Transformation of Logical Specification into IP-formulas

Qiang Li Yike Guo* Tetsuo Ida**

Research Center
PFU Limited, Japan
*Imperial College
180 Queen's Gate, London SW7 2BZ, U.K.
**Institute of Information Sciences and Electronics
University of Tsukuba, Japan

1 Introduction

Integer programming (IP) is an important mathematical optimisation technique which attempts to find the best possible answer to a given problem. The standard formulation of IP problems is defined as:

$$\text{Maximise } CX \text{ subject to } AX \leq b.$$

where C is an integer n-vector, A a $m \times n$ integer matrix, b an integer m-vector and X a n-vector of natural numbers.
Solving this formulation is to find the values of the decision variables X that maximise the value of the objective function CX, called objective in short, while satisfying the constraints $AX \leq b$.

Since the late 1940's there has been a large number of articles concerned with IP problems. However practical adoption of the IP techniques has been hampered. We identify two major bottlenecks that have led to this. Firstly the limitation of the classical approach has been one of modelling. Both the specification and analysis of the solution of real world problems are highly complex. Hence, non-specialist potential users, such as managers, are discouraged from formulating the problem or interpreting the results. Secondly, progress has been limited by the lack of effective algorithms to tackle the real world problems and the absence of good high performance computing environment to compute solutions at a reasonable speed.

We have been working on parallel methods for IP and stochastic IP for decision support systems [8][9][10]. In order to exploit the power of these new algorithms in real world applications, a declarative modelling system becomes necessary, that enables us to specify complex decision support procedures. Logically specified decision problems will be transformed into admissible constraints which will then be solved using constraint solvers on a parallel computer.

In this paper, we describe a modelling language for IP based on the predicate calculus and present a systematic algorithm for transforming logic formulas into IP formulas. We discuss the implementation for this algorithm. The rest of the paper is organized as follows. Section 2 contains the brief review of approaches for logic modelling IP. In section 3, we describe a language for modeling IP in the framework of the first-order logic. In section 4, we present an algorithm for transforming logical formulas into IP formulas. In section 5 we discuss the implementation of the algorithm in Mathematica. Finally, we give conclusion and future works in section 6.

2 Review of logical modelling frameworks for IP

Since the early 50's many discrete optimization techniques have been developed and have made a significant contribution to solving decision problems using a quantitative (model) representation. During the 80's some logic-based modelling frameworks for IP have been proposed. These frameworks can be divided into two classes: constructing IP models by the predicate calculus and constructing IP models by the propositional calculus. The first class may be represented by the work of McKinnon & Williams [1]. The second may be represented by the works of Hadjiconstantinou & Mitra[2] and Raman & Grossmann[3].

Hadjiconstantinou & Mitra propose a modelling tool which employs propositional calculus as a basis for formalizing the modelling of IP. They transform the tree-represented logical formulas directly into IP forms.

Raman & Grossmann propose a modelling framework based on propositional calculus and a specialized algorithm for discrete linear programming problems. Their approach is to convert the logical formulas in the propositional calculus model into the conjunctive normal form and then to obtain the equivalent IP form.

Mckinnon & Williams propose a logical modelling framework based on the predicate calculus and a series of transformation rules, which automatically converts all the input predicates into an intermediate form called ge-form, from which an IP model can be generated.

The predicate calculus offers a more flexible modelling tool than the propositional calculus. In the predicate calculus one can declare predicates (n-ary relations) between objects. But it is difficult to transform the predicate into IP forms directly. By introducing ge-form, Mckinnon & Willams transform the predicate into ge-forms first and then transform ge-forms into IP forms. Mckinnon & Willams' method based on the predicate calculus has, thus, the obvious advantages over the previous other works. However, their method lacks rigorous definition of modelling language and systematic transformation algorithm. In this paper, we extend McKinnon & Williams' idea and propose a modelling language $\mathcal{L}^+$ based on the predicate logic and present rules for transforming formulas of the language into IP formulas with rigorous proofs of the soundness of the transformation. Our language for modelling IP is presented in the next section.

3 Modelling language $\mathcal{L}^+$

In this section we describe a modelling language $\mathcal{L}^+$ for describing IP problems. Our aim is to dispense with the commonly used algebraic model by a logical model in problem specification. An algebraic model is an optimisation-oriented analytical model used in mathematical programming. Real world problems are described in mathematical terms. The main components of the formulation are decision variables, constraints and objective functions. The solver tries to find values of the decision variables that maximise or minimise the values of the objective functions while satisfying the constraints.

We assume the language $\mathcal{L}$ of the first-order predicate logic in this paper. Our modelling language $\mathcal{L}^+$ extends $\mathcal{L}$ by introducing meta predicates at_least(m, S), at_most(m, S), and none(S) where m is an integer and S is a set of $\mathcal{L}^+$ formulas. The relation at_least(m, S) (at_most(m, S)) expresses that at least (at most) m formulas in S must hold, while none(S) expresses that no formula in S holds.

By the use of $\mathcal{L}^+$, we can describe IP problems more naturally and logically, yet keeping track of the algebraic properties of the original IP problems.

Now, we define the language $\mathcal{L}^+$. $\mathcal{L}^+$ is a language of quantifier-free formulas built over the following alphabet.

(a) improper symbols: (,), {, } and logical connectives $\vee, \wedge, \rightarrow, \leftrightarrow, \neg$

(b) integer variables: $d, x, y, z, \ldots$

(c) propositional variables: $p, q, r, \ldots$

(d) integer constants: ...,-2,-1,0,1,2,...

(e) function symbols: $+, -, \times$

(f) predicate symbols: $=, \geq, \leq, >, <$

The function symbol $-$ is used as both unary and binary operators. In the former case it is prefix, in the latter it is infix. All the other function and predicate symbols are binary and infix. Function symbols are used to form arithmetic terms, and predicate symbols to form constraints over arithmetic terms.

Following are some examples of arithmetic predicates. Let x, y, z be integer variables. Expressions $2x$, $3y$, $4z$, x, 10, $6z$, $2x + 3y - 4z$, $x - y$, $10 + 6z$ are arithmetic terms. For brevity we omit $\times$ in $2x$, $3y$, $4z$, and $6z$. Expressions $2x + 3y - 4z \geq 0$ and $x - y = 10 + 6z$ are constraints specified by arithmetic relations between the arithmetic terms $2x + 3y - 4z$ and 0 in the former, and $x - y$ and $10 + 6z$ in the latter.

In addition to these standard arithmetic terms, we will use, as a meta notation, a summation of arithmetic terms $\sum_i a_i x_i$, where a_i is an integer coefficient of integer variable x_i. Constraints and propositional variables are combined to form logical formulas. The formation rules for well formed logical formulas (hereafter we simply call formulas) are as follows:

1. A propositional variable or a constraint is an atomic formula.
2. An atomic formula is a formula.
3. $P_1 \; \Xi \; P_2$ is a formula if P_1 and P_2 are formulas, and Ξ is one of the logical connectives $\vee$, $\wedge$, $\rightarrow$ and $\leftrightarrow$.
4. $\neg P$ is a formula if P is a formula.
5. at_least(m, S) is a formula if $m(\geq 1)$ is an integer and S is a set of formulas.
6. at_most(m, S) is a formula if $m(\geq 1)$ is an integer and S is a set of formulas.
7. none(S) is a formula if S is a set of formulas.

A set is represented a set of formulas in the usual way. For example, $\{P_1, ..., P_n\}$ is a set of formulas if $P_1, ..., P_n$ are formulas.

The first four definition rules (1.to 4.) are standard definitions of propositional formulas (see, e.g. standard text book[6]), while the last three rules (5. to 7.) are newly introduced. As we see in Rule $\mathcal{T}$ in section 4.1, at_most(m, S) and none(S) are expressed by at_least(m, S), hence they are unnecessary theoretically. They are provided as a primitive for convenience in describing IP problems. We assume the standard precedence rules for logical connectives and the use of parentheses for changing the precedence. For example,

$$\neg p \vee x \leq 2 \leftrightarrow y \leq 8 \wedge z \geq 7$$

is understood as

$$((\neg p) \vee (x \leq 2)) \leftrightarrow ((y \leq 8) \wedge (z \geq 7))$$

Example 3.1 If there are at most 2 analysts or at most 3 administrators in a research group for a project then there must be at most 8 programmers or at least 7 engineers and vice versa.

