Computability issues in MINLP

Leo Liberti

LIX CNRS, École Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau, France
Email address: leo.liberti@polytechnique.edu

Contents

CHAPTER 1

Computability

In this chapter we look at whether MINLP can be solved. By "solving" a class of problems such as MINLP we mean whether there exists a single algorithm that solves every instance of every problem in the class MINLP. We prove that such an all-powerful algorithm does not exist.

1.1. Formal systems

Computation is a mechanical process, namely a process that does not require human intervention. This process is based on the manipulation of strings of characters, so we call it "syntactical". We represent such processes by means of mathematical entities known as "formal systems".

- A *formal system* (FS) S is defined by an alphabet, a grammar, some axioms, and some inference rules.
  - The **alphabet** is a finite or countable set of symbols.
  - The (formal) **grammar** encodes the rules for deciding whether a given string of alphabet symbols is a (syntactically) valid formula or sentence. The symbols in the left hand side of grammar rules are *nonterminal symbols*. The others are *terminal symbols*; the first nonterminal symbol is called *initial*.
  - The set A of **axioms** contains syntactically valid sentences that is assumed to hold (see below about the interpretation of "holding").
  - The set R of **inference rules** contains rules for constructing new sentences that hold from old ones that hold.
- The **theory** of a FS is the smallest set of sentences obtained by recursively applying R to A. By definition of inference rules, all sentences in a theory hold in the FS.

The verification process encoded in the grammar decides whether the given string is syntactically valid or invalid in the FS. If it is valid, it may be a formula or a sentence. A formula has at least one *free variable symbol*, i.e. a symbol that stands for something else, but which remains unspecified. Sentences have no free variable symbols. This distinction will become clearer later.

The grammar describes an algorithm that takes a string in input, and recursively matches its parts to the various nonterminal symbols in the grammar. It stops with YES (i.e. the string is valid) when all the parts have been matched to terminal symbols, otherwise it stops with NO. Yet unmatched parts are called *terms*.

EXAMPLE 1.1 (*How grammars work*). Let us consider the following **grammar** for a tiny fragment of English (the **alphabet** is the English alphabet). The "|" character in the right hand sides of the grammar rules stands for "or".

$$\begin{array}{rcl}
\mathsf{phrase} & \longrightarrow & \mathsf{subject\ verbal\ object} \\
\mathsf{subject} & \longrightarrow & \text{This} \mid \text{Computers} \mid \text{building} \mid \text{I} \\
\mathsf{verbal} & \longrightarrow & \mathsf{adverb\ verb} \mid \mathsf{verb} \\
\mathsf{adverb} & \longrightarrow & \text{never} \\
\mathsf{verb} & \longrightarrow & \text{is} \mid \text{run} \mid \text{am} \mid \text{tell} \\
\mathsf{object} & \longrightarrow & \text{the } \mathsf{noun} \mid \text{a } \mathsf{noun} \mid \mathsf{noun} \\
\mathsf{noun} & \longrightarrow & \text{university} \mid \text{world} \mid \text{cheese} \mid \text{lies}
\end{array}$$

Consider the string "This is a university". The grammar starts with the initial symbol phrase, and matches the whole string to it. Then it looks in the grammar for the definition of phrase, finding "subject verbal object". It then looks for the definition of subject, finding an alternative between "This", "Computers", "building", and "I", all of which are terminal symbols. Since "This" matches the first word in the string, "This" becomes a term of type subject. It then climbs back up in the recursion calls, and follows verbal, and so on. Eventually it maches all words, and verifies that the string is valid. In the process, it finds the terms "This" (subject), "is" (verbal and verb), "a university" (object), and "university" (noun). Similarly, this grammar verifies that the strings "a building is a university", and "Computers run a university" are also valid. □

A sentence *holds in the FS* when it is an axiom, or it is derived from axioms by a sequence of inference rules. When we say that "a sentence holds in the FS" we do not specify what status holds for the sentence. We leave this ambiguous because FSs are syntactical constructs, whereas specifying a status (e.g. "holds true") involves a semantic component. If you require a firmer grounding, I suggest you think of "hold" as "is provable", on the basis that FSs are syntactical constructs supposed to describe mechanical (e.g., computerized) proofs. If two sentences P, Q are identical, we write $P \equiv Q$.

