ldots, n$$

are all indefinite, with two exceptions:

$$E_i = 0, \quad \text{iff} \quad A_i = \emptyset$$  

(5.3)

$$E_i = 1, \quad \text{iff} \quad A_i = D.$$  

(5.4)

The set $D$ is called definition domain of the virtual point $v$.

Note that if $v$ is virtual for some collection of sets $\{A_i\}$, it may not be virtual for some other collection of sets $\{B_j\}$, since the condition (5.1) may not hold for all $B_j$.

The definition of a virtual point allows to assert the events which use the universal quantifier, since whatever is true for the virtual point, is also true for any other point. More precisely, if some state becomes prohibited for the virtual point, it will also be prohibited for any other point from the same domain.

Next important concept is an information related to the quantified event. We introduce here a new definition.

**Definition: Informative event.**

Let us consider a system of events $\{E_1, \ldots, E_N\}$. Let also $s$ be the system state vector. Suppose that some event $E_i$ takes both values – true and false. Let us find a sub-vector $s_{sub} = s^{(i)}$ which contains all states of the original $S$-vector corresponding to the event value $E_i = 1$. We say that the value $E_i = 1$ is informative if the sub-vector $s_{sub}$ constrained to the subset $\{E_k\}_{k \neq i}$ has smaller size than the original $S$-vector $s$ constrained to the same subset $\{E_k\}_{k \neq i}$. Similarly we define when the value $E_i = 0$ is informative.

If the value $E_i$ has only one value – true or false, it is by definition not informative.
The meaning of this definition is obvious – if an event value true (or false) is informative, then the values of other events would be affected if the informative event became true (or false). In the following example

\[ s = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix}; \quad s^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}; \quad s_{(1)} = \begin{bmatrix} 0 & 0 \end{bmatrix} \]  \tag{5.5}

the event value \( E_1 = 1 \) is not informative, since the original vector \( s \) and the sub-vector \( s^{(1)} \) constrained to the event \( E_2 \) are both \( \{-1\} \). The event value \( E_1 = 0 \) is informative, since the constrained sub-vector \( s_{(1)} \) becomes \( \{0\} \) and hence contains fewer allowed states than the original vector \( \{-1\} \). Similarly, the event value \( E_2 = 1 \) is informative, and \( E_2 = 0 \) is not informative.

**Note:** If the event value \( E_i = 1 \) is non-informative, then there exists such a reduced form of the state vector which contains in the \( i \)th column only values 1 or \(-1\). Indeed, for any state with \( E_i = 0 \) there must exist an identical state with \( E_i = 1 \), otherwise the event value \( E_i = 1 \) would be informative. The opposite statement is also true: if there exists a reduced form of the state vector, which contains only values 1 and \(-1\) in the \( i \)th column, then the event value \( E_i = 1 \) is non-informative.

Let us consider a few examples where the quantified event \( P \) is informative or not.

1. **Solitary statement**

   \[ P \]  \tag{5.6}

   The statement itself is supposed to be true by definition. Thus the event value \( 1 \) is not informative.

2. **Consequent statement in a material implication**

   \[ Q \rightarrow P \]  \tag{5.7}

   For the events \( E_1 \equiv Q, E_2 \equiv P \) we have the following state vector

   \[ s = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \]

   The event value \( E_2 = 1 \) is not informative.

3. **Consequent statement in a material implication which is a part of a compound formula**

   \[ R \equiv (Q \rightarrow P) \]  \tag{5.8}

   For the events \( E_1 \equiv R, E_2 \equiv Q, E_3 \equiv P \) we have the following state vector

   \[ s = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \]
The event value $E_3 = 1$ is informative. Indeed, the S-vector constrained to \{\(E_1, E_2\)\} is

$$q = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The sub-vector $s^{(3)}$ constrained to \{\(E_1, E_2\)\} is

$$r = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Obviously $r$ is smaller than $q$, and hence $E_3 = 1$ is informative.

4. Antecedent statement in a material implication

$$P \rightarrow Q \quad (5.9)$$

For the events $E_1 \equiv P$, $E_2 \equiv Q$ we have the following state vector

$$s = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}$$

The event value $E_1 = 1$ is informative in this case.

5. Either side in a biconditional relation

$$Q \leftrightarrow P \quad (5.10)$$

For the events $E_1 \equiv Q$, $E_2 \equiv P$ we have the following state vector

$$s = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$$

The event value $E_2 = 1$ is informative in this case too.

