# Logic and Mathematical Programming

Sanjoy K. Mitter
(Joint work with V. Borkar, V. Chandru
[both of Indian Institute of Science]
and D. Micciancio [MIT]) *

Department of Electrical Engineering and Computer Science
and
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

## 1 Introduction

A fundamental problem in logic is determining whether a formula is satisfiable, i.e. there exists a valuation for the variables occurring in the formula that makes the whole formula true. Logical deduction can be easily reduced to satisfiability: formula $\phi$ is a logical consequence of a set of formulas $A$ if and only if the set of formulas $A \cup \{\neg\phi\}$ is unsatisfiable. Therefore algorithms to decide the satisfiability of formulas can immediately be turned into procedures for logical deduction and automated reasoning.

In fact, the first serious studies of spatial embeddings of logic [5, 4] have been primarily aimed to transfer methodologies and algorithms from the field of mathematical programming to the symbolic world of computational logic.

A new perspective on these studies is presented in [3]. In [3] the embedding of logic into mathematical programming is used to prove some well known theorems of first order logic. The novelty of this work is not in the results achieved, but in the approach used: the topological structure of the space logical satisfiability is embedded into is exploited to gain structural insights.

We are interested in logic mainly as a language to describe and reason about computer programs. ¿From this point of view, it would be interesting to see to what extent the spatial embeddings studied for propositional logic can be extended to other logic languages, such as dynamic logic [7] and process logic [6].

Finding embeddings of dynamic logic in the style of [3] is presumably a hard problem because of some non-compactness results that affect that logic.

*This research has been supported by U.S. Army Research Office grant DAAL03-92-G-0115 to the Center for Intelligent Control Systems and by a grant from Siemens AG.

Therefore it seems desirable to explore the feasibility of such embeddings by choosing a fairly simple subset of dynamic logic, namely modal logic.

The rest of this report is organized as follows. In Section 2 we review the definitions and results presented in [3] for propositional and predicate logic. Section 3, 4 and 5 summarize results obtained in [3] In section 6 we show how modal logic can be spatially embedded into a linear systems. Finally, connections of modal logic to the bisimulation relation are described in section 7 together with a simple example.

# 2 Propositional and Predicate Logic

In propositional logic, formulas are built up from propositional variables through the use of the usual boolean connectives $\vee$, $\wedge$ and $\neg$. Propositional variables can evaluate to either *True* or *False*. In order to embed a propositional formula into a linear program, we can associate the numbers 0 and 1 to the the symbolic values *False* and *True*. It is then natural to express disjunctions as $0-1$ linear inequalities and formulas in conjunctive normal form as $0-1$ linear systems.

For example the satisfiability of the logical formula

$$x \vee \neg y \vee z$$

can be embedded as solubility of the inequality

$$x + (1 - y) + z \geq 1$$

where $x, y, z$ are $0-1$ variables.

The conjunctive normal formula

$$(x) \wedge (y \vee \neg z) \wedge (\neg w \vee x \vee \neg y) \wedge (w \vee z)$$

is satisfiable if and only if the following system has solution:

$$\begin{aligned} x &\geq 1 \\ y + (1 - z) &\geq 1 \\ (1 - w) + x + (1 - y) &\geq 1 \\ w + z &\geq 1 \\ x, y, z, w &= 0 \text{ or } 1 \end{aligned}$$

It is conventional in mathematical programming to express linear systems in matrix notation:

$$Ax \geq b$$

where $A$ is a matrix of coefficients, $x$ is a vector of unknown and $b$ is a vector of constants. For example the above system can be rewritten as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \geq \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix}.$$

In general we can associate to each formula in conjunctive normal form $\phi$ a system $Ax \geq b$ of $c$ linear inequalities over $0-1$ variables, where $c$ is the number of clauses in $\phi$. The satisfiability of $\phi$ is equivalent to the solubility of the associated system

$$Ax \geq b, x \in \{0,1\}^n . \tag{1}$$

This spatial embedding of propositional logic is extended in [3] to predicate calculus, using infinite dimensional linear programming. Briefly, a first order formula is transformed into an infinite conjunction of propositional clauses and the resulting infinite propositional formula is embedded into a system $\mathcal{A}x \geq \beta$ over an infinite set $x$ of $0-1$ variables. The passage from predicate calculus to propositional logic uses standard techniques from logic (*Skolemization*). The embedding of infinitary propositional calculus into infinite mathematical programming extends the finitary case in the obvious way. As usual the formula is satisfiable if and only if the associated system has solution. The structure of the resulting system (the *clausal form* and *finite support* property, [3]) and general properties of the product of topological spaces are used in [3] to prove the Herbrand theorem and the existence of a minimal solution for systems of Horn clauses.