Example 1.2 (*Sentences that hold and those that don't*). With reference to the grammar in Example 1.1, let us now say that "a building is a university" is an **axiom**, and none of the other sentences are axioms. Let us also say that the **inference rules** allow the replacement of the string "a building is" with "Computers run". Then one may never derive "This is a university" from axioms and inference rules, which. On the other hand, one may derive the sentence "Computers run a university". Therefore "This is a university" does not hold in this FS, but "Computers run a university" holds. □

Example 1.3 (*Sentences vs. formulæ*). With reference to the grammar in Example 1.1, "This is a university" contains a pronoun, "this", which acts like a variable symbol. Even outside of FSs, no human could say whether "This is a university" is true or false without knowing what "This" refers to. Back into the FS, pronouns may be resolved (for example by means of an inference rule that replaces "This" with some other string such as "a building") or not. In this case, the pronoun is not resolved, which means that this string is a valid formula, rather than a sentence. Unlike sentences, the property of "holding" does not apply to formulæ.

Example 1.4 (*Spelling out the grammar and the inference rules*). The FS A of alternating strings [**2**, Ch. I, §0].

- **Alphabet**: $\{a, b, \varnothing\}$.
- **Grammar**:

  $\mathsf{S} \longrightarrow \mathsf{S}\,a \mid \mathsf{S}\,b \mid \varnothing$

- **Axioms**:

$$a \tag{1.1}$$
$$b \tag{1.2}$$
$$a\,b \tag{1.3}$$
$$b\,a \tag{1.4}$$
$$\forall s \in \mathsf{S} \quad s\,a \quad \to \quad s\,a\,b \tag{1.5}$$
$$\forall s \in \mathsf{S} \quad s\,b \quad \to \quad s\,b\,a \tag{1.6}$$

- **Inference rules**: from two sentences P, $P \to Q$ that hold, the sentence Q holds (*modus ponens*).

Some remarks are in order.

(1) The terminal symbol $\varnothing$ is known as the "empty symbol".
(2) Consider the sentence $a\,b\,a$: is it valid? We use the grammar, starting with the rule $\mathsf{S} \to \mathsf{S}\,a$, which yields S to be the sentence $a\,b$. We then invoke the rule $\mathsf{S} \to \mathsf{S}\,b$, which yields S to be the sentence a. Next, we invoke $\mathsf{S} \to \mathsf{S}\,a$ again, yielding $\mathsf{S} = \varnothing$. The process ends having matched the whole given sentence, which means that the grammar accepts the sentence: the sentence is valid. Is $a\,a$ valid? By applying $\mathsf{S} \to \mathsf{S}\,a$ twice, we see that the sentence is valid. By contrast, $a\,b\,c$ is an invalid sentence.
(3) The last two axioms (1.5)-(1.6) do not just use alphabet characters, but also other characters. Each such axiom is not really an axiom but rather an *axiom schema*, i.e. a "template" describing a countable set of axioms. These schemata state: "for every sentence of nonterminal type S in this FS, every sentence that looks syntactically like "$s\,a \to s\,a\,b$" is an axiom". Schemata are allowed on the basis that any computer can recognize whether a given sentence matches an axiom schema. The symbol "$\to$" occurring in axioms simply replaces the construct "if... then..." (the symbol "$\longrightarrow$" occurring in formal grammars denotes instead the application of a parsing rule).
(4) Again about $\to$: $p \to q$ is equivalent to $(\neg p) \vee q$: the truth tables of these sentences are the same.
(5) We now discuss axioms and inference rules. Every sentence in the axiom list is assumed to hold. Is the sentence $a\,a$ a theorem of A? Not according to axioms (1.1)-(1.4). Can we find an axiom of the form $P \to Q$ that holds, where Q is $a\,a$? Neither of (1.5)-(1.6) applies. Moreover, it is easy to see that the application of modus ponens to the axioms always generates longer sentences. Therefore $a\,a$ is not a theorem of A. Is a, b, a a theorem of A? By the Axiom schema (1.6), the sentence schema $s\,b \to s\,b\,a$ holds. By Axiom (1.3), the sentence $a\,b$ holds. So, if we replace s with a in (1.6) we see that the sentence $a\,b \to a\,b\,a$ holds. Now we apply modus ponens to $P \equiv a\,b$ and $Q \equiv a\,b\,a$, which shows that $a\,b\,a$ holds: this sentence is therefore a theorem of A.