As we remember, the information conveyed by an S-vector is contained in those states, which are prohibited by the S-vector. Consider some S-vector $s_1$. The prohibited states comprise the information state vector

$$\bar{s}_1 = 1 - s_1$$

Let us consider some other state vector $s_2$, which prohibits another set of states

$$\bar{s}_2 = 1 - s_2$$

If the states prohibited by $s_2$ are a subset of the states prohibited by $s_1$

$$\bar{s}_2 \subset \bar{s}_1$$

then the S-vector $s_2$ contains some subset of the information conveyed by $s_1$. 
We say that an S-vector is *legal* if it contains correct information describing relations between system events. A legal S-vector prohibits only those states which are inconsistent with the system relations. Only legal S-vectors can be used for the logical inference.

We conclude that if in the example above $s_1$ is a legal S-vector, so is $s_2$. It contains a correct information about the system, though it is less informative than the vector $s_1$. Thus the S-vector $s_2$ can be legally added to the system S-matrix.

Next we give a definition of a transformation of an S-vector, which turns an informative event into a non-informative by reducing the number of prohibited states.

**Definition: Z-transformation.**

*Let us consider a state vector $s$. We introduce the following transformation procedure: all values “0” in the $n$th column are replaced by “-1”. We refer to this transformation as “Zero-transformation” for the $n$th event, or simply $Z_n$-transformation. For instance for the relation $E_1 ⇔ (E_2 \rightarrow E_3)$ the state vector

$$s = \begin{pmatrix}
1 & 1 & 1 \\
1 & 0 & -1 \\
0 & 1 & 0
\end{pmatrix}$$

the state vector

after $Z_3$-transformation becomes

$$s[^3] = \begin{pmatrix}
1 & 1 & 1 \\
1 & 0 & -1 \\
0 & 1 & -1
\end{pmatrix}$$

We use the upper index with the number of the modified column to designate the transformed state vector.*

Obviously, if $s$ is a legal S-vector, then the transformed S-vector $s[^n]$ is also a legal S-vector, which contains less information than the original vector.

The Z-transformed S-vector has the following properties

1. After $Z_n$-transformation the event value $E_n = 1$ becomes non-informative (see the note on p. [17]).

2. If the event value $E_n = 1$ is non-informative, then $Z_n$-transformation has no effect on the S-vector (the state vector is invariant under the $Z_n$-transformation).

3. The event $E_n$ can not be proven false based on the information of the modified S-vector. Indeed, no matter what the values of other events are, the value of $E_n$ can only be 1 or $-1$, ie, in any interpretation the event $E_n$ is either true or indefinite.
The reason why we need the Z-transformation can be best explained on the following example. Let us consider the relation

\[ Q \leftrightarrow \forall x P(x) \]

applied to some constant point \( a \). Designating \( E_1 \equiv Q, E_2 \equiv [a \in A_P] \) we formally get a relation

\[ E_1 \leftrightarrow E_2 \]

The formal propositional S-vector corresponding to the latter relation is

\[ s = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \]

The \( Z_2 \)-transformation gives

\[ s^{[2]} = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \]

Instead of using the full S-vector \( s \), we have to add the modified vector \( s^{[2]} \) to the S-matrix. Let’s show why. First we observe that if \( Q \) is true, then for any point \( x \) the statement \( P(x) \) holds. In particular, \( P(a) \) is true. So the state \( \{1 \ 1\} \) is allowed. Next if \( Q \) is false, then we conclude that “not for all \( x \) the statement \( P(x) \) is true”

\[ \neg \forall x P(x) \]

However we do not know if \( P(a) \) is true for some fixed point \( a \), and hence the original state vector would give a wrong information that \( P(a) \) is false. The modified S-vector contains much less information, in particular the row \( \{0 \ -1\} \) is equivalent to saying “if \( Q \) is false, then \( P(a) \) can be anything”. So the modified vector conveys a correct logical meaning. This is a consequence of the feature of the modified S-vector, that the transformed event can not be proven false.

If we consider the logical inference in the opposite direction – from \( P \) to \( Q \), we also notice an inconsistency, if we try to use the original S-vector. Indeed, if \( P(a) \) is false, then \( Q \) is false too, whatever point \( a \) is (one counter-example is sufficient to prove the quantified event wrong). Thus the state \( \{0 \ 0\} \) is allowed. However if \( P(a) \) is true, from the original S-vector we would make a wrong conclusion that \( Q \) is true. It would be wrong, since in order to prove \( Q \) true, we need to show that \( P(x) \) holds for all points from the domain. If we use the modified S-vector instead, then no conclusion can be made about \( Q \) if \( P(a) \) is true (this is a feature of the transformed S-vector that the quantified event becomes non-informative). Thus the modified S-vector conveys a consistent logical information in this case too.