Let integer variables x_1, x_2, x_3 and x_4 represent the number of analysts, administrators, programmers and engineers, respectively. We model the above problem in $\mathcal{L}^+$:

$$x_1 \leq 2 \vee x_2 \leq 3 \leftrightarrow x_3 \leq 8 \vee x_4 \geq 7$$

4 Algorithm for transformation on $\mathcal{L}^+$

A specification of an IP-problem expressed as an $\mathcal{L}^+$-formulas, although rigorous, will not be very helpful from computational point of view unless the specification can be executed in some way and it delivers a solution of the IP-problem. There would be a method for directly executing $\mathcal{L}^+$-formulas, but this is not what we aim for since we know already that in the domain of our interest we have efficient methods for solving IP-problems. Therefore we transform $\mathcal{L}^+$-formulas to expressions that IP-solvers will accept, i.e. a system of linear constraints, which we call formulas of IP-form, or simply IP-formulas. In this section, we present an algorithm that transforms $\mathcal{L}^+$-formulas to IP-formulas.

Our algorithm will use an intermediate form to be called Γ-form like ge-form in the method proposed by Mckinnon & Williams. A formula of Γ-form or Γ-formula in short, is either an atomic $\mathcal{L}^+$-formula or an expression of the form $\Gamma_m S$, meaning that m or more formulas in S are true. An IP-formula is a set of constraints $\{C_1, \ldots, C_n\}$, where each constraint C_i is a linear equality or a linear inequality involving with integer variables and constants.

Any $\mathcal{L}^+$-formula can be transformed to a Γ-formula since we have the following logical equivalence:

$$\Gamma_1 \{P_1, \ldots P_n\} \equiv P_1 \vee \cdots \vee P_n \tag{4.1}$$

and

$$\Gamma_n \{P_1, \ldots, P_n\} \equiv P_1 \wedge \cdots \wedge P_n \tag{4.2}$$

and any $\mathcal{L}^+$-formulas are transformed to a logically equivalent normal forms of either a DNF (disjunctive normal form) or a CNF (conjunctive normal form). Here we say that P and Q are logically equivalent, written as $P \equiv Q$, iff for any model and valuation the value of P and Q are the same.

With this observation one would conclude that CNF or DNF would be used instead of Γ-form. However, this would lead to formulations of large size, resulting in an unnecessarily large set of IP-formulas.

In summary our approach for modelling IP-problems consists in the following steps:

1. to describe an IP-problem in $\mathcal{L}^+$-formulas,
2. to transform $\mathcal{L}^+$-formulas to Γ-formulas,
3. to transform Γ-formulas to IP-formulas, and
4. solving IP-formulas by a specialised IP-solver.

4.1 Transformation of $\mathcal{L}^+$-formulas into Γ-formulas

First we introduce an equivalence rule $\mathcal{E}$. By this rule, all $\mathcal{L}^+$-subformulas of the form of the left-hand side of the rules are eliminated. Then resulting $\mathcal{L}^+$-formulas are transformed by rule $\mathcal{T}$ to Γ-formulas.

Equivalence rule $\mathcal{E}$:

Let P, Q be $\mathcal{L}^+$-formulas.

$(\mathcal{E}_1)$: $\mathcal{E}[\neg\neg P] = \mathcal{E}[P]$
$(\mathcal{E}_2)$: $\mathcal{E}[P \to Q] = \mathcal{E}[\neg P] \vee \mathcal{E}[Q]$
$(\mathcal{E}_3)$: $\mathcal{E}[P \leftrightarrow Q] = \mathcal{E}[P \to Q] \wedge \mathcal{E}[Q \to P)$
$(\mathcal{E}_4)$: $\mathcal{E}[P \vee Q] = \mathcal{E}[P] \vee \mathcal{E}[Q]$
$(\mathcal{E}_5)$: $\mathcal{E}[P \wedge Q] = \mathcal{E}[P] \wedge \mathcal{E}[Q]$
$(\mathcal{E}_6)$: $\mathcal{E}[A] = A$ if A is an atomic formula

Rule $\mathcal{T}$: transformation of $\mathcal{L}^+$-formula into Γ-formula

Let P, Q be $\mathcal{L}^+$-formulas, S be a set of $\mathcal{L}^+$-formulas, $\overline{S} = \{\neg P | P \in S\}$, and $|S|$ be the cardinality of S.

$(\mathcal{T}_1)$: $\mathcal{T}[P \vee Q] = \Gamma_1\{\mathcal{T}[P], \mathcal{T}[Q]\}$
$(\mathcal{T}_2)$: $\mathcal{T}[P \wedge Q] = \Gamma_2\{\mathcal{T}[P], \mathcal{T}[Q]\}$
$(\mathcal{T}_3)$: $\mathcal{T}[\text{at\_least}(m, S)] = \Gamma_m \mathcal{T}'[S]$
$(\mathcal{T}_4)$: $\mathcal{T}[\text{at\_most}(m, S)] = \Gamma_{|S|-m} \mathcal{T}'[\overline{S}]$
$(\mathcal{T}_5)$: $\mathcal{T}[\text{none}(S)] = \Gamma_{|S|} \mathcal{T}'[\overline{S}]$
$(\mathcal{T}_6)$: $\mathcal{T}[\neg P] = \neg \mathcal{T}[P]$
$(\mathcal{T}_7)$: $\mathcal{T}[A] = A$ if A is an atomic formula
where $\mathcal{T}'[\{P_1, \ldots, P_n\}] = \{\mathcal{T}[P_1], \ldots, \mathcal{T}[P_n]\}$

After applying $\mathcal{T}$-rule, we eliminate negated Γ-formulas by the following rule.

Rule $\mathcal{N}$: elimination of negated Γ-formulas

$(\mathcal{N}_1)$: $\mathcal{N}[\neg\Gamma_m S] = \Gamma_{|S|-m+1} \mathcal{N}'[\overline{S}]$
$(\mathcal{N}_2)$: $\mathcal{N}[\neg(\sum_{j=1}^n a_j x_j \Theta b)] = \sum_{j=1}^n a_j x_j \overline{\Theta} b$
$(\mathcal{N}_3)$: $\mathcal{N}[\neg(\sum_{j=1}^n a_j x_j = b)] =$
$\Gamma_1\{\sum_{j=1}^n a_j x_j < b, \sum_{j=1}^n a_j x_j > b\}$
$(\mathcal{N}_4)$: $\mathcal{N}[\neg p] = \neg p$ if p is a propositional variable
$(\mathcal{N}_5)$: $\mathcal{N}[\Gamma_m S] = \Gamma_m \mathcal{N}'[S]$
$(\mathcal{N}_6)$: $\mathcal{N}[A] = A$ if A is an atomic formula

where $\mathcal{N}'[\{P_1, ..., P_n\}] = \{\mathcal{N}[P_1], ..., \mathcal{N}[P_n]\}$,
Θ is one of $\leq, \geq, <, >$, and
$\overline{\Theta}$ is one of $>, <, \geq, \leq$, respectively.

Repeated application of the following flattening rules may further simplify nested Γ-formulas.