□

EXERCISE 1.5. Prove that the theory of A (the FS of Example (1.4)) consists of all and only the alternating sequences of a, b. □

EXAMPLE 1.6 (*An elementary FS about numbers*). The FS N encodes the additive group of integers.

- **Alphabet**: $\{+, -\} \cup \mathbb{N}$.
- **Grammar**:
  $$\begin{array}{rcl} \mathsf{expr} & \longrightarrow & \mathsf{expr\ op\ expr} \mid (\ -\mathsf{expr}\ ) \mid \mathsf{int} \\ \mathsf{op} & \longrightarrow & + \mid - \\ \mathsf{int} & \longrightarrow & \mathbb{N} \end{array}$$
- **Axioms**:
  $$\begin{array}{rcl} \forall a, b \in \mathsf{expr} \quad a + b & = & b + a \\ \forall a, b, c \in \mathsf{expr} \quad (a + b) + c & = & a + (b + c) \\ \forall a \in \mathsf{expr} \quad a + (-a) & = & 0 \\ \forall a \in \mathsf{expr} \quad a + 0 & = & a \\ \forall a \in \mathsf{expr} \quad a - (-a) & = & a + a \end{array}$$
- **Inference rules**: for two valid sentences P, Q of N,
  $$(P \ \wedge \ (P \to Q)) \ \to \ Q$$
  (*modus ponens*).

Some remarks.

(1) We recall that $\longrightarrow$ denotes a parsing rule in formal grammars, while $\to$ replaces "if... then..." in axioms.
(2) The symbol "=" appearing in the axioms is not explicitly part of the alphabet of N, but it is part of the first-order logic required to work with integers (see e.g. Peano's axioms).
(3) All of the axioms are in fact axiom schemata. The first four correspond to: commutativity, associativity, identity, inverse. The last one is a syntactical axiom that allows for a more humanly readable sentences.
(4) Although we did not explicitly list them, the axioms also include those that allow integer computation with sums and differences (i.e. Peano's axioms applied recursively).

□

EXERCISE 1.7. Is "$1 + 2 - 3$" a valid sentence in N? How about "$1 + 2 - -3$"? What is the theory of N (the FS of Example 1.6)? □

1.2. Decision problems

A *decision problem* is a set P of sentences. One must decide if a given sentence s belongs to P. A decision problem P is *decidable* if there exists a decision algorithm $\mathcal{D} : P \to \{0, 1\}$ such that, for each $s \in P$, $\mathcal{D}(s) = 1$ if $s \in P$, and $\mathcal{D}(s) = 0$ if $s \notin P$.

A FS F naturally provides a decision problem about its theory: given a sentence s produced by F, does it hold or not? If we call sentences that hold "provable", then one must decide if f is provable from the axioms and the rules of inferences of F. In this syntactical sense, a *proof* of s is a finite sequence of valid sentences of F ending with s, where each sentence in the sequence is either an axiom or is derived from previous sentences in the sequence by inference rules.

A simple syntactical transformation turns any sequence of sentences $(t_1, \ldots, t_n)$ into a single sentence $t_1 \wedge \ldots \wedge t_n$. Therefore proofs are also conjunctions of sentences, and, therefore, also sentences themselves.

1.2.1. Decidability and completeness. A FS F is *decidable* if its associated decision problem is decidable. Moreover, F is *complete* if, for any s produced by F, either s provable, or $\neg s$ is provable (we recall that "$\neg$" denotes logical negation). If there is at least one sentence s such that neither s nor $\neg s$ is provable in F, we say that F is *incomplete*.

If F is complete, it is also decidable. Consider the following decision algorithm: given f generated in F, starting from $i = 1$ construct all of the (finitely many) proofs of given length i. If one of these proofs proves f, stop and return YES. If one of these proofs proves $\neg f$, stop and return NO. If no proof of length i stops the algorithm, increase i and repeat. As long as F is complete, this algorithm must terminate.

The fact that there are finitely many proofs of given length follows because axiom schemata (which describe potentially infinite sets of axioms) only give rise to finitely many axioms within a given proof context.