If the quantified event is informative, then we have to apply a Z-transformation to the state vector before adding it to the state matrix. The modified S-vector conveys a correct logical information, but leads to a loss of information. In particular, with the modified S-vector we can not express two following statements:

1. if \( \forall x P(x) \) is true then \( Q \) is true
2. if $Q$ is false, then “not for all $x P(x)$ is true”

Below we show how the first statement can be expressed with the so-called e-vector. The second statement can be expressed with the so-called q-vector (we won’t be considering q-vectors in this paper).

If the quantified event is non-informative, then no Z-transformation is required, since it would have no effect on the original S-vector (property 2 at p. [13]). There is no loss of information in this case.

5.2 Rules for quantified event

The interpretation of the quantified event depends on whether the event value 1 is informative or not. First we consider the case when the quantified event value 1 is informative.

5.2.1 Informative case

An informative event can be applied to an arbitrary constant term, and to a virtual point. The corresponding state vector has to be modified by the Z-transformation in order to provide a correct logical information. However, as was mentioned before, in the informative case the Z-transformation leads to the loss of information.

In this section we consider how the statements like

if ∀$x P(x)$ is true, then $Q$ is true

can be expressed by means of a virtual point and a new type of the state vector, which we refer to as “e-vector”.

First we notice that in order to express this information, we have to use a virtual point. The value of the event $P(a)$ for an arbitrary constant point $a$ is not representative, since what is true for the point $a$ is not necessary true for another point $b$. The virtual point serves as a representative for an arbitrary point.

Before formulating the formal rules, we demonstrate the idea on a simple example. Let us consider the following statement

\[ \forall x \ P(x) \rightarrow Q \]  \hspace{1cm} (5.11)

Let $v$ be a virtual point for the space $A_P$. We apply the statement to the virtual point. Designating $E_1 \equiv_v v \in A_P \ E_2 \equiv Q$ we write the latter relation as

\[ E_1 \rightarrow E_2 \]  \hspace{1cm} (5.12)

First we write the propositional S-vector for this logical connective:

\[ s = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix} \]  \hspace{1cm} (5.13)
After $Z_1$-transformation it becomes

$$s^{[4]} = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix} = \begin{pmatrix} -1 & -1 \end{pmatrix}$$  \hspace{1cm} (5.14)$$

In this example the modified vector bears no information, and can be dropped. However in more general case the modified state vector should be added to the system matrix.

Now we want to express the information contained in the original statement: if for every point $x$ the statement $P(x)$ holds, then statement $Q$ holds. How can we verify that the statement $P(x)$ holds for every point by observing the event $P(v)$? According to the definition of the virtual point, if the value of the event $P(v)$ is identically true (i.e., if all values 0 vanish from the relevant column of the system S-vector), then the space $A_P$ coincides with the domain $D$, and hence any point from that domain is an element of $A_P$, which is equivalent to saying that $\forall x P(x)$ is true. For the considered example this can be expressed as follows: if the system state vector does not tolerate the state vector $\begin{pmatrix} 0 & -1 \end{pmatrix}$, then this implies that the statement $P(x)$ holds for any point $x$, and hence we can use the original unmodified state vector $\begin{pmatrix} 1 & 1 \end{pmatrix}$ together with the knowledge that the event $E_1$ is always true, i.e., we only need the sub-vector corresponding to $E_1 = 1$.

$\begin{pmatrix} 0 & -1 \end{pmatrix}$

We introduce a new

**Definition:** *Exception state vector.*

Let there be a quantified event applied to some virtual point $v$, and $n$ be the number of the event. Let also $s$ be the state vector generated for the logical expression according to the rules of the propositional logic.

We introduce a new type of state vector, which we refer to as exception-vector, or simply e-vector.

E-vector consists of two parts: trigger $T$ and terminator $K$.