Rule $\mathcal{F}$: Flattening of Γ-formulas

$(\mathcal{F}_1)$: $\mathcal{F}[\Gamma_1(\{\Gamma_1 S_1\} \cup S_2)] = \Gamma_1(S_1 \cup S_2)$
$(\mathcal{F}_2)$: $\mathcal{F}[\Gamma_{|S_2|+1}(\{\Gamma_{|S_1|} S_1\} \cup S_2)] = \Gamma_{|S_1 \cup S_2|}(S_1 \cup S_2)$

Before we proceed further, we make sure that the above rules $\mathcal{E}$, $\mathcal{T}$, $\mathcal{N}$ and $\mathcal{F}$ are correct. Obviously rule $\mathcal{E}$ is correct since it is a standard logical equivalence. We consider transformation rule $\mathcal{T}$, $\mathcal{N}$ and $\mathcal{F}$. We rely on certain algebraic properties that hold for domains of integers and operators that are applied to integer values. Hence, it suffices to define the correctness of the transformations in a specialised domain.

Let $\mathcal{M}$ be a model whose domain of interpretation is the set of integers. Arithmetic operators used in the language of $\mathcal{L}^+$ are interpreted as usual; e.g. $+$ is an addition of integers. Then we define the correctness of the transformation as follows.

Definition 4.1 We call a transformation rule $\mathcal{H}$ $\mathcal{M}$-correct if

$$\mathcal{M}[\![\mathcal{H}[P]]\!]_\phi = \mathcal{M}[\![P]\!]_\phi$$

for any valuation ϕ associated with $\mathcal{M}$.

By (4.1) and (4.2), it is easy to see that the transformation rule $\mathcal{T}$ is $\mathcal{M}$-correct for any model $\mathcal{M}$. The flattening rule $\mathcal{F}$ is a consequence of the associativity of connective $\vee$. So, $\mathcal{F}$ is $\mathcal{M}$-correct for any model $\mathcal{M}$. It is easy to prove rule $\mathcal{N}$ is $\mathcal{M}$-correct by analysis of the values of $\mathcal{M}[\![\mathcal{N}[\mathcal{P}]]\!]_\phi$ and $\mathcal{M}[\![\mathcal{P}]\!]_\phi$.

By the transformation rules $\mathcal{T}$, $\mathcal{E}$ and $\mathcal{F}$, we can translate all $\mathcal{L}^+$-formulas into Γ-formulas.

Example 4.1 We translate the following $\mathcal{L}^+$-formula into a Γ-formula.

$$p_1 \wedge p_2 \rightarrow p_3 \vee p_4$$

We first translate the above formula into

$$\neg(p_1 \wedge p_2) \vee (p_3 \vee p_4)$$

by rule $(\mathcal{E}_2)$.
By rule $(\mathcal{T}_1)$, we get a Γ-form:

$$\Gamma_1\{\neg\ \Gamma_2\{p_1, p_2\}, \Gamma_1\{p_3, p_4\}\}.$$

We eliminate negated Γ-formulas by $(\mathcal{N}_1)$:

$$\Gamma_1\{\Gamma_1\{\neg\ p_1, \neg\ p_2\}, \Gamma_1\{p_3, p_4\}\}.$$

Finally, we flatten the Γ-formula by $(\mathcal{F}_1)$:

$$\Gamma_1\{\neg\ p_1, \neg\ p_2, p_3, p_4\}.$$

Example 4.2 We transform the $\mathcal{L}^+$-formulas in Example 3.1 into a Γ-formula.
$x_1 \leq 2 \vee x_2 \leq 3 \leftrightarrow x_3 \leq 8 \vee x_4 \geq 7$
By rule $(\mathcal{E}_3)$, we translate the above formula into the following formula:

$$(\neg(x_1 \leq 2 \vee x_2 \leq 3) \vee (x_3 \leq 8 \vee x_4 \geq 7)) \wedge$$
$$((\neg(x_3 \leq 8 \vee x_4 \geq) \vee (x_1 \leq 2 \vee x_2 \leq 3))$$

By $(\mathcal{T}_1)$ and $(\mathcal{T}_2)$ we get a Γ-formula:

$$\Gamma_2\{\Gamma_1\{\neg\ \Gamma_1\{x_1 \leq 2, x_2 \leq 3\}, \Gamma_1\{x_3 \leq 8, x_4 \geq 7\}\},$$
$$\Gamma_1\{\neg\ \Gamma_1\{x_3 \leq 8, x_4 \geq 7\}, \Gamma_1\{x_1 \leq 2, x_2 \leq 3\}\}\}.$$

We apply rule $(\mathcal{N}_1)$ to eliminate negated Γ-formulas.

$$\Gamma_2\{\Gamma_1\{\Gamma_2\{x_1 > 2, x_2 > 3\}, \Gamma_1\{x_3 \leq 8, x_4 \geq 7\}\},$$
$$\Gamma_1\{\Gamma_2\{x_3 > 8, x_4 < 7\}, \Gamma_1\{x_1 \leq 2, x_2 \leq 3\}\}\}.$$

Then by $(\mathcal{F}_1)$ we obtain the following Γ-formula:

$$\Gamma_2\{\Gamma_1\{\Gamma_2\{x_1 > 2, x_2 > 3\}, x_3 \leq 8, x_4 \geq 7\},$$
$$\Gamma_1\{\Gamma_2\{x_3 > 8, x_4 < 7\}, x_1 \leq 2, x_2 \leq 3\}\}.$$

4.2 Transformation of Γ-formulas into IP-formulas

We give the transformation rule $\mathcal{R}$ of Γ-formulas into IP-formulas in Figure 1. The key idea of the transformation is that (1) we introduce decision variables that take the values of either 1 (implying true) or 0 (implying false) and that (2) propositional variables are treated in the same way as in decision variables. Suppose that p denotes a propositional variable that represents a proposition describing an action, option or decision. With a propositional variable p, we associate a decision variable δ_p with the property:

$$\delta_p = 1 \quad \text{iff} \quad \mathcal{M}[\![p]\!]_\phi = \text{T, and}$$
$$\delta_p = 0 \quad \text{iff} \quad \mathcal{M}[\![p]\!]_\phi = \text{F}$$

for any valuation ϕ.

By introducing new variable d_i $(i = 1, ..., n)$, we translate a Γ-formula into a conditional Γ-formula of the form $d_i \rightarrow P$, which then is translated into an IP-formula.

$(\mathcal{R}_1)$: $\mathcal{R}[p] = \delta_p \geq 1$
$(\mathcal{R}_2)$: $\mathcal{R}[\neg p] = \delta_p \leq 0$
$(\mathcal{R}_3)$: $\mathcal{R}[A] = A$ if A is a constraint
$(\mathcal{R}_4)$: $\mathcal{R}[\Gamma_m\{P_1, ..., P_n\}] = \exists \overline{d}\{d_1 + \cdots + d_n \geq m,\ \mathcal{R}[d_1 > 0 \rightarrow P_1], ..., \mathcal{R}[d_n > 0 \rightarrow P_n]\}$
$(\mathcal{R}_5)$: $\mathcal{R}[d > 0 \rightarrow p] = \delta_p \geq d$
$(\mathcal{R}_6)$: $\mathcal{R}[d > 0 \rightarrow \neg p] = 1 - \delta_p \geq d$
$(\mathcal{R}_7)$: $\mathcal{R}[d > 0 \rightarrow \Gamma_m\{P_1, ..., P_n\}] = \exists \overline{d}\{d_1 + \cdots + d_n \geq md,\ \mathcal{R}[d_1 > 0 \rightarrow P_1], ..., \mathcal{R}[d_n > 0 \rightarrow P_n]\}$
$(\mathcal{R}_{8.1})$: $\mathcal{R}[d > 0 \rightarrow \sum_{j=1}^n a_j x_j \leq b] = \sum_{j=1}^n a_j x_j - b \leq U(1-d)$
$(\mathcal{R}_{8.2})$: $\mathcal{R}[d > 0 \rightarrow \sum_{j=1}^n a_j x_j < b] = \sum_{j=1}^n a_j x_j - b < (U+1)(1-d)$
$(\mathcal{R}_{8.3})$: $\mathcal{R}[d > 0 \rightarrow \sum_{j=1}^n a_j x_j \geq b] = \sum_{j=1}^n a_j x_j - b \geq L(1-d)$
$(\mathcal{R}_{8.4})$: $\mathcal{R}[d > 0 \rightarrow \sum_{j=1}^n a_j x_j > b] = \sum_{j=1}^n a_j x_j - b > (L-1)(1-d)$
$(\mathcal{R}_{8.5})$: $\mathcal{R}[d > 0 \rightarrow \sum_{j=1}^n a_j x_j = b] = \{\mathcal{R}[d > 0 \rightarrow \sum_{j=1}^n a_j x_j \geq b], \mathcal{R}[d > 0 \rightarrow \sum_{j=1}^n a_j x_j \leq b]\}$
where $\overline{d}$ is an abbreviation of $d_1, ..., d_n$.