On the other hand, there are cases of incomplete and undecidable FSs, as well as incomplete and decidable. The point is that that, even if there are sentences s such that neither s nor $\neg s$ is provable in F, a decision algorithm might still be able to recognize such s syntactically, and return 0. The subtlety is that decision algorithms simply decide whether $s \in P$ or not, but if $s \notin P$, it might be because $\neg s \in P$ or because neither s nor $\neg s$ are in P. A decision algorithm that returns 0 does not need to specify which of the two alternatives holds.

1.2.2. Other types of problems. Not all problems are decision problems. Given a function f defined by its description, does there exist an algorithm that computes it? This problem is a *computability* problem.

Exercise 1.8. Let f be the function mapping every $i \in \mathbb{N}$ to the i-th prime. Is it computable?

For problems of other types, we usually consider their *solvability*. This is also the case of optimization problems.

In the rest of this chapter, we shall provide FSs for modelling feasibility problems in Polynomial Programming (PP) and Integer Polynomial Programming (IPP). Since both of these problem classes are subclasses of NLP and MINLP respectively, we are going to be able to reason about the solvability of NLPs and MINLPs.

1.3. A FS for MP?

Although a general FS for MP was never proposed so far, we can describe the part that is used to generate valid MP formulations: the alphabet and the grammar. We are not going to give complete formal descriptions of these because it would be long and tedious. Software such as AMPL [**1**], however, provides an example of an implemented grammar for MP. The appendix of the AMPL book gives the rules of the formal grammar. We note that:

- the **alphabet** is a union of several alphabets: (i) integers; (ii) a countable supply of variable symbols; (iii) all arithmetic (sum, subtraction, multiplication, division, power) and some transcendental (logarithm, exponential, trigonometric) operators; (iv) logical quantifiers (for all, there exists), connectives (and, or, not, implies), and relations (membership, equality); (v) MP symbols (objective direction operators, sum and product quantifiers, less than or equal, greater than or equal).

- The **grammar** produces: (i) floating point numbers out of integers, (i) mathematical expressions in numbers and variables, (iii) whole MP formulations. Floating point numbers and mathematical expressions are valid formulæ, whereas formulations are sentences.

The extension of this alphabet and grammar to a full FS MP requires axioms and inference rules.

For **axioms**, a possible idea is to introduce an axiom schema based on a general solution algorithm $\mathcal{A}$ applicable to all sentences in MP, such that, for any $P \in \mathsf{MP}$, $\mathcal{A}(P) = 1$ if P has optima or feasible points, and $\mathcal{A}(P) = 0$ if P is infeasible. One inherent limitation of this idea is that non-terminating algorithms do not qualify as valid axiom schemata. This idea therefore rests on whether there exists a single general solution algorithm that establishes feasibility of all sentences in MP. We shall see in the following that such an algorithm does not exist. The best one can do is to apply this FS to a subset of MP for which such an algorithm does exist, e.g. Mixed-Integer Polynomial Programs (MIPP) with bounded integer variables.

Inference rules are not strictly necessary for this FS, but one may, for convenience, introduce exact reformulations (i.e. those that leave formulation feasibility unchanged) as a set of inference rules.

The theory of MP is the set of all feasible MP formulations.

1.4. The PA1 formal system

Here we look at a simpler FS than MP, based on Peano axioms with first order logic, called PA1. It constructs formal sentences involving integers. Recall that in first order variable symbols may only range over the "base elements" of the discourse (in this case the integers), but not over relations between integers.

EXAMPLE 1.9 (*Propositional, first, and second order logic*). The sentence $P \vee Q \to P$ is part of *propositional logic*, which deals with sentences without quantifiers. The sentence $\forall x\ (x+0 = x)$ is part of *first order logic*, an extension of propositional logic that allows the use of variables over a certain domain of discourse (in this case the integers). The formal version of the natural language sentence "for every arithmetical operator $\oplus \in \{+, \times\}$ on integers there exists an integer e such that for all integer x we have $x \oplus e = x$" is an example of *second order logic*, an extension of first order logic that allows quantification over relations concerning the domain of discourse. □

Our interest in PA1 is that, in particular, it constructs sentences corresponding to feasibility-only IPPs of the form

$$\min\{0 \mid \forall i \leq m\ p_i(x) \leq 0\},$$

where $x = (x_1, \ldots, x_n)$ is a vector of decision variables, and each $p_i(x)$ is a multivariate polynomial. Solving the IPP above involves finding a solution x' such that $p_i(x) = 0$ for all $i \leq m$. So the PA1 sentence corresponding to a IPP is of existential type:

$$\exists x \in \mathbb{N}^n\ \forall i \leq m \quad p_i(x) = 0. \tag{1.7}$$

Such a sentence "holds" if it can be proved that the quantified x actually exists, i.e. an existential PA1 sentence holds if the corresponding IPP is feasible.