Trigger $T$ of the e-vector is defined as a subset of $s$ corresponding to $E_{n} = 0$

$$T = s_{(n)}$$  \hspace{1cm} (5.15)$$

Terminator $K$ of the e-vector is defined as a subset of $s$ corresponding to $E_{n} = 1$

$$K = s^{(n)}$$  \hspace{1cm} (5.16)$$

\footnote{We say that two state vectors do not tolerate each other if their conflict vanishes.}
We designate the e-vector as $s^{<n>}$ and write it as

$$s^{<n>} = \left( \frac{T}{K} \right) = \left( \frac{s_{(n)}}{s^{(n)}} \right)$$

The e-vector is applied to the system state vector by the following rule. The main system state vector is tested for the exception by conflicting with the trigger. If the conflict vanishes (ie, if the main state vector does not tolerate the trigger), then we say that the exception is triggered. In this case the terminator has to be added to the system S-matrix as a regular conjunction S-vector.

Strictly speaking, the trigger could in this case be defined simply as $t_{(n)}$ (which would also be practically more efficient), but the definition above is more generic, and as we will see, is also applicable to the case of multiple quantified events.

For example, let us consider the relation

$$E_1 \equiv (E_2 \rightarrow E_3)$$

where $E_3$ is a quantified event applied to a virtual point. The propositional S-vector is given by

$$s = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$Z_3$-transformed vector is given by

$$s^{[3]} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

The corresponding e-vector becomes

$$s^{<3>} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & -1 & 1 \end{pmatrix}$$

Both vectors – $s^{[3]}$ and $s^{<3>}$ should be added to the S-matrix.

5.2.2 E-vector Life circle

E-vector has slightly different life cycle than a c-vector. As was mentioned above, any S-vector which contains some information about the events, should be added to the system S-matrix. As soon as the main system S-vector is conflicted by a c-vector, the latter can be removed, as the conflict passes all the information to the main S-vector.
E-vector, whenever added to the system state matrix, can pass the information to the main S-vector by the trigger/terminator rule described above: if the trigger is not tolerated (ie, when the exception is triggered), then the terminator should be added to the S-matrix as an ordinary c-vector. After that the e-vector can be removed from the system S-matrix. However if the exception has not been triggered, it still may be triggered at some later stage, as more information is added to the system. Thus, the e-vector, which does not trigger the exception, has to stay in the system S-matrix. The e-vector has to be tested for exception every time a new information is added to the system.

Now let us formulate the rules for the informative quantified event:

- The informative quantified event can be applied to any constant term, or to a virtual point.
- Let $s$ be the state vector which corresponds to the logical relation in propositional logic. The state vector $s[i]$ obtained from $s$ by the $Z_i$-transformation, should be added to the system (here $i$ is the number of the quantified event). The modified S-vector is treated as an ordinary conjunction S-vector.
- If applied to a virtual point, an exception state vector $s^{<i>}$ should be added to the system too. The e-vector application and life cycle are described above.
- A virtual point $v$ is suitable to be substituted for the quantified variable $x$, if both $v$ and $x$ have the same definition domain. If there are no suitable virtual points in the system, a new virtual point has to be created.

Notice an asymmetry between the ways we interpret values “0” and “1” of the quantified event $\forall x P(x)$. The reason for that is in the logic of the quantified expression itself. In order to prove quantified statement true, one has to prove it for all possible values of the variable $x$. In order to prove it wrong, it is sufficient to find one counter-example.

One important note should be made about the virtual points. A remarkable feature of the virtual point is that we only need to introduce very few virtual point for every domain. Namely, for each domain we need no more virtual points then the maximal number of quantified variables used in any of the logical sentences. This makes the practical usage of the approach very efficient.

5.2.3 Non-informative case

If the quantified event is non-informative, then the corresponding propositional state vector conveys a correct logical information, and does not require $Z$-transformation. For this reason the application of the quantified event in the non-informative case is trivially simple – it can be applied to any term, and interpreted as an ordinary propositional formula.
• The formula with universal quantifier can be applied to any constant term in the system, or to any virtual point.

• A conjunction state vector for the formula has to be generated according to the rules of propositional logic. In the non-informative case the state vector does not need to undergo the Z-transformation.

• No e-vector needs to be generated, as it would convey no information.

Example 1  Consider case 1 from above – solitary statement

\[ \forall x \; P(x) \]  

which in set-theoretical representation reads

\[ \forall x \; x \in A \]  

For every point \( a \) (virtual or not), we create an event

\[ E_a \equiv a \in A \]  

Each such event is added to the system. The state vector corresponding to the event is simply \( \{-1 \ldots 1 \ldots -1\} \), where 1 stands in the place of the event \( E_a \).