Figure 4.1: Rule $\mathcal{R}$

In order to translate a conditional Γ-formula that contains constraint of the form $\sum_{j=1}^n a_j x_j \Theta b$, where Θ is one of the predicate symbols $\geq$, $\leq$, $>$ and $<$, into an IP-formula, we use the upper and lower bounds U and L of $\sum_{j=1}^n a_j x_j - b$ for given ranges of $x_j, j = 1, ..., n$. With L and U, we translate the conditional constraints into IP-formulas (see $\mathcal{R}_{8.1}$, $\mathcal{R}_{8.2}$, $\mathcal{R}_{8.3}$ and $\mathcal{R}_{8.4}$). For the conditional Γ-formula that contains constraint $\sum_{j=1}^n a_j x_j = b$, we replace the constraint $\sum_{j=1}^n a_j x_j = b$ with an equivalent form $\sum_{j=1}^n a_j x_j \geq b \wedge \sum_{j=1}^n a_j x_j \leq b$ and translate it into two conditional Γ-formulas (see $\mathcal{R}_{8.5}$).

After the above transformation, we compute the bounds U and L for each $\sum_{j=1}^{n} a_j x_j - b$. These bounds are computed using the bounds of $x_j, j = 1, ..., n$ that are usually specified separately or inferred from characteristics of the problems. We compute U and L as follows.

Let $l_j \leq x_j \leq u_j$ $(j = 1, ..., n)$,

$$U = \sum_{i \in \alpha} a_i u_i + \sum_{i \in \beta} a_i l_i - b,$$

$$L = \sum_{i \in \alpha} a_i l_i + \sum_{i \in \beta} a_i u_i - b,$$

where $\alpha = \{i \mid a_i > 0\}$, $\beta = \{i \mid a_i < 0\}$ and
$\alpha \cup \beta = \{1, ..., n\}$.

Bounds l_j and u_j for x_j, $j = 1, ..., n$, are natural numbers and u_j can be allowed to be infinite in this computation. With the computation, we eliminate some redundant IP-formulas by checking the value of U or L. For example, consider the following rule:

$(\mathcal{R}_{8.1})$: $\mathcal{R}[d > 0 \rightarrow \sum_{j=1}^{n} a_j x_j \leq b] =$
$\sum_{j=1}^{n} a_j x_j - b \leq U(1 - d)$.

The idea behind $(\mathcal{R}_{8.1})$ is that we use

$$\sum_{j=1}^{n} a_j x_j - b \leq U$$

as an expression whose value is always true.
If $\neg(U > 0)$,

$$\mathcal{M}[\![\sum_{j=1}^{n} a_j x_j - b \leq 0]\!]_\phi = \mathrm{T}$$

for any valuation ϕ. Hence, IP formula $\sum_{j=1}^{n} a_j x_j - b \leq U(1 - d)$ is redundant in this case and will not be generated. Otherwise we generate this formula.

Some IP-solvers such as CPLEX [7] assume all constraint boundaries are closed. In such a case, we remove strict inequalities $>$ and $<$ before giving the IP-formulas to these IP-solvers. It can be done easily by converting $>$ and $<$ into $\geq$ and $\leq$ respectively using the following equivalence:

$\mathcal{M}[\![\sum_{j=1}^{n} a_j x_j > b]\!]_\phi = \mathcal{M}[\![\sum_{j=1}^{n} a_j x_j \geq b + 1]\!]_\phi$, and
$\mathcal{M}[\![\sum_{j=1}^{n} a_j x_j < b]\!]_\phi = \mathcal{M}[\![\sum_{j=1}^{n} a_j x_j \leq b - 1]\!]_\phi$

for any valuation ϕ.

The output of rule $\mathcal{R}$ is a set of IP-formulas. A set of IP-formulas $\{C_1, \ldots, C_n\}$ represents a conjunction of formulas $C_1 \wedge \cdots \wedge C_n$. Due to the associativity of $\wedge$, we can flatten a set of formulas. For example, we can flatten a nested set $\{q, \{C_1 \ldots, C_n\}, \{D_1, \ldots, D_m\}\}$ to a logically equivalent set $\{q, C_1 \ldots, C_n, D_1, \ldots, D_m\}$. This allows us to use nested sets in Figure 1 to represents IP-formulas. We will, however, assume that nested sets of IP-formulas are flattened implicitly whenever necessary.

The output of rule $\mathcal{R}$ may be an existentially quantified IP-formula. The quantification is dropped in IP-formulas. The original logical specification is implicitly existentially quantified formulas, and hence resulting IP-formulas are implicitly existentially quantified. Note, however, all the transformation rules are applicable to universally quantified formulas.

We have the following soundness result of rule $\mathcal{R}$.

$(\mathcal{R}_9)$: $\mathcal{R}[\Gamma_n\{P_1,...,P_n\}] = \{\mathcal{R}[P_1],...,\mathcal{R}[P_n]\}$

$(\mathcal{R}_{10})$: $\mathcal{R}[d > 0 \rightarrow \Gamma_n\{P_1,...,P_n\}] = \{\mathcal{R}[d > 0 \rightarrow P_1],...,\mathcal{R}[d > 0 \rightarrow P_n]\}$

$(\mathcal{R}_{11})$: $\mathcal{R}[\Gamma_m\{P_1,...,P_r,p_1,...,p_s,\neg p_{s+1},...,\neg p_t\}] =$
$\exists \overline{d}\{\mathcal{R}[d_1 > 0 \rightarrow P_1],...,\mathcal{R}[d_r > 0 \rightarrow P_r], \sum_{i=1}^{r} d_i + \sum_{i=1}^{s} \delta_{p_i} + \sum_{i=s+1}^{t} (1 - \delta_{p_i}) \geq m\}$

$(\mathcal{R}_{12})$: $\mathcal{R}[d > 0 \rightarrow \Gamma_m\{P_1,...,P_r,p_1,...,p_s,\neg p_{s+1},...,\neg p_t\})] =$
$\exists \overline{d}\{\mathcal{R}[d_1 > 0 \rightarrow P_1],...,\mathcal{R}[d_r > 0 \rightarrow P_r], \sum_{i=1}^{r} d_i + \sum_{i=1}^{s} \delta_{p_i} + \sum_{i=s+1}^{t} (1 - \delta_{p_i}) \geq md\}$

where P_i $(i = 1,...,r)$ is a Γ-formula and p_i $(i = 1,...,t)$ is a propositional variable.

Figure 4.2: Optimising Rule $\mathcal{R}_o$

Theorem 4.1 Rule $\mathcal{R}$ is $\mathcal{M}$-correct.

Proof: The proof is given in appendix A. □

We give the optimising rules of $\mathcal{R}$ in Figure 2. It is easy to prove that the optimising rules are correct in the same way as in the proof of Theorem 4.1. By the rules $\mathcal{R}$ and $\mathcal{R}_o$, we can translate all Γ-formulas into IP-formulas.

Example 4.3 We transform the Γ-formula in Example 4.1 to an IP-formula.

(1) $\Gamma_1\{\neg p_1, \neg p_2, p_3, p_4\}$

Applying $(\mathcal{R}_{11})$ to (1), we obtain:
$(1 - \delta_{p_1}) + (1 - \delta_{p_2}) + \delta_{p_3} + \delta_{p_4} \geq 1$, that is, $\delta_{p_1} + \delta_{p_2} - \delta_{p_3} - \delta_{p_4} \leq 1$

Example 4.4 We transform the Γ-formula in Example 4.2 to the IP-formula.