Here is a description of PA1.

- **Alphabet**: $+, \times, \wedge, \vee, \rightarrow, \forall, \exists, \neg, =, 0, S(\cdot)$ and a countable set of variable symbols $x_1, x_2, \ldots$. The $S(\cdot)$ function is the *successor* function, i.e. all integers i we have $S(i) = i + 1$.
- **Grammar**: first order logic with integer arithmetic. Instead of giving a precise description, we simply recall that this grammar produces valid formulæ that may have unquantified variable symbols, and valid sentences where no unquantified variables appear. This implies that only sentences may "hold" (or not hold), and that sentences may consist of a formula prefixed by the appropriate quantifiers.
- **Axioms**: Peano axioms, namely

  PA1 $\forall x\ (0 \neq S(x))$
  PA2 $\forall x, y\ (S(x) = S(y) \rightarrow x = y)$
  PA3 $\forall x\ (x + 0 = x)$
  PA4 $\forall x\ (x \times 0 = 0)$
  PA5 $\forall x, y\ (x + S(y) = S(x + y))$
  PA6 $\forall x, y\ (x \times S(y) = x \times y + x)$
  PA7 for every $(k+1)$-ary function ϕ, if $y = (y_1, \ldots, y_k)$
  $\forall y\ (\phi(0, y) \wedge \forall x(\phi(x, y) \rightarrow \phi(S(x), y))) \rightarrow \forall x \phi(x, y)$.

  Axiom PA7 is an axiom schema that encodes mathematical induction.

- **Inference rules**: in PA1 we inherit inference rules from two more basic FSs, i.e. propositional logic (PL), and predicate calculus (PC), which adds variables to PL. See the Wikipedia entry "List of rules of inference" for a complete discussion of these rules. The most important rule is *modus ponens* (mp), which is the only one necessary as long as the sentences $P \rightarrow (Q \rightarrow P)$, $(P \rightarrow (Q \rightarrow R)) \rightarrow ((P \rightarrow Q) \rightarrow (P \rightarrow R))$, $(\neg P \rightarrow \neg Q) \rightarrow (Q \rightarrow P)$ are added to the axioms.

  Here are the main rules from PL:

  PL1 law of the excluded middle: if Q holds whenever P holds, and $\neg Q$ holds, then $\neg P$ holds
  PL2 anything follows from a contradiction: if P and $\neg P$ both hold, then any Q holds
  PL3 double negation introduction and elimination: if P holds, $\neg\neg P$ holds, and if $\neg\neg P$ holds, P holds
  PL4 deduction theorem: if Q holds whenever P holds, then $P \rightarrow Q$
  PL5 *modus ponens*: if $P \rightarrow Q$ and P both hold, then Q holds
  PL6 *modus tollens*: if $P \rightarrow Q$ and $\neg Q$ both hold, then $\neg P$ holds
  PL7 conjunction introduction: if P and Q both hold, then $P \wedge Q$ holds
  PL8 conjunction elimination: if $PandQ$ holds, then both P and Q hold
  PL9 disjunction introduction: if P holds then $P \vee Q$ holds for whatever Q; if Q holds then $P \vee Q$ holds for whatever P
  PL10 case analysis: if $P \rightarrow R$, $Q \rightarrow R$ and $P \vee Q$ all hold, then R holds
  PL11 disjunctive syllogism: if $P \vee Q$ and $\neg P$ both hold, then Q holds (and symmetrically, if $\neg Q$ holds, then P holds)
  PL12 constructive dilemma: if $P \rightarrow R$, $Q \rightarrow S$, and $P \vee Q$ all hold, then $R \vee S$ holds.