Example 2  Consider case 2 from above – consequent statement in a material implication

\[ Q \rightarrow \forall x \; P(x) \]  

For every point \( a \) (virtual or not) we create an event \( E_a \equiv a \in A \)  

Defining \( E_Q \equiv Q \) we get

\[ E_Q \rightarrow E_a \]  

The corresponding state vector becomes

\[ s = \begin{cases} 1 & 1 \\ 0 & -1 \end{cases} \]  

Notice that Z-transformation for the second event would not have any impact on the state vector.

5.2.4 Summary for universal quantifier

Practically to interpret the quantified event, one could proceed as follows.

• Apply the quantified event to any constant term (provided its domain coincides or is a subset of the domain of the quantified variable), or to a suitable a virtual point.
• Generate a state vector according to the propositional logic rules.

• Apply Z-transformation to the state vector. If transformed vector coincides with the original one, then the quantified event is non-informative. Otherwise it is informative.

• Add the modified S-vector to the system S-matrix.

• If the event turns out to be informative, and if the point is virtual – generate an e-vector and add it to the system S-matrix.

• If the event turns out to be informative, make sure to apply it to the virtual point. Create one, if there is no suitable virtual point available.

5.3 Long sentences and multiple quantified events

Let us consider a compound logical sentence, which contains more than one logical connective

\[ E_1 \equiv (E_2 \otimes (E_3 \otimes (\ldots))) \] (5.20)

where in place of each symbol \( \otimes \) can be an arbitrary logical connective. Some of the events \( E_i \) can be quantified, and some of the quantified events may use different quantified variables.

The rules of S-algebra can be generalised for an arbitrary number of quantified events with arbitrary number of quantified variables. This is however beyond the scope of this paper, as we do not have a purpose to present the complete theory of S-algebra, but rather to demonstrate the concept of S-algebraic approach. In this paper we only consider the simplest case of two adjacent quantified events using the same quantified variable.

Compound logical sentences, containing more than one logical connective (in which case it also may contain more than one quantified event), should be decomposed into elementary sentences with the help of supplementary events. Each of the elementary sentences has a structure

\[ E_s \equiv (E_2 \otimes E_3) \]

where \( E_s \) is the supplementary event. The supplementary event is not a quantified event, regardless of whether any or both events \( E_2 \) and \( E_3 \) are quantified. After calculating the state vectors representing each of the elementary sentences, the quantified events can be removed as was described before.

If only one of the two events \( E_2 \) and \( E_3 \) is quantified, the rules described in the previous sections apply.

In this section we consider a case of two quantified events. Consider the relation \( E_s \equiv (E_2 \otimes E_3) \) in which \( E_2 \) and \( E_3 \) are quantified events applied to a virtual point. Let \( s \) be the propositional state vector for the relation . Then
• The Z-transformed vector is constructed by substituting -1 in place of the event values $E_2$ and $E_3$ for all rows of $s$ with $E_s = 0$.

• The e-vector consists of Trigger $T$ and Terminator $K$. The trigger consists of rows of $s$ corresponding to the event value $E_s = 0$. The Terminator consists of the rows of $s$ corresponding to the event value $E_s = 1$.

Let for instance consider the following relation

$$R \equiv \forall x \; (P(x) \rightarrow Q(x)) \quad (5.21)$$

Substituting the virtual point $v$ in place of the quantified variable and designating

$$E_1 \equiv R, \quad E_2 \equiv v \in A_P, \quad E_3 = v \in A_Q \quad (5.22)$$

we obtain the formula

$$E_1 \equiv (E_2 \rightarrow E_3) \quad (5.23)$$

where both $E_2$ and $E_3$ are quantified.

The propositional S-vector is given by

$$s = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad (5.24)$$

The Z-transformed S-vector becomes

$$s^{[2,3]} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix} \quad (5.25)$$

The corresponding e-vector follows as

$$s^{<2,3>} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & -1 \end{bmatrix} \quad (5.26)$$

Below we consider a few basic theorems of the set theory and show how they can be used for the logical inference.