$\Gamma_2\{\Gamma_1\{\Gamma_2\{x_1 > 2, x_2 > 3\}, x_3 \leq 8, x_4 \geq 7\},$
$\Gamma_1\{\Gamma_2\{x_3 > 8, x_4 < 7\}, x_1 \leq 2, x_2 \leq 3\}\}$

By $(\mathcal{R}_9)$, we have
(1) $\Gamma_1\{\Gamma_2\{x_1 > 2, x_2 > 3\},\ x_3 \leq 8,\ x_4 \geq 7\}$,
(2) $\Gamma_1\{\Gamma_2\{x_3 > 8, x_4 < 7\},\ x_1 \leq 2,\ x_2 \leq 3\}$.
$(\mathcal{R}_4)$ transforms (1) to give
(1.1) $d_1 > 0 \rightarrow \Gamma_2\{x_1 > 2, x_2 > 3\}$,
(1.2) $d_2 > 0 \rightarrow x_3 \leq 8,\ \ d_3 > 0 \rightarrow x_4 \geq 7$,
(1.3) $d_1 + d_2 + d_3 \geq 1$.
$(\mathcal{R}_{10})$ transforms (1.1) to give
(1.1.1) $d_1 > 0 \rightarrow x_1 > 2,\ \ d_1 > 0 \rightarrow x_2 > 3$.

Let U_{x-b} and L_{x-b} represent the upper bound of $x - b$ and the lower bound of $x - b$ respectively. By $(\mathcal{R}_{8.4})$ we transform (1.1.1) into the following IP-formulas:

(1.1.1.1) $\{x_1 - 2 > (L_{x_1-2} - 1)(1 - d_1) : L_{x_1-2} \leq 0\}$,
$\{x_2 - 3 > (L_{x_2-3} - 1)(1 - d_1) : L_{x_2-3} \leq 0\}$.

$(\mathcal{R}_{8.1})$ and $(\mathcal{R}_{8.3})$ transform (1.2) to give

(1.2.1) $\{x_3 - 8 \leq U_{x_3-8}(1 - d_2) : U_{x_3-8} > 0\}$,
$\{x_4 - 7 \geq L_{x_4-7}(1 - d_3) : L_{x_4-7} < 0\}$.

($\mathcal{R}_4$) transforms (2) to give
(2.1) $d_4 > 0 \rightarrow \Gamma_2\{x_3 > 8, x_4 < 7\}$,
(2.2) $d_5 > 0 \rightarrow x_1 \leq 2,\ \ d_6 > 0 \rightarrow x_2 \leq 3$,
(2.3) $d_4 + d_5 + d_6 \geq 1$.
($\mathcal{R}_{10}$) transforms (2.1) to give
(2.1.1) $d_4 > 0 \rightarrow x_3 > 8,\ \ d_4 > 0 \rightarrow x_4 < 7$.
($\mathcal{R}_{8.4}$) and ($\mathcal{R}_{8.2}$) transform (2.1.1) to give

$$(2.1.1.1)\quad \{x_3 - 8 > (L_{x_3-8} - 1)(1 - d_4) : L_{x_3-8} \leq 0\},$$
$$\{x_4 - 7 < (U_{x_4-7} + 1)(1 - d_4) : U_{x_4-7} \geq 0\}.$$

Two applications of ($\mathcal{R}_{8.1}$) transform (2.2) to give

$$(2.2.1)\quad \{x_1 - 2 \leq U_{x_1-1}(1 - d_5) : U_{x_1-1} > 0\},$$
$$\{x_2 - 3 \leq U_{x_2-3}(1 - d_6) : U_{x_2-3} > 0\}.$$

Let $1 \leq x_i \leq 10$ $(i = 1, .., 4)$, we compute U_{x_i-b} and L_{x_i-b} and get the following IP-formulas:
$x_1 - 2d_1 > 0$,
$x_2 - 3d_1 > 0$,
$x_3 + 2d_2 \leq 10$,
$x_4 - 6d_3 \geq 1$,
$d_1 + d_2 + d_3 \geq 1$,
$x_3 - 8d_4 > 0$,
$x_4 + 4d_4 < 11$,
$x_1 + 8d_5 \leq 10$,
$x_2 + 7d_6 \leq 10$,
$d_4 + d_5 + d_6 \geq 1$.

5 Implementation in Mathematica

We have implemented the transformation discussed in this paper in Mathematica 3.0. The implemented system, to be called `TIP` (Transformation of Integer Programming) is used as a front-end to existing solvers. By using Mathlink and CGI, we connect `TIP` to (possibly remote) IP-solvers such as CPLEX [7] or solvers that we have developed [8][9] via Web page. When `TIP` is called in Web page, with $\mathcal{L}^+$-formulas, `TIP` will generate IP-formulas, which then are sent to the solvers together with an objective function via Mathlink and CGI. The answer, an optimal solution for the IP problem, is sent back to the Web page. `TIP` consists of the functions `GenIP[spec_,decl_,opts___]`, `RRC[spec_]`, `RuleE[spec_]`, `RuleT[spec_]`, `RuleN[spec_]`, `RuleF[spec_]` and `RuleR[spec_]`. `RRC[spec_]` (Remove Redundant Constraints) checks whether a given constraint is redundant and outputs empty set {} if redundant or else a simplified constraint after substituting the value of the bounds. `RuleE[spec_]` to `RuleR[spec_]` define the transformation rules $\mathcal{E}$ to $\mathcal{R}$ respectively. `GenIP[spec_,decl_,opts___]` is the main function which is called with a logical specification `spec`, a list of propositional variables and integer variables with their bounds `decl` and an option `opts`. By option `opts`, users specify whether the generated IP-formulas will be simplified or not with `IPFormSimplify→True` (default) or `IPFormSimplify→False`. `TIP` first checks each variable in the logic specification `spec` whether it is declared in `decl` and outputs an error message if some variables are not declared or else outputs the generated IP-formulas. We give the whole program of `TIP` in appendix B.

Example 5.1 We buy nine types of stocks numbered by 1 to 9. If three or more types of stocks {1 to 5} are bought, or less than four types of stocks {3 to 6,8,9} are bought then at most two types of stocks {6 to 9} will be bought unless none of stocks {5 to 7} are bought. Question: Which stocks can we buy to get as many types as possible?

We can formulate the above problem in $\mathcal{L}^+$ by using propositional variable s_i for $i = 1, ..., 9$ to represent i-th type of the stock. $s_i = \mathrm{T}$ means we buy the i-th type of the stock and $s_i = \mathrm{F}$ means we do not buy the i-th type of the stock. The constraint of the problem can be formulated as follows.

$$\begin{array}{l}(\mathrm{at\_least}(3, \{s_1, s_2, s_3, s_4, s_5\}) \vee \\ \ \mathrm{at\_most}(3, \{s_3, s_4, s_5, s_6, s_8, s_9\})) \wedge \\ \ \neg\ \mathrm{none}(s_5, s_6, s_7) \rightarrow \mathrm{at\_most}(2, \{s_6, s_7, s_8, s_9\})\end{array}$$

We transform the $\mathcal{L}^+$-formula into the IP-formulas by `TIP`.

$$\begin{array}{l}\texttt{GenIP[(atleast[3,}\{s_1, s_2, s_3, s_4, s_5\}\texttt{]} \vee \\ \qquad \texttt{atmost[3,}\{s_3, s_4, s_5, s_6, s_8, s_9\}\texttt{])} \wedge \\ \qquad \neg\ \texttt{none[}\{s_5, s_6, s_7\}\texttt{]} \\ \qquad \rightarrow\ \texttt{atmost[2,}\{s_6, s_7, s_8, s_9\}\texttt{]}, \\ \qquad \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}\texttt{]}\end{array}$$