We now discuss these results in summary form.

# 3 Infinite Dimensional $0-1$ Linear Programs

Consider a mathematical program of the form

$$\mathcal{D} = \{x \in \{0,1\}^\infty : \mathcal{A}_x \geq \beta\} \tag{2}$$

Each row of the matrix $\mathcal{A}$ has entries that are $0, \pm 1$, and each entry of the (uncountably) infinite column $\beta$ is $1-$ the number of $-1$'s in the corresponding row of $\mathcal{A}$. So this is just an infinite version of (1). The finite support of the rows of $\mathcal{A}$ is the important structural property that permits the compactness theorems based on product topologies to go through in the ensuing development. It is a natural restriction in the context of first order logic as it corresponds to the finite "matrix" property of first order formulae. Note that compactness theorems can be pushed through for more general infinite mathematical programs using the so called "weak $*$ topologies" but this shall not concern us.

In discussing Horn Logic, we will encounter the continuous (linear programming) relaxation of our infinite mathematical program (2).

$$\bar{\mathcal{D}} = \{x \in [0,1]^\infty : \mathcal{A}x \geq \beta\} \tag{3}$$

Let $\{\mathcal{A}_\alpha x \geq \beta_\alpha\}_{\alpha \in \mathcal{I}}$ denote a suitable indexing of all finite subfamilies of $\{\mathcal{A}x \geq \beta\}$. And for each $\alpha$ in the uncountable set $\mathcal{I}$ let

$$\begin{aligned} \mathcal{D}_\alpha &= \{x \in \{0,1\}^\omega : \mathcal{A}_\alpha x \geq \beta_\alpha\} \\ \bar{\mathcal{D}}_\alpha &= \{x \in [0,1]^\omega : \mathcal{A}_\alpha x \geq \beta_\alpha\} \end{aligned}$$

Thus,

$$\mathcal{D} = \cap_{\alpha \in \mathcal{I}} \mathcal{D}_\alpha$$
$$\bar{D} = \cap_{\alpha \in \mathcal{I}} \bar{\mathcal{D}}_\alpha$$

The analysis of finite dimensional mathematical programs such as (1) is based on elementary techniques from combinatorial and polyhedral theory. The situation in the infinite dimensional case gets more complicated. Constraint qualification is a sticky issue even for semi-infinite mathematical programs. The standard approach in infinite dimensional mathematical programming is to impose an appropriate (weak) topological framework on the feasible region and then use the power of functional analysis to develop the structural theory.

# 4 A Compactness Theorem

A classical result in finite dimensional programming states that if a finite system of linear inequalities in $\Re^d$ is infeasible, there is a "small" $(d+1)$ subsystem that is also infeasible. This compactness theorem is a special case of the ubiquitous Helly's Theorem. Analogous theorems are also known for linear constraints on integer valued variables. In the infinite dimensional case, we could hope for the "small" witness of infeasibility to simply be a *finite* witness. This is exactly what we prove for mathematical programs of the form (3) and (2).

Let $\mathcal{S}_\gamma$, $\gamma \in \mathcal{G}$, be copies of a Hausdorff space $\mathcal{S}$. Let $\mathcal{S}^{\mathcal{G}} = \Pi_{\gamma \in \mathcal{G}} \mathcal{S}_\gamma$. The product *topology* on $\mathcal{S}^{\mathcal{G}}$ is the topology defined by a basis $\Pi_\gamma \mathcal{O}_\gamma$ where the $\mathcal{O}_\gamma$ are open in $\mathcal{S}_\gamma$ and $\mathcal{O}_\gamma = \mathcal{S}_\gamma$ for all but at most finitely many $\gamma \in \mathcal{G}$. A classical theorem on compact sets with product topology is that of Tychonoff which states that

**Theorem 4.1** *Arbitrary (uncountable) products of compact sets with product topology are compact.*

Next we show that $\mathcal{D}_\alpha$ and $\bar{\mathcal{D}}_\alpha$, with product topologies, are also compact for any $\alpha$ in $\mathcal{I}$. This follows form the corollary and the lemma below.