### 5.4 Example theorems

#### 5.4.1 Theorem $T_{\subseteq}$

Let us consider the following theorem.

$$(A \subseteq B) \leftrightarrow \forall x \; (x \in A \rightarrow x \in B) \quad (5.27)$$
This theorem emerges as a basic set theory concept. Applying the quantified events to a virtual point \( v \), and designating

\[
E_1 \equiv A \subseteq B; \quad E_2 \equiv v \in A; \quad E_3 \equiv v \in B;
\]  
(5.28)

we get the following propositional sentence

\[
E_1 \leftrightarrow (E_2 \rightarrow E_3)
\]  
(5.29)

We could introduce a supplementary event \( E_s \) such that

\[
E_s \equiv (E_2 \rightarrow E_3); \quad E_1 \leftrightarrow E_s;
\]

However since effectively \( E_s \) and \( E_1 \) are identical, we can omit the supplementary event and interpret \( E_1 \) as if it was supplementary. For the set \( \{E_1, E_2, E_3\} \) we get the state vectors coinciding with the vectors (5.25) 5.26:

\[
s^{[2,3]} = \begin{cases} 
1 & 1 & 1 \\
1 & 0 & -1 \\
0 & -1 & -1
\end{cases}
\]  
(5.30)

\[
s^{\langle 2,3 \rangle} = \begin{pmatrix} 
0 & 1 & 0 \\
1 & 1 & 1 \\
1 & 0 & -1
\end{pmatrix}
\]  
(5.31)

### 5.4.2 Theorem \( T_\equiv \)

Let us consider the following theorem.

\[
(A \equiv B) \leftrightarrow \forall x \ (x \in A \leftrightarrow x \in B)
\]  
(5.32)

Applying the quantified events to a virtual point \( v \), and designating

\[
E_1 \equiv A \equiv B; \quad E_2 \equiv v \in A; \quad E_3 \equiv v \in B;
\]  
(5.33)

we get the following propositional sentence

\[
E_1 \leftrightarrow (E_2 \leftrightarrow E_3)
\]  
(5.34)

Similarly to the previous theorem we get the following state vectors

\[
s^{[2,3]} = \begin{cases} 
1 & 1 & 1 \\
1 & 0 & 0 \\
0 & -1 & -1
\end{cases}
\]  
(5.35)

\[
s^{\langle 2,3 \rangle} = \begin{pmatrix} 
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 1 & 1 \\
1 & 0 & 0
\end{pmatrix}
\]  
(5.36)
5.4.3 Theorem $T_\cup$

Consider the following theorem.

\[ (A = B \cup C) \leftrightarrow \forall x \ (x \in A \leftrightarrow (x \in B \land x \in C)) \]  

(5.37)

Applying the quantified events to a virtual point $v$, and designating

\[ E_1 \equiv A = B \cup C ; \ E_2 \equiv v \in A ; \ E_3 \equiv v \in B ; \ E_4 \equiv v \in C \]  

(5.38)

we get the following propositional sentence

\[ E_1 \leftrightarrow (E_2 \leftrightarrow (E_3 \land E_4)) \]  

(5.39)

The PL S-vector for this formula reads

\[
s = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 \\
\end{bmatrix} \]

(5.40)

We get the following $c$- and $e$-state vectors

\[
s^{[2,3,4]} = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & -1 & -1 & -1 \\
\end{bmatrix} ; \quad s^{<2,3>} = \begin{bmatrix}
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
\end{bmatrix} \]

(5.41)

5.4.4 Theorem $T_\varnothing$

Consider the following theorem

\[ (A = \varnothing) \leftrightarrow \forall x \ (x \in A) \]  

(5.42)

Applying the quantified event to a virtual point $v$ and designating

\[ E_1 \equiv A = \varnothing \quad E_2 \equiv x \in A \]  

(5.43)

we get the relation

\[ E_1 \leftrightarrow \neg E_2 \]  

(5.44)
Because of the logical negation, the roles of true and false values are swapped for this example. The propositional state vector for \( \{E_1, E_2\} \) is given by
\[
s = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}
\] (5.45)
The Z-transformed vector is thus
\[
s[-2] = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}
\] (5.46)
The e-vector is given by
\[
s^{-2} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}
\] (5.47)
here we used index -2 to emphasise that the value of the quantified event is inverted.

5.4.5 Examples of logical inference

In this section we show how the logical inference works with the help of the virtual points. We consider a couple of demonstrative examples.

**Example 1** Let \( A, B \) and \( C \) be some sets. Consider the following relations:

\[
A \subseteq B; \quad B \subseteq C
\] (5.48)

We show how the relation

\[
A \subseteq C
\] (5.49)

can be derived with the help of state vectors.

Let \( v \) be a virtual point for the sets \( A, B, C \). We introduce the following events

\[
E_1 \equiv A \subseteq B \\
E_2 \equiv B \subseteq C \\
E_3 \equiv A \subseteq C \\
E_4 \equiv v \in A \\
E_5 \equiv v \in B \\
E_6 \equiv v \in C
\]

By definition events \( E_1 \) and \( E_2 \) are true.