The IP-formulas generated are:

$$\begin{array}{l}d_1 + d_2 + d_3 \geq 1, \\ 1 - \delta_{s_5} \geq d_2, \\ 1 - \delta_{s_6} \geq d_2, \\ 1 - \delta_{s_7} \geq d_2, \\ 4 - \delta_{s_6} - \delta_{s_7} - \delta_{s_8} - \delta_{s_9} \geq 2d_1, \\ 5 - \delta_{s_1} - \delta_{s_2} - \delta_{s_3} - \delta_{s_4} - \delta_{s_5} \geq 3d_3, \\ \delta_{s_3} + \delta_{s_4} + \delta_{s_5} + \delta_{s_6} + \delta_{s_8} + \delta_{s_9} \geq 4d_3, \\ 0 \leq d_1 \leq 1, 0 \leq d_2 \leq 1, 0 \leq d_3 \leq 1, \\ 0 \leq \delta_{s_1} \leq 1, 0 \leq \delta_{s_2} \leq 1, 0 \leq \delta_{s_3} \leq 1, \\ 0 \leq \delta_{s_4} \leq 1, 0 \leq \delta_{s_5} \leq 1, 0 \leq \delta_{s_6} \leq 1, \\ 0 \leq \delta_{s_7} \leq 1, 0 \leq \delta_{s_8} \leq 1, 0 \leq \delta_{s_9} \leq 1.\end{array}$$

Then we send the above IP-formulas and an objective function $\texttt{Max}\ \sum_{i=1}^{9} \delta_{s_i}$ to an IP-solver and finally get the optimal solution:

objective=7,
$\delta_{s_1} = 1$, $\delta_{s_2} = 1$, $\delta_{s_3} = 1$, $\delta_{s_4} = 1$, $\delta_{s_5} = 1$, $\delta_{s_6} = 1$,
$\delta_{s_7} = 1$, $\delta_{s_8} = 0$, $\delta_{s_9} = 0$, $d_1 = 1$, $d_2 = 0$, $d_3 = 0$.

6 Conclusion and Future work

We have defined a logical modelling language $\mathcal{L}^+$ for IP problems and presented a systematic algorithm for translating the logical formulas into IP formulas with rigorous proofs of the soundness of the transformation. We have implemented the transformation on Mathematica

3.0. We will continue this research aiming to provide a practical system for multi-objective decision with uncertainty. In particular, we will refine and enhance the logical specification system by introducing constructs for specifying uncertainty. The transformation algorithm will be further developed to support dealing with stochastic variables.

References

[1] K.I.M.McKinnon, H.P.Williams, *Constructing Integer Programming Models by The Predicate Calculus.* Annals of Operations Research, 21(1989) 227–246.

[2] E.Hadiconstantinou, G.Mitra, *A linear and discrete programming framework for representing qualitative knowledge.* Journal of Economic Dynamics and Control 18(1994) 273–297. North-Holland.

[3] R.Raman, I.E.Grossmann, *Relation between MILP modelling and logical inference for process synthesis.* Computers and chemical engineering, 15(2):73–84,1991.

[4] A.L.Brearley, G.Mitra, H.P.Williams, *Analysis of mathematical programming problems prior to applying the simplex algorithm.* Mathematical programming 8(1975) 54–83. North-Holland.

[5] S.Wolfram, *The Mathematica Book.* Wolfram Media Inc. Champaing, Illinois, USA, 1996.

[6] Jean H.Gallier, *Logic for Computer Science: Foundations of Automatic Theorem Proving.* John Wiley & Sons, Inc. New York, USA, 1987.

[7] *Using the CPLEX Callable Library.* CPLEX Optimization, Inc. USA, 1995.

[8] Q.Li, Y.Guo, T.Ida, J.Darlington, *The Minimised Geometric Buchberger Algorithm: An Optimal Algebraic Algorithm for Integer Programming.* Proceedings of the International Symposium on Symbolic and Algebraic Computation , Maui, Hawaii U.S.A. pp. 331–338, 1997.

[9] Q.Li, Y.Guo, T.Ida, *A Parallel Algebraic Approach Towards Integer Programming.* Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems, Washington,D.C.,U.S.A. pp. 59–64, 1997.

[10] Q.Li, F.Janssen, Z.Yang, T.Ida, *ILIN: An Implementation of the Integer Labeling Algorithm for Integer Programming.* IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences. Vol. E81-A No.2 pp. 304–309, 1998.

Appendix A.

In this appendix we give a proof of Theorem 4.1.

Theorem A.1. Rule $\mathcal{R}$ is $\mathcal{M}$-correct.

Proof: We prove $\mathcal{M}[\![\mathcal{R}[P]]\!] = \mathcal{M}[\![P]\!]$ by induction on the number of Γ symbols occurring in a Γ-formula P. We prove rules $(\mathcal{R}_5) \sim (\mathcal{R}_{8.1})$. Rules $(\mathcal{R}_1)$, $(\mathcal{R}_2)$ and $(\mathcal{R}_4)$ can be proved as special cases of rules $(\mathcal{R}_5) \sim (\mathcal{R}_7)$, respectively. Rules $(\mathcal{R}_{8.2}) \sim (\mathcal{R}_{8.5})$ are proved similarly to rule $(\mathcal{R}_{8.1})$.

Let P be a formula of the form $d > 0 \rightarrow Q$. We distinguish the following cases depending on the structure of Q.

(1) Q is an equation of the form p or $\neg p$. We take an arbitrary but fixed valuation ϕ. By easy case analysis of the values of $\mathcal{M}[\![d > 0]\!]_\phi$ and $\mathcal{M}[\![p]\!]_\phi$ with definition of δ_p, we have

$$\mathcal{M}[\![d > 0 \rightarrow p]\!]_\phi = \mathcal{M}[\![\delta_p \geq d]\!]_\phi$$
$$\mathcal{M}[\![d > 0 \rightarrow \neg p]\!]_\phi = \mathcal{M}[\![1 - \delta_p \geq d]\!]_\phi.$$

(2) Q is a Γ-formula.
We prove that

$$\begin{aligned}&\mathcal{M}[\![d > 0 \rightarrow \Gamma_m \{P_1, \ldots, P_n\}]\!]_\phi = \\ &\quad \mathcal{M}[\![\exists \overline{d}\{d_1 + \ldots + d_n \geq md, \\ &\qquad \mathcal{R}[d_1 > 0 \rightarrow P_1], \ldots, \mathcal{R}[d_n > 0 \rightarrow P_n]\}]\!]_\phi \end{aligned} \qquad (A.1)$$

for an arbitrary but fixed valuation ϕ.

We further distinguish two cases depending on the values of $\phi(d)$. Since d is a decision variable, d is assigned the value of either 1 or 0. Suppose $\phi(d) = 0$. Then $\mathcal{M}[\![d > 0]\!]_\phi =$ F, it is easy to see that by taking $\phi(d_1) = 0, ..., \phi(d_n) = 0$, both side of Eq.(A.1) reduce to T.