**Corollary 4.2** $\{x \in \{0,1\}^\infty\}(\{x \in [0,1]^\infty\})$ *(with product topology) is compact.*

**Lemma 4.3** *The set* $\{x : \mathcal{A}_\alpha \geq \beta_\alpha\}(\alpha \in \mathcal{I})$ *is closed and hence compact.*

**Theorem 4.4** $\mathcal{D}(\bar{\mathcal{D}})$ *is empty if and only if* $\mathcal{D}_\alpha(\bar{\mathcal{D}}_\alpha)$ *is empty for some* $\alpha \in \mathcal{I}$.

# 5 Herbrand Theory and Infinite $0-1$ Programs

We will assume that the reader has a basic familiarity with Predicate Logic. In particular, we assume that the reader is familiar with the Skoleam Normal Form, the Herbrand Universe and the Horn Formula (see [8]).

Assuming now that $H$ is a Horn formula as defined above, we formulate the following infinite dimensional optimization problem.

$$\inf\left\{\sum x_j : \mathcal{A}_x \geq \beta, x \in [0,1]^\infty\right\} \tag{4}$$

where linear inequalities $\mathcal{A}_x \geq \beta$ are simply the clausal inequalities corresponding to the ground clauses of $H$. The syntactic restriction on Horn clauses translates to the restriction that each row of $\mathcal{A}$ has at most one $+1$ entry (all other entries are either 0 or $-1$'s — only finitely many of the latter though). We shall prove now that if the infinite linear program (4) has a feasible solution then it has an integer optimal $(0-1)$ solution. Moreover, this solution will be a least element of the feasible space i.e., it will simultaneously minimize all components over all feasible solutions.

**Lemma 5.1** *If the linear program (4) is feasible then it has a minimum solution.*

**Lemma 5.2** *If $x^1$ and $x^2$ are both feasible solutions for (4) then so is $\{\bar{x}_j = \min(x_j^1, x_j^2)\}$.*

**Theorem 5.3** *If the linear program (4) is feasible, then it has a unique $0-1$ optimal solution which is the least element of the feasible set.*

The interpretation of this theorem in the logic setting is that if a Horn formula $H$ has a model then it has a least model (a unique minimal model). This is an important result in model theory (semantics) of so-called definite logic programs.

# 6 Modal Logic

Modal logic extends classical logic introducing new quantifiers over formulas, called modalities. The set of modalities may be different from one logic to the other. For example, dynamical logic can be viewed as a modal logic where the modalities are programs. Here we consider a special case of dynamic logic where programs are single atomic actions. More precisely the set of modalities is $\{\Box_a\}_{a\in\Sigma}$ (and their duals $\{\Diamond_a\}_{a\in\Sigma}$) where $\Sigma$ is a set of symbols.

A model $M = (W, T, \sigma)$ for a modal formula $\phi$ is given by a set of *worlds* $W$, a family of transition functions $T = \{t_a : W \to 2^W\}_{a\in\Sigma}$ labeled by the symbols in $\Sigma$, and a valuation for the variables $\sigma : V \to 2^W$ that associate to each propositional variable the set of worlds in which the variable is true. Given a model $M = (W, T, \sigma)$ and a world $s$ in $W$ the truth value of a modal formula is defined by induction on the structure of the formula as follows:

- $M, s \models x$ iff $s \in \sigma(x)$,
- $M, s \models \phi \wedge \psi$ iff both $M, s \models \phi$ and $M, s \models \psi$,
- $M, s \models \phi \vee \psi$ iff either $M, s \models \phi$ or $M, s \models \psi$,

- $M, s \models \neg\phi$ iff not $M, s \models \phi$ ,
- $M, s \models \Box_a \phi$ iff $M, t \models \phi$ for all $t \in t_a(s)$,
- $M, s \models \Diamond_a \phi$ iff $M, t \models \phi$ for some $t \in t_a(s)$.

A formula $\phi$ is true in a model $M$, written $M \models \phi$, if $\phi$ is true in all worlds of $M$. A formula $\phi$ is satisfiable iff it is true in some model.