We apply theorem \( T \subseteq \) to all three events \( A \subseteq B, B \subseteq C \) and \( C \subseteq D \). For the relation

\[
E_1 \iff (E_4 \rightarrow E_5)
\]
we get the following vectors (pivot columns are highlighted):

\[
\begin{align*}
\mathbf{s}_1^{[4,5]} &= \begin{pmatrix} 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 0 & -1 & -1 \\ 0 & -1 & -1 & -1 & -1 & -1 \end{pmatrix} ; \quad (5.50) \\
\mathbf{s}_1^{<4,5>} &= \begin{pmatrix} 0 & -1 & -1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 0 & -1 & -1 \end{pmatrix} ; \quad (5.51)
\end{align*}
\]

For the relation \( E_2 \leftrightarrow (E_5 \rightarrow E_6) \)

we get:

\[
\begin{align*}
\mathbf{s}_2^{[5,6]} &= \begin{pmatrix} -1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 & 0 & -1 \\ -1 & 0 & -1 & -1 & -1 & -1 \end{pmatrix} ; \quad (5.52) \\
\mathbf{s}_2^{<5,6>} &= \begin{pmatrix} -1 & 0 & -1 & -1 & 1 & 0 \\ -1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 & 0 & -1 \end{pmatrix} ; \quad (5.53)
\end{align*}
\]

For the relation \( E_3 \leftrightarrow (E_4 \rightarrow E_6) \)

we get:

\[
\begin{align*}
\mathbf{s}_3^{[4,6]} &= \begin{pmatrix} -1 & -1 & 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 & -1 & -1 \end{pmatrix} ; \quad (5.54) \\
\mathbf{s}_3^{<4,6>} &= \begin{pmatrix} -1 & -1 & 0 & 1 & -1 & 0 \\ -1 & -1 & 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 0 & -1 & -1 \end{pmatrix} ; \quad (5.55)
\end{align*}
\]

The premise of the problem, \( E_1 = 1 \) and \( E_2 = 1 \), can be expressed with the state vector

\[
\mathbf{s}_p = \{1 \ 1 \ -1 \ -1 \ -1 \} \quad (5.56)
\]

Thus the state matrix \( \mathbf{S} \) contains 7 state vectors, of which 4 are c-vectors, and 3 are e-vectors.

\[
\mathbf{S} = \begin{bmatrix}
\mathbf{s}_p & \mathbf{s}_1^{[4,5]} & \mathbf{s}_1^{<4,5>} & \mathbf{s}_2^{[5,6]} & \mathbf{s}_2^{<5,6>} & \mathbf{s}_3^{[4,6]} & \mathbf{s}_3^{<4,6>}
\end{bmatrix}
\]

Let the main state vector \( \mathbf{s} \) coincides at the beginning with \( \mathbf{s}_p \)

\[
\mathbf{s} = \mathbf{s}_p
\]

31
The main state vector does not tolerate the trigger of \(s^{<4,5>}_1\) and \(s^{<5,6>}_2\), and thus both exceptions are triggered, and terminators are added to the state matrix. After conflicting by both terminators, the main state vector becomes

\[
\mathbf{s} = \begin{bmatrix} 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 0 & 0 & -1 \end{bmatrix}
\]

Next we notice that the new main state vector does not tolerate trigger of \(s^{<4,6>}_3\), and hence the terminator of \(s^{<4,6>}_3\) has to be added to the matrix. After conflict, the main vector becomes

\[
\mathbf{s} = \begin{bmatrix} 1 & 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & -1 \end{bmatrix}
\]

The event \(E_3\) is identically true. We thus have proven that \(A \subseteq C\) follows from \((A \subseteq B) \land (B \subseteq C)\).

**Example 2** Consider the following relations:

\[
A \subseteq B; \quad B \subseteq A
\]

We show how the relation

\[
A \equiv B
\]

can be proven with the help of state vectors.