Next consider the case that $\phi(d) = 1$. Let $S_1, \ldots, S_k$ be all the m-subsets of $\{P_1, \ldots, P_n\}$, where $k =_n C_m$. Suppose that the left-hand side of Eq. (A.1)= T, *i.e.* $\mathcal{M}[\![d > 0 \rightarrow S_1 \vee \ldots \vee S_k]\!]_\phi =$ T. This implies that there exists a set $S_j = \{P_{j_1}, \ldots, P_{j_m}\} \in \{S_1, \ldots, S_k\}$ such that $\mathcal{M}[\![P_{j_1}]\!]_\phi =$ T, $\ldots, \mathcal{M}[\![P_{j_m}]\!]_\phi =$ T. We define a new valuation ϕ' that modifies ϕ as follows.

$$\begin{aligned}&\phi'(x) = \phi(x), \quad \text{if } x \notin \{d_1, \ldots, d_n\} \\ &\phi'(d_i) = 1, \quad \text{if } i \in \{j_1, \ldots j_m\} \\ &\phi'(d_i) = 0, \quad \text{if } i \in \{1, \ldots, n\} - \{j_1, \ldots j_m\}\end{aligned}$$

We have to prove

$$\begin{aligned}&\mathcal{M}[\![d > 0 \rightarrow \Gamma_m \{P_1, \ldots, P_n\}]\!]_{\phi'} = \\ &\quad \mathcal{M}[\![\{d_1 + \ldots + d_n \geq md, \\ &\qquad \mathcal{R}[d_1 > 0 \rightarrow P_1], \ldots, \mathcal{R}[d_n > 0 \rightarrow P_n]\}]\!]_{\phi'}\end{aligned}$$

With this valuation ϕ', we have

$$\begin{aligned}&\mathcal{M}[\![d_1 + \ldots + d_n \geq md]\!]_{\phi'} = \text{T}, \text{since } \phi'(d) = 1 \\ &\mathcal{M}[\![d_i > 0 \rightarrow P_i]\!]_{\phi'} = \text{T}, \text{ for } i \in \{j_1, \ldots j_m\} \\ &\mathcal{M}[\![d_i > 0 \rightarrow P_i]\!]_{\phi'} = \text{T}, \\ &\qquad \text{for } i \in \{1, \ldots, n\} - \{j_1, \ldots j_m\}.\end{aligned}$$

By induction hypothesis, we have

$$\mathcal{M}[\![\mathcal{R}[d_i > 0 \rightarrow P_i]]\!]_\rho = \mathcal{M}[\![d_i > 0 \rightarrow P_i]\!]_\rho,$$

for arbitrary (but associated with $\mathcal{M}$) valuation ρ.
In particular,

$$\mathcal{M}[\![\mathcal{R}[d_i > 0 \rightarrow P_i]]\!]_{\phi'} = \mathcal{M}[\![d_i > 0 \rightarrow P_i]\!]_{\phi'} = \text{T}.$$

Hence, Eq. (A.1) holds.

Suppose next that the left-hand side of Eq.(A.1) is F, *i.e.* $\mathcal{M}[\![d > 0 \rightarrow S_1 \vee \ldots \vee S_k]\!]_\phi = \mathrm{F}$. Then for all S_j $(j = 1, ..., k)$, $\mathcal{M}[\![S_j]\!]_\phi = \mathrm{F}$. This implies that there exists $i \in \{1, \ldots, n\}$ such that $\mathcal{M}[\![P_i]\!]_\phi = \mathrm{F}$. For this case, we define a valuation ϕ' as follows:

$$\phi'(d_i) = 1$$
$$\phi'(x) = \phi(x), \quad \text{otherwise}$$

Then $\mathcal{M}[\![d_i > 0 \rightarrow P_i]\!]_{\phi'} = \mathrm{F}$. The same reasoning is applied to this case as in the case of T and we have $\mathcal{M}[\![\mathcal{R}[d_i > 0 \rightarrow P_i]]\!]_{\phi'} = \mathrm{F}$. Hence, the right-hand side of Eq.(A.1) is F.

(3) Q is a constraint.
We only prove that

$$\mathcal{M}[\![d > 0 \rightarrow \textstyle\sum_{j=1}^{n} a_j x_j \leq b]\!]_\phi =$$
$$\mathcal{M}[\![\textstyle\sum_{j=1}^{n} a_j x_j - b \leq U(1-d)]\!]_\phi \qquad (A.2)$$

We distinguish two cases depending on the value assignment of d.

Case $\phi(d) = 0$.
The left-hand side of Eq. (A.2) is equal T. The right-hand side is equal to $\mathcal{M}[\![\sum_{j=1}^{n} a_j x_j - b \leq U]\!]_\phi$, and hence is equal to T by definition of U.

Case $\phi(d) = 1$.
If $U > 0$, both sides of Eq.(A.2) are equal to $\mathcal{M}[\![\sum_{j=1}^{n} a_j x_j \leq b]\!]_\phi$. If $U \leq 0$, the right-hand side of Eq.$(A.2)$ is equal to $\mathcal{M}[\![\sum_{j=1}^{n} a_j x_j - b \leq U]\!]_\phi$, and the left-hand side of Eq.$(A.2)$ is equal to $\mathcal{M}[\![\sum_{j=1}^{n} a_j x_j - b \leq 0]\!]_\phi$. Since U is the upper bound of $\sum_{j=1}^{n} a_j x_j - b$, both are equal to T.

□

Appendix B.

■ This appendix gives all the program of TIP.

Function **ΓFormQ**[x] checks whether an expression is Γ−form .

```
ΓFormQ[Γ[_, _]] := True;
ΓFormQ[_] := False;
```

Function **VarQ**[x] checks whether x is a propositional variable or an integer variable.

```
VarQ[x_Symbol] /; x =!= True ∧ x =!= False := True;
VarQ[_] := False;
```

Function **NegPVarQ**[x] checks whether x is a negated propositional variable.

```
NegPVarQ[¬ x_Symbol] /; x =!= True ∧ x =!= False := True;
NegPVarQ[_] := False;
```

Function **PLiteralQ**[p] checks whether p is a propositional variable or a negated propositional variable

```
PLiteralQ[p_] := VarQ[p] ∨ NegPVarQ[p];
NotPLiteralQ[p_] := ¬ PLiteralQ[p]
```

Function **BoundedQ**[x] checks whether a variable x is properly bounded.

```
BoundedQ[m_Integer ≤ x_?VarQ ≤ n_Integer] := True;
BoundedQ[_] := False;
```

Function **ConstraintQ**[c] checks whether an expression c is a constraint . Note that we use == to denote the equality of symbol of L^+.

```
ConstraintQ[x_ ≥ y_ | x_ > y_ | x_ ≤ y_ | x_ < y_ | x_ == y_ ] := True;
ConstraintQ[_] := False;
```

Function **PredOrder**[x,y] defines a predicate order in a Γ−formula.

```
PredOrder[x_?VarQ, r_[_, _]] := True;
PredOrder[Not[_], r_[_, _]] := True;
PredOrder[x_, y_] := False
```

Function **PLiteral2IPvar**[e] translates a propositional variable e= **p** and a formula e=**¬p** into IP−formulas.

```
PLiteral2IPvar[x_?VarQ] := Bounds⟦Position[Bounds, x]⟦1, 1⟧, 2⟧ ;
PLiteral2IPvar[¬ x_?VarQ] := 1 - Bounds⟦Position[Bounds, x]⟦1, 1⟧, 2⟧ ;
```

Function **Raux**[x,y,z] is an auxiliary function of **RuleR**[x].

```
Raux[{x___?PLiteralQ, y___ ?NotPLiteralQ}, m_, n_] :=
 Module[{newvarlist = Map[Unique[d]&, {y}]}, Bounds = Bounds⋃Map[0 ≤ # ≤ 1&, newvarlist];
  Prepend[Map[RuleR, MapThread[Function[{d, a}, d > 0 ⇒ a], {newvarlist, {y}}]],
   (Apply[Plus, newvarlist⋃Map[PLiteral2IPvar, {x}]] ≥ m n)]];
```

Function **RuleE**[x] defines rule E.

```
RuleE[x_] := x //. {p_ ⇒ q_ -> ¬ p ∨ q, p_ ⇔ q_ -> (p ⇒ q) ∧ (q ⇒ p)};
```

Function **RuleT**[x] defines rule T.

```
RuleT[x1_ ∨ x2_] := Γ[1, {RuleT[x1], RuleT[x2]}];
RuleT[x1_ ∧ x2_] := Γ[2, {RuleT[x1], RuleT[x2]}];
RuleT[¬ x_] := ¬ RuleT[x] ;
RuleT[atleast[m_, S_]] := Γ[m, Map[RuleT, S]];
RuleT[atmost[m_, S_]] := Γ[Length[S] - m, Map[RuleT, Map[Not, S]]];
RuleT[none[S_]] := Γ[Length[S], Map[RuleT, Map[Not, S]]];
RuleT[x_?VarQ] := PVarDecl[x];
RuleT[x_?ConstraintQ] := IPVarDecl[x];
RuleT[True] := Throw[{True, Bounds}];
RuleT[False] := Throw["Wrong specification"]
```