  Unlike rules of PL, the applicability of the rules of PC depends on the context (i.e. the point of the proof at which they are applied). Let

x be a variable symbol of the alphabet, f be a valid formula produced by the grammar, and $P(f|x)$ be the sentence obtained by replacing in P each occurrence of x with f, whenever x is *free* in P (i.e. not bound by a quantifier within P). The main rules of PC are:

- PC1 universal introduction: assuming (i) f is variable not occurring in P and (ii) f is not quantified in a scope including P, if $P(f|x)$ holds then $\forall x\ P$ holds
- PC2 universal eliminattion: assuming no free occurrence of x in P is within a scope of a quantifier acting on a variable in f, if $\forall x\ P$ holds, then $P(f|x)$ holds
- PC3 existential introduction: assuming no free occurrence of x in P is within a scope of a quantifier acting on a variable in f, if $P(f|x)$ holds, then $\exists x\ P$ holds
- PC4 existential elimination: assuming (i) f is a variable not occurring in P, (ii) Q is a sentence where f does not appear, (iii) f is not quantified in a scope including P or Q, if $\exists x\ P$ holds and Q can be derived from $P(f|x)$, then Q holds.

EXAMPLE 1.10 (*How proofs work in FSs*). Using PA1 we prove that the sentence $\forall x(x = x)$ holds:

by (AxiomPA3)	$\forall x$	$x + 0 = x$	(1)
by (eq)	$\forall t, r, s$	$t = r \to (t = s \to r = s)$	(2)
by $(1, 2)$	$\forall x$	$x + 0 = x \to (x + 0 = x \to x = x)$	(3)
by $(1, 3, \mathsf{mp})$	$\forall x$	$x + 0 = x \to x = x$	(4)
by $(1, 4, \mathsf{mp})$	$\forall x$	$x = x$	QED

Note: (eq) above denotes a theorem from PL with equality:
(PL Eq Axiom) $(t = r \wedge t = s) \to r = s$
(same truth tables) $t = r \to (t = s \to r = s)$. □

The theory generated by PA1 consists of the provable first-order formal sentences about $\mathbb{N}$.

One last remark about PA1: it can easily be extended to $\mathbb{Z}$ by replacing every $x \in \mathbb{Z}$ occurring in a sentence by the expression $y - z$ where $y, z \in \mathbb{N}$. This makes the sentence formally part of PA1.

1.5. The Reals formal system

We now introduce a FS that describes provable sentences about real numbers. As for PA1, we are interested specifically in existentially quantified sentences

$$\exists x \in \mathbb{R}^n\ \forall i \leq m \quad p_i(x) = 0, \tag{1.8}$$

that provide a good model for PP.

- **Alphabet**: $+, \times, \wedge, \vee, \forall, \exists, =, <, \leq, 0, 1$, countably many variable symbols.
- **Grammar**: again we do not go into details. It generates valid formulæ and sentences, similarly to PA1. Only sentences may hold (or not hold).
- **Axioms**: field axioms and order axioms, namely
  (1) Associativity of addition and multiplication (field)
  (2) Commutativity of addition and multiplication (field)
  (3) Additive and multiplicative identity (field)

(4) Additive inverses (field)
(5) Multiplicative inverses (field)
(6) Distributivity of multiplication over addition (field)
(7) Only one of $a > b$, $a = b$, $b > a$ holds (order)
(8) Transitivity: if $a > b$ and $b > c$ then $a > c$ (order)
(9) Invariance to addition and multiplication (order):
$a > b \to a + c > b + c$ and $a > b \wedge c > 0 \to ac > bc$.

- Inference rules: same as PA1.

The theory generated by Reals consists of the provable first-order formal sentences about $\mathbb{R}$.

1.6. Incompleteness of PA1

Gödel's 1st incompleteness theorem states that any FS that extends PA1 is either incomplete or inconsistent.

We recall that an incomplete FS F has sentences P such that neither P nor $\neg P$ holds. In the provability interpretation of "holds", this means that some sentences of F cannot be proved or disproved within the system. This is evidently an unsatisfactory state of affairs, in the sense that it would be more desirable to be able to prove or disprove every sentence generated by the grammar of F.

A FS F is *inconsistent* if there is a sentence P such that both P and $\neg P$ hold. By the inference rule PL2 (also known as *ex falso quodlibet*), this shows that all sentences Q produced by the FS must then hold. Although we listed this as an inference rule, it can be easily proved in PL from other inference rules:

(by hypothesis)	both P and $\neg P$ hold	
(by disjunction introduction)	$P \vee Q$ holds	
(by disjunctive syllogism)	$\neg P$ and $P \vee Q$ hold, so Q holds	QED

A FS having a theory consisting of all of its sentences offers a trivial decision problem (the decision algorithm always returning 1 on all sentences is correct) and is therefore useless (alphabet and grammar would have sufficed). An inconsistent FS is therefore totally useless.