So far, we have defined a logic language that extends propositional logic and we have defined a notion of satisfiability for formulas in that languages. We want to embed the satisfiability of modal formulas into linear problems, as it has been done for propositional logic.

We now propose embedding of modal logic, that preferably preserves the finiteness property of propositional logic. The intuition behind the embedding that we define is to use "timed" linear systems of the form

$$A_0 x(t) + A_1 x(t+1) \geq b$$

where the "time" $t$ is used to express the "dynamics" associated to the modal operators. The above system is of a kind usually encountered in the study of dynamical systems and can be rewritten in a more compact way using a shift operator $^*$ as follows:

$$A_0 x + A_1 x^* \geq b.$$

Here the variable $x$ is a function of time $t$ and the action of the shift operator on $x$ is given by $x^*(t) = x(t+1)$.

In order to simplify the presentation in the rest of this section we will take $\Sigma = \{a\}$ so that we have only two modal operators $\Box$ and $\Diamond$ (the subscript $a$ is omitted for brevity). However, everything can be extended to the general case, with $|\Sigma|$ possibly greater than one, with obvious modifications.

Since the transition function $t$ may associate to each world more than one successor (or even none), the dynamics expressed by the modal operators $\Box$ and $\Diamond$ has a branching structure. Therefore *time* is not the right concept to express the modalities. We will consider a notion of *generalized time* $T$. Variable $x$ is still a function of $T$, but there are two shift operators $^\Box$ and $^\Diamond$ that can act over $x$. Putting it together, we want to embed the satisfiability problem for modal logic into a system of the kind

$$A_0 x + A_1 x^\Box + A_2 x^\Diamond \geq b.$$

where the vector $x$ is a function of $T$ and the action associated to the shift operators $^\Box$ and $^\Diamond$ is the following. We have said that $T$ can be thought as a "time" in a broad sense (carrying on this analogy, we call *istants* the elements of $T$). Each istant in $T$ may have more than one immediate successor in $T$. Let $\tau$ be a function that associates to each istant the set of its immediate successors in $T$. The result $x^\Box$ of applying the $^\Box$ shift to $x$ is the set of all possible values that vector $x$ can take after one unit of "time". Analogously the result $x^\Diamond$ of applying the $^\Diamond$ shift to $x$ is some of the possible values that vector $x$ can take after one unit of "time".

Now we look at how modal formulas can be represented in this framework. Clearly any propositional formula that doesn't make use of the modalities can be embedded into a system with $A_1$ and $A_2$ both equal to the null matrix. Also, formulas that make very simple use of the modalities can be directly embedded. For example the formula

$$(x \vee \neg(\Diamond y \wedge z)) \wedge (\Box z \rightarrow z)$$

can be rewritten as

$$(x \vee \neg z \vee \Box\neg y) \wedge (\Diamond\neg z \vee z)$$

and then represented by the system

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} x + \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} x^{\Box} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} x^{\Diamond} \geq \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

Things get harder if the formula makes a more complex use of modal operators. For example there is no direct way to express the formula $\Diamond\Box x$ directly into our system. It seems that the flat structure of the linear system does not allow us to represent nested modal operators. A more subtle problem arises when translating the formula $\Box x \vee \Box y$. One could be tempted to embed this formula into the system

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}^{\Box} \geq 1.$$

At first sight this seems correct but a more careful exam shows that the meaning of the above system is the formula $\Box(x \vee y)$ which is not equivalent to $\Box x \vee \Box y$. In fact, the shift operator $^{\Box}$ acts on the vector $\begin{bmatrix} x \\ y \end{bmatrix}$ as a whole and therefore we cannot choose $x^{\Box}$ and $y^{\Box}$ independently of each other.

We will now illustrate an embedding technique that solves the above problems and allows the encoding of arbitrarily complex formulas into linear systems. The resulting system is finite, and its size is not significantly greater of the starting modal formula. These ideas are due to D. Micciancio [9].