Let \(v\) be a virtual point for the sets \(A, B\). We introduce the following events

\[
\begin{align*}
E_1 & \iff \ A \subseteq B \\
E_2 & \iff \ B \subseteq A \\
E_3 & \iff \ A \equiv B \\
E_4 & \iff \ v \in A \\
E_5 & \iff \ v \in B
\end{align*}
\]

By definition events \(E_1\) and \(E_2\) are true.
As before, we get the following state vectors

\[
\begin{align*}
\mathbf{s}^{[4,5]}_1 &= \begin{pmatrix} 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 0 & -1 \\ 0 & -1 & -1 & -1 & -1 \end{pmatrix} ; \\
\mathbf{s}^{<4,5>}_1 &= \begin{pmatrix} 0 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 0 & -1 \end{pmatrix} ; \\
\mathbf{s}^{[5,4]}_2 &= \begin{pmatrix} -1 & 1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 & 0 \\ -1 & 0 & -1 & -1 & -1 \end{pmatrix} ; \\
\mathbf{s}^{<5,4>}_2 &= \begin{pmatrix} -1 & 0 & -1 & 1 & 0 \\ -1 & 1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 & 0 \end{pmatrix} ; \\
\mathbf{s}^{[4,5]}_3 &= \begin{pmatrix} -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & 0 & -1 & -1 \end{pmatrix} ; \\
\mathbf{s}^{<4,5>}_3 &= \begin{pmatrix} -1 & -1 & 0 & 1 & 0 \\ -1 & -1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 0 & 0 \end{pmatrix} ;
\end{align*}
\]

corresponding to the relations, and the initial state vector

\[
\mathbf{s}_p = \{1 \hspace{0.2cm} 1 \hspace{0.2cm} -1 \hspace{0.2cm} -1 \hspace{0.2cm} -1 \}
\]

representing the initial condition. The main state vector \(s\) initially coincides with \(s_p\)

\[
s = s_p
\]

First two triggers are not tolerated by the main S-vector, and the terminators are added to the state matrix. After conflicting with the main we get

\[
s = \begin{pmatrix} 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 0 & 0 \end{pmatrix}
\]

Now the third exception is triggered, and after the conflict with the terminator, the main state vector becomes

\[
s = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}
\]

which implies that \(A \equiv B\).
5.4.6 Derived theorems

In the considered examples we have shown how to prove some relation between sets using virtual points. The proved relation can be used to derive a new “higher level” theorems. For instance, for some sets $A, B, C$, from the relations

$$( A \subseteq B ) \land ( B \subseteq C )$$

we derived that

$$A \subseteq C$$

In this derivation we used some fixed sets $A, B, C$. However if the sets were not chosen in any special way, or by other words, if we used the quantified sets $\forall A, B, C$, then the derived relation can be applied to any other combination of sets. Thus a new theorem can be formulated as:

$$\forall A, B, C \ ( ( A \subseteq B ) \land ( B \subseteq C ) ) \rightarrow ( A \subseteq C )$$

This is a new theorem which has not been in the system before the derivation. Now it can be applied directly to an arbitrary combination of sets without going through the virtual points formalism. This derivation however is completely in responsibility of the programming language which implements the logical system, since the S-algebra only provides a way of computing the truth values of events.

If the programming language implementing the logical system is equipped with the induction principle, then the derived theorem can be extended with an arbitrary degree of nested applications of the theorem above

$$\forall A_1, \ldots, A_n \ ( A_1 \subseteq A_2 ) \land ( A_2 \subseteq A_3 ) \land \ldots \land ( A_{n-1} \subseteq A_n ) \rightarrow ( A_1 \subseteq A_n ) \quad ( 5.68 )$$

Again, the induction principle has to be a feature of the implementation of the logic.

6 Conclusion

The presented examples of the logical inference serve here only a demonstration purpose. Normally the simple theorems like $( A \subseteq B ) \land ( B \subseteq C ) \vdash ( A \subseteq C )$ are formulated explicitly in a set theory, and thus do not need to be derived. However on the example above we demonstrated how the theorems can be derived from the very basic principles of the set theory. The inference algorithm is very generic and powerful – it can be applied to a theory of arbitrary complexity (which uses the expressive power of HOL), for which there exist no a priori “high level” axioms, and the traditional logical inference approach would require a derivation of a large number of intermediate theorems. In S-algebraic approach the intermediate results do not need to be formulated explicitly. They occur automatically (though implicitly) in the process of algebraic transformations.

This also demonstrates one of the main advantages of the S-algebraic approach: the process of a logical inference does not need any sequential derivation (eg, a resolution
algorithm). The basic set-theoretical axioms can be applied to the logical system in an arbitrary order. The information conveyed in a theorem is passed to the S-matrix in form of S-vectors, regardless whether it leads to any meaningful conclusion right away. It will automatically become meaningful as soon as enough information is added to the S-matrix.

Since in the S-algebraic theorem proving algorithm we do not have the explicit intermediate results, it might be rather difficult to follow the proof, and to translate it into a human language. The ideas of graph theory can be used for the proof analysis, however this topic is beyond the scope of this paper.