According to Gödel's 1st incompleteness theorem, FSs extending PA1 are either unsatisfactory or totally useless.

1.6.1. Gödel's incompleteness theorem. The proof of Gödel's incompleteness theorem is both technical and thought-provoking. The technical part, however, is not very thought-provoking (just a lot of details). Accordingly, we shall focus on the thought-provoking part.

Let G be any FS extending PA1. We prove that G is complete iff it is inconsistent.

For any valid sentence P in G, "P is provable in G" is a natural language sentence about a fact related to G. We shall prove later that a formal version of this statement can be generated in G. So this statement is both part of the formal language of G as well as the informal meta-language we use to discuss G. We establish the following convention:

- we use the informal sentence "P is provable in G" in meta-language;
- we use "$\mathsf{G} \vdash P$" in the formal language.

We shall see that the formal sentence "$\mathsf{G} \vdash P$" in G can be used to generate the more interesting sentence

$$\phi \equiv \mathsf{G} \nvdash \phi. \tag{1.9}$$

In the meta-language, ϕ states "ϕ not provable in G".

Note that ϕ is self-referential. Since FSs encode syntactical operations only, there is no issue of the "ill-defined meaning" of ϕ. We are not "defining" ϕ in the sense of understanding what it means: we are simply constructing the sentence ϕ by means of the grammar and rules of G. (And, by the way, not every semantic self-referential definition is ill-defined.)

A reader might wonder why we call Eq. (1.9) a "sentence" rather than "a formula", since ϕ appears to be a free variable symbol in the string. But $\mathsf{PA1}$ is a 1st order logic, so the variable symbols may only range over integers, whereas ϕ denotes a sentence. We are actually using the various sentence variables (P, Q, ϕ, etc.) as short-hand in order to avoid writing complete sentences, which may be excessively long. The point is that ϕ can (at least potentially) be written down explicitly in a finite amount of space. This is not true of a variable ranging over the integers, which should be written down as "$0 \vee 1 \vee 2 \vee \ldots$", taking an infinite amount of space.

(1) Assume first that G is complete. Then either ϕ is provable in G or $\neg\phi$ is provable in G. We look at these two cases in turn.
   - If ϕ is provable in G then we replace ϕ with its definition (Eq. (1.9)) and infer that $\mathsf{G} \nvdash \phi$ is provable in G. This, in turn, implies that ϕ is not provable in G, which leads to a contradiction.
   - If $\neg\phi$ is provable in G, then we again replace ϕ with its definition and infer that $\neg\mathsf{G} \nvdash \phi$ is provable in G. This, in turn, implies that it is not true that ϕ is not provable in G, or, equivalently, that ϕ is provable in G. Therefore both $\neg\phi$ and ϕ are provable in G, which means that G is inconsistent.

   Since the first case leads to a contradiction, the second case necessarily holds (law of the excluded middle): the completeness of G implies its inconsistency.

(2) Assume next that G is inconsistent. Then any sentence Q generated by G is provable in G, making G complete.

Aside from the formal constructions in G of the two sentences above ($\mathsf{G} \vdash P$, and the sentence ϕ formally stating that $\mathsf{G} \nvdash \phi$), this argument concludes the proof of Gödel's 1st incompleteness theorem. If we want G to be consistent, it must be incomplete.

1.6.1.1. *Gödel's encoding.* The second part of this proof (the argument 2 above) is valid for every FS. The first part (the argument 1) actually uses the fact that G extends $\mathsf{PA1}$. Since $\mathsf{PA1}$ generates statements about integers, we will need to devise an encoding of formulæ of $\mathsf{PA1}$ in integers, and the corresponding decoding.

While every valid formula can be encoded into an integer, the encoding may be such that not every integer can be decoded into a formula. Gödel's original encoding and decoding is ingenious but complicated, and constitutes the part of Gödel's proof that is technical but not necessarily thought-provoking.

We are going to assume for simplicity that the encoding and the decoding are given: for any s generated by G we denote the unique integer corresponding to s

by $\ulcorner s \urcorner$. Conversely, for any integer n corresponding to a formula s of G, we write $\langle n \rangle = s$. Thus, for any formula s we have $\langle \ulcorner s \urcorner \rangle = s$.