The method is based on the introduction of new variables associated to subexpressions of the logic formula and is defined as a recursive procedure **Embed**($\phi$). On input a formula $\phi$ of propositional modal logic, **Embed**($\phi$) returns a system of linear equations over the variables of $\phi$, plus some fresh variables introduced during the execution of the procedure, whose solubility over $0-1$ variables is equivalent to the satisfiability of the original formula $\phi$. First we define a procedure to embed formulas of the form $x \leftrightarrow \phi$:

**Embed**($x \leftrightarrow \phi$)

- if $\phi = x$, then return $\{x \geq 1\}$,
- if $\phi = \neg\psi$, then introduce a fresh variable $z$ and return

$$\{-x - z \geq -1, x + z \geq 1\} \cup \textbf{Embed}(z \leftrightarrow \psi)$$

- if $\phi = \psi_1 \wedge \psi_2$, then introduce two fresh variables $z_1$ and $z_2$ and return

$$\{-x + z_1 \geq 0, -x + z_2 \geq 0, x - z_1 - z_2 \geq -1\}$$

$$\cup \mathbf{Embed}(z_1 \leftrightarrow \psi_1) \cup \mathbf{Embed}(z_2 \leftrightarrow \psi_2)$$

- if $\phi = \psi_1 \vee \psi_2$, then introduce two fresh variables $z_1$ and $z_2$ and return

$$\{x - z_1 \geq 0, x - z_2 \geq 0, -x + z_1 + z_2 \geq 0\}$$

$$\cup \mathbf{Embed}(z_1 \leftrightarrow \psi_1) \cup \mathbf{Embed}(z_2 \leftrightarrow \psi_2)$$

- if $\phi = \Box\psi$, then introduce a fresh variable $z$ and return

$$\{-x + z^{\Box} \geq 0, x - z^{\Box} \geq 0\} \cup \mathbf{Embed}(z \leftrightarrow \psi)$$

- if $\phi = \Diamond\psi$, then introduce a fresh variable $z$ and return

$$\{-x + z^{\Diamond} \geq 0, x - z^{\Box} \geq 0\} \cup \mathbf{Embed}(z \leftrightarrow \psi)$$

The general case easily follows. Any formula $\phi$ can be embedded into the linear system $\{z \geq 1\} \cup \mathbf{Embed}(z \leftrightarrow \phi)$ where $z$ is a variable not occurring in $\phi$.

Applying the function **Embed** to the formula $\Diamond\Box x$ we get the system

$$\begin{aligned} z_1 &\geq 1 \\ -z_1 + z_2^{\Diamond} &\geq 0 \\ z_1 - z_2^{\Diamond} &\geq 0 \\ -z_2 + z_3^{\Box} &\geq 0 \\ z_2 - z_3^{\Box} &\geq 0 \\ -z_3 + x &\geq 0 \\ z_3 - x &\geq 0 \end{aligned}$$

Obviously we could have embedded the same formula in the smaller system

$$\begin{aligned} z^{\Diamond} &\geq 1 \\ -z + x^{\Box} &\geq 0 \\ z - x^{\Diamond} &\geq 0. \end{aligned}$$

However, even if the system obtained by applying **Embed** is not the smallest possible, it can be formally proved the the result of the given procedure is never much bigger than necessary. Namely the system $\mathbf{Embed}(\phi)$ has at most $3n + 1$ rows where $n$ is the size of the formula $\phi$.

The last system can be written in matrix notation as

$$\begin{bmatrix} 0 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix}^{\Box} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix}^{\Diamond} \geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Here we see how the introduction of a new variable $z$ allows us to represent a formula with nested modal operators.

Now consider the formula $\Box x \vee \Box y$, we have already remarked that this formula cannot be straightforwardly translated into the system

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}^{\Box} \geq 1$$

which in fact represent a different formula, namely $\Box(x \vee y)$. Let's see how expressions with multiple modal operators in the same clause are handled. The result of applying the embedding function to formula $\Box x \vee \Box y$ is the system

$$\begin{aligned}
z_1 &\geq 1 \\
-z_1 + z_2 + z_3 &\geq 0 \\
z_1 - z_2 &\geq 0 \\
z_1 - z_3 &\geq 0 \\
-z_2 + z_4^{\Box} &\geq 0 \\
z_2 - z_4^{\Box} &\geq 0 \\
-z_3 + z_5^{\Box} &\geq 0 \\
z_3 - z_5^{\Box} &\geq 0 \\
-z_4 + x &\geq 0 \\
z_4 - x &\geq 0 \\
-z_5 + y &\geq 0 \\
z_5 - y &\geq 0
\end{aligned}$$

or with a few simplifications

$$\begin{aligned}
z + y^{\Box} &\geq 1 \\
-z + x^{\Box} &\geq 0 \\
z - x^{\Box} &\geq 0.
\end{aligned}$$

This last example shows how introducing a new variable $z$ we can split a clause with multiple occurrences of the same modal operator into the conjunction of several clauses each of which contains at most one modal operator.