Function **RuleN** [x] defines rule N.

```
RuleN[Γ[m_, S_]] := Γ[m, Map[RuleN, S]];
RuleN[¬ Γ[m_, S_]] := Γ[Length[S] - m + 1, Map[RuleN, Map[Not, S]]];
RuleN[¬ (x1_ == x2_)] := Γ[1, {x1 > x2, x1 < x2}];
RuleN[x_] := x;
```

Function **RuleF**[x] defines rule F.

```
RuleF[x_] := x //. {RF1, RF2}
RF1 = Γ [1, {x1___, Γ[1, {x2___}], x3___}] :> Γ[1, {x1, x2, x3}];
RF2 = Γ[m_, {x1___, Γ[n_, S_], x2___}] /; m == Length[{x1, x2}] + 1 ∧ n == Length[S] :>
  Γ[n + m - 1, S ⋃ {x1, x2}];
```

Function **RuleR**[x] defines rule R.

```
RuleR[r_[x1_, x2_]] := r[x1, x2];
RuleR[p_?PLiteralQ] := PLiteral2IPvar[p] ⩾ 1;
RuleR[Γ[m_, S_]] /; m == Length[S] := Map[RuleR, S];
RuleR[Γ[m_, S_]] /; m < Length[S] := Raux[Sort[S, PredOrder], m, 1];
RuleR[d_ > 0 ⇒ Γ[m_, S_]] /; m == Length[S] := Map[RuleR, Map[(d > 0 ⇒ #)&, S]];
RuleR[d_ > 0 ⇒ Γ[m_, S_]] /; m < Length[S] := Raux[Sort[S, PredOrder], m, d];
RuleR[d_ > 0 ⇒ p_?PLiteralQ] := PLiteral2IPvar[p] ⩾ d;
RuleR[d_ > 0 ⇒ x1_ ⩾ x2_] := Module[{L}, (x1 - x2 ⩾ L (1 - d))_L];
RuleR[d_ > 0 ⇒ x1_ > x2_] := Module[{L}, (x1 - x2 > (L - 1) (1 - d))_L];
RuleR[d_ > 0 ⇒ x1_ ⩽ x2_] := Module[{U}, (x1 - x2 ⩽ U (1 - d))^U];
RuleR[d_ > 0 ⇒ x1_ < x2_] := Module[{U}, (x1 - x2 < (U + 1) (1 - d))^U];
RuleR[d_ > 0 ⇒ x1_ == x2_] := {RuleR[d > 0 ⇒ x1 ⩾ x2], RuleR[d > 0 ⇒ x1 ⩽ x2]};
```

Function **Lowerbound**[A] computes the lowerbound of constraint A with respects to the given Bounds, where Bounds is specifed as a list of terms of the form a ≤x≤b. **Lowerbound** uses auxiliary function **Lb**.

```
Lowerbound[A_?AtomQ ] := Lb[A ];
Lowerbound[A_ ] := Map[Lb, A];
Lb[term : a_ * x_ ] /; a > 0 :=
 Module[{i = Position[Bounds, _ ≤ x ≤ _][[1, 1]]}, term /. {Bounds[[i, 2]] → Bounds[[i, 1]]}];
Lb[term : a_ * x_ ] /; a < 0 :=
 Module[{i = Position[Bounds, _ ≤ x ≤ _][[1, 1]]}, term /. {Bounds[[i, 2]] → Bounds[[i, 3]]}];
Lb[x_?VarQ ] := Bounds[[Position[Bounds, _ ≤ x ≤ _][[1, 1]], 1]]
Lb[a_ ] := a
```

Likewise function **Upperbound** is defined.

```
Upperbound[A_] := Map[Ub, A];
Upperbound[A_?AtomQ] := Ub[A ];
Ub[term : a_ * x_ ] /; a > 0 :=
 Module[{i = Position[Bounds, _ ≤ x ≤ _][[1, 1]]}, term /. {Bounds[[i, 2]] → Bounds[[i, 3]]}];
Ub[term : a_ * x_ ] /; a < 0 :=
 Module[{i = Position[Bounds, _ ≤ x ≤ _][[1, 1]]}, term /. {Bounds[[i, 2]] → Bounds[[i, 1]]}];
Ub[x_?VarQ ] := Bounds[[Position[Bounds, _ ≤ x ≤ _][[1, 1]], 3]];
Ub[a_ ] := a;
```

Function **RRC**[x] (Remove Redundant Constraints) is defined below using the following auxiliary functions: **RCaux** checks whether a given constraint is redundant and outputs {} if redundant or else a simplified constraint after substituting the value of the Bounds.

```
RRC[A_] := Map[RCaux, A];
RCaux[ (A_ ≤ B_)U_ ] := Module[{Uw = Upperbound[A]},
  If[Uw > 0, A ≤ (B /. {U → Uw}), {}]];
RCaux[(A_ < B_) U_] :=
 Module[{Uw = Upperbound[A]},
  If[Uw ≥ 0, A < (B /. {U → Uw}), {}]];
RCaux[ (A_ ≥ B_) L_ ] := Module[{Lw = Lowerbound[A]},
  If[Lw < 0, A ≥ (B /. {L → Lw}), {}]];
RCaux[(A_ > B_) L_] :=
 Module[{Lw = Lowerbound[A]},
  If[Lw ≤ 0, A > (B /. {L → Lw}), {}]];
RCaux[A_] := A
```

Function **MakeBounds**[x] checks whether the input bound of each integer variable is right and generates a decision variable and its bound for each propositional variable.

```
MakeBounds[ x_?BoundedQ ] := x;
MakeBounds[x_?VarQ] := 0 ≤ Pvarx ≤ 1;
```

Founction **PVarDecl**[e] checks whether each propositional variable is properly declared.

```
PVarDecl[v_] := If[MemberQ[Bounds, v, {2, 3}],
  v, Print["Propositional Variable ",  v, " not declared."];
  Throw[Bounds]];
```

Founction **IPVarDecl**[e] checks whether each integer variable is properly declared.

```
IPVarDecl[v_?VarQ] :=
 If[MemberQ[Bounds, v, 2], v, Print["Integer variable ",  v, " not declared."];
  Throw[Bounds]];
IPVarDecl[t : f_[__]] := Map[IPVarDecl, t];
IPVarDecl[x_Integer] := x;
IPVarDecl[x_] :=
 Module[{}, Print["Constraint ", x, " is not well formed"]; Throw[Bounds]]
```

Founction **IneqSimplify**[e] simplifies expressions.

```
IneqSimplify[a_ > b_] := Simplify[a - b] > 0;
IneqSimplify[a_ ≥ b_] := Simplify[a - b]  ≥ 0;
IneqSimplify[a_ < b_] := Simplify[a - b] < 0;
IneqSimplify[a_ ≤ b_] := Simplify[a - b]  ≤ 0;
```

Founction **Options**[GenIP] specifies that the generated IP−formulas will be simplified.

```
Options[GenIP] = {IPFormSimplify -> True}
```

This is the main function **GenIP**. **GenIP** needs three arguments **spec** , **decl** and **opts**. **spec** is a logical specification, **decl** is a list of propositional variables and the bounds of integer variables of the form a ≤x≤b, and **opts** is an option to specify whether the generated IP−formulas will be simplified or not with **IPFormSimplify−>True** or **IPFormSimplify−>False.**

```
GenIP[spec_, decl_, opts___] :=
 Catch[Block[{Pvar = Unique[δ] , Bounds = Map[MakeBounds, decl]} ,
  With[{ipform =
      Flatten[{Composition[RRC, Flatten, RuleR, RuleF, RuleN, RuleT, RuleE][spec]}]},
    If[IPFormSimplify /. {opts} /. Options[GenIP],
      Map[IneqSimplify, ipform], ipform, ipform] ⋃
     Bounds]]]
```