We have showed how any formula of modal logic can be translated into a "small" linear system of the form

$$A_0 x + A_1 x^{\Box} + A_2 x^{\Diamond} \geq b.$$

The equivalence of the system with the modal formula can be easily proved by induction on the size of the formula. The linear system has the same "clausal form" property shown in [3] for the propositional logic embedding. Another property enjoyed by this linear system is that each row of the matrices $A_1$ and $A_2$ has at most one non-null entry. It is because of this last property that the shift operators $^{\Box}$ and $^{\Diamond}$ can be applied to the unknown vector $x$ as a whole, as opposed to being applied componentwise.

# 7 Modal Logic and Bisimulation

The relevance of modal logic in the context of modeling distributed computing is exemplified by its relationship with bisimulation, a widely accepted equivalence relation between labeled transition systems.

A labeled transition system is a graph whose edges are labeled with symbols from some alphabet $\Sigma$. Formally a labeled transition system is a tuple $(N, E, L)$ where $N$ is a set of nodes, $E$ is a binary relation on $N$ and $L$ is a function from $N$ to $\Sigma$. The nodes $N$ represent the possible internal states of a process or set of processes, the labels $\Sigma$ are actions the system may perform, and the edges of the graph $E$ express how the internal state of the system changes following the execution of an action. Usually some node $s \in N$ is designated as the starting node, the initial state of the process represented by the transition system.

Two labeled transition systems $(N_1, E_1, L_1, s_1)$ and $(N_2, E_2, L_2, s_2)$ are bisimilar if there exists a binary relation $R \subseteq N_1 \times N_2$ such that

- $(s_1, s_2) \in R$
- for all $(t_1, t_2) \in R$:
  - if $(t_1, t_1') \in E_1$, then there exists some $t_2'$ such that $(t_2, t_2') \in E_2$ and $L_2(t_2, t_2') = L_1(t_1, t_1')$.
  - if $(t_2, t_2') \in E_2$, then there exists some $t_1'$ such that $(t_1, t_1') \in E_1$ and $L_2(t_1, t_1') = L_1(t_2, t_2')$.

It is natural to view labeled transition systems as models for modal logic. The nodes in the graph are the worlds of the model and the transition relation $t_a$ maps node $s$ to the set $\{t : (s, t) \in E\}$. We can ask when two labeled transition systems can be distinguished by modal formulas. In other words, given two labeled transition systems we look for some formula that is true in one system but false in the other. Two labeled transition systems are considered equivalent if no such formula exists.

It turns out that this notion of equivalence is exactly bisimilarity. Two labeled transition systems are bisimilar if and only if they satisfy the same set of modal formulas. For a formal proof of this statement together with a more accurate description of the relationship between modal logic and bisimulation the reader is referred to [2].

Here we will only illustrate the mentioned result on a simple scheduler example taken from [1]. The scheduler described in [1] communicates with a set $\{P_i\}_i$ of $n$ processes through the actions $a_i$ and $b_i$ $(i = 1, \ldots, n)$. These actions have the following meaning:

- action $a_i$ signals $P_i$ has started executing,
- action $b_i$ signals $P_i$ has finished its performance.

Each process wishes to perform its task repeatedly. The scheduler is required to satisfy the following specification:

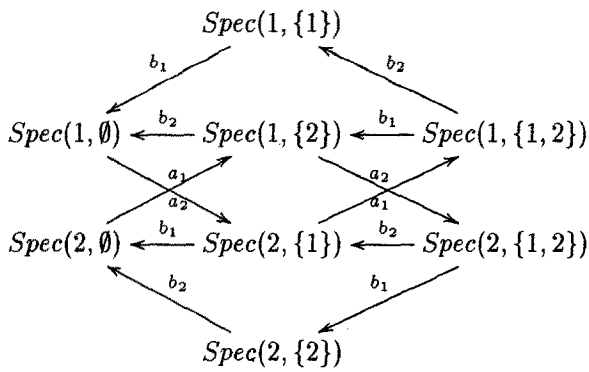


Figure 1: Simple scheduler specification

- actions $a_1, \ldots, a_n$ are performed cyclically starting with $a_1$,
- action $a_i$ and $b_i$ are performed alternatively.

Informally processes start their task in cyclic order starting with $P_1$ and each process finish one performance before it begins another.

Then a modular implementation of the scheduler is suggested. The implementation is based on a set of $n$ components $C_1, \ldots, C_n$ connected in cycle that pass a token each other in cyclic order. There is exactly one token, initially owned by $C_1$, going around. Furthermore, each component $C_i$ performs action $a_i$ after receiving the token and before passing it to $a_{(i \bmod n)+1}$. Then after the token has been passed to $a_{(i \bmod n)+1}$, $C_i$ performs $b_i$ before receiving the token again. For a more accurate description of this example the reader is referred to the original text [1, pages 113–123] where both the specification and the implementation of the scheduler are formally given using the *CCS* language.

If the number $n$ of processes being scheduled equals two, the specification is given by the labeled transition system shown in Figure 1, while the implementation gives the system described by the labeled transition system in Figure 2.

If the system $[C_1|\ldots|C_n]$ were a correct implementation of the specification, the two systems in Figure 1 and 2 would not be distinguishable by any modal formula. However this is not the case since formula $s \rightarrow \Box_{a_1}\Box_{a_2}\Diamond_{b_2}t$[1] is true in the system depicted in Figure 1 but not in the one shown in Figure 2. The formula $s \rightarrow \Box_{a_1}\Box_{a_2}\Diamond_{b_2}t$ can be translated into the linear system

$$\begin{aligned} -s + x^{\Box_{a_1}} &\geq 0 \\ -x + y^{\Box_{a_2}} &\geq 0 \\ x - y^{\Diamond_{a_2}} &\geq 0 \\ -y + t^{\Diamond_{b_2}} &\geq 0 \end{aligned}$$

[1]Here $s$ is a predicate true only in the starting state and $t$ is a predicate always true

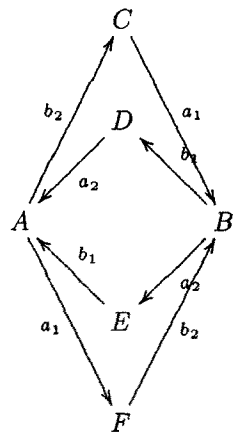


Figure 2: Simple scheduler implementation

$$\begin{aligned} y - t^{\Box b_2} &\geq 0 \\ t &\geq 0 \end{aligned}$$

which has solution

$$\begin{aligned} s &= \{Spec(1, \emptyset)\} \\ x &= \{Spec(2, \{1\})\} \\ y &= \{Spec(1, \{1, 2\})\} \\ t &= \{Spec(i, S) : i \in \{1, 2\}, S \subseteq \{1, 2\}\} \end{aligned}$$

in the model associated to the system in Figure 1 but has no solution in the model associated to the implementation. In conclusion the linear system shows that the proposed implementation of the scheduler does not satisfy the given specification.

# References

[1] R. Milner, *Communication and Concurrency.* Prentice Hall, London (1989).

[2] J. van Benthem, J. van Eijck, V. Stebletsova, *Modal logic, transition systems and processes* Math Centrum, CS-R9321 (1993).

[3] V. S. Borkar, V. Chandru, S. K. Mitter, *A Linear Programming Model of First Order Logic* Indian Institute of Science, TR IISc-CSA-95-5 (1995).

[4] R. G. Jeroslow, *Logic-Based Decision Support: Mixed Integer Model Formulation* Annals of Discrete Mathematics 40. North-Holland (Amsterdam 1989).

[5] R. G. Jeroslow, *Computation-oriented reductions of predicate to propositional logic* Decision Support Systems 4 (1988) 183–197.

[6] V. R. Pratt, *Process Logic*, Proc. 6th Ann. ACM Symp. on Principle of Programming Languages (Jan. 1979).

[7] V. R. Pratt, *Dynamic Logic*, Proc. 6th International Congress for Logic, Philosophy, and Methodology of Science, (Hanover, Aug. 1979).

[8] U. Schöning, *Logic for Computer Scientists*, Birkhäuser (1989).

[9] D. Micciancio and S. K. Mitter: Forthcoming LIDS Technical Report, M.I.T.