The integer encoding of a formula is also called its *Gödel number*.

1.6.1.2. *The syntactical concept of proof.* A large portion of the technical part of Gödel's proof uses his encoding in order to construct the relation $\mathsf{proof}(x, y)$ on two integers x, y. This relation has the following property: as long as x, y can be decoded into sentences, $\mathsf{proof}(x, y)$ is a sentence of G that states that the sentence $\langle x \rangle$ is a proof of the sentence $\langle y \rangle$.

The realisation that a certain paragraph contains a proof of a certain statement might appear full of human significance to many mathematicians. What Gödel showed is that, as long as the reasoning is confined to a given FS, this realisation can be carried out by a computer. Moreover, he also showed that the meta-linguistic discourse that says "there's a proof of P in G" can itself be formalized directly within the FS. This is why we were able to assume, in the argument above, that the sentence "$\mathsf{G} \vdash P$" can be generated within G.

If this still sounds surprising, recall that proofs in FSs are simply sentences structured as conjunctions of smaller sentences with this property: the first is an axiom, the last is the sentence being proved, and the intermediate ones are derived from previous ones using inference rules. Valid sentences are produced by grammars over alphabets, axioms are valid sentences that hold, and inference rules produce sentences that hold from other sentences that hold. Everything is purely syntactical. Gödel's construction of $\mathsf{proof}(x, y)$ is a careful encoding in integers over every axiom and inference rule of $\mathsf{PA1}$, such that $\langle x \rangle$ is a syntactical proof of $\langle y \rangle$ within $\mathsf{PA1}$.

1.6.1.3. *Constructing Gödel's self-referential sentence.* Gödel's proof hinges over Eq. (1.9). We shall construct it from $\mathsf{proof}(x, y)$ and from a function on three integers, $\mathsf{sost}(m, n, p)$ that is defined whenever m, p are Gödel numbers of valid formulæ, and n is the Gödel number of a formula consisting of a variable symbol. This function returns an integer which is the Gödel number of the formula f obtained by replacing every variable symbol $\langle n \rangle$ in the formula $\langle m \rangle$ by the integer p.

EXAMPLE 1.11 (*Formal replacements with* sost). Consider the formula $h \equiv \neg\mathsf{proof}(x, y)$. Let $m = \ulcorner h \urcorner$, $n = \ulcorner x \urcorner$, and $p = \ulcorner a \wedge s \wedge \langle y \rangle \urcorner$, where a is an axiom of $\mathsf{PA1}$ and s is a sentence that holds in $\mathsf{PA1}$. Then $\mathsf{sost}(m, n, p)$ is the integer that encodes the formula

$$\neg\mathsf{proof}(\ulcorner a \wedge s \wedge \langle y \rangle \urcorner, y),$$

which states that $a \wedge s \wedge \langle y \rangle$ is not a proof of $\langle y \rangle$. □

Armed with proof and sost, we define in G the valid formula

$$\neg\exists x \; \mathsf{proof}(x, \mathsf{sost}(y, \langle y \rangle, y),$$

which we call $\gamma(y)$ to emphasize the fact that y is a free variable in this formula. Note the use of the sost operation in $\gamma(y)$: whenever the variable symbol y appears in the formula $\langle y \rangle$, it is replaced with the formula $\langle y \rangle$.

EXAMPLE 1.12 (*A self-referential replacement*). Consider the formula h as defined in Eg. 1.11, and let $y = \ulcorner h \urcorner$. Then $\mathsf{sost}(y, \langle y \rangle, y)$ returns the integer that encodes the formula

$$\neg\mathsf{proof}(x, \ulcorner \neg\mathsf{proof}(x, y) \urcorner) \quad (\star),$$

since the variable symbol y in the second argument of the proof relation appearing in h was replaced with h itself. The resulting formula $(\star)$ states that $\langle x \rangle$ is not a proof of the fact $\langle x \rangle$ is not a proof of y. □

We now come to the crucial point. We let $q = \ulcorner \gamma(y) \urcorner$, and define $\phi \equiv \gamma(q)$, i.e. $\phi \equiv \neg \exists x \ \mathsf{proof}(x, \mathsf{sost}(q, \langle y \rangle, q))$. Also, let $\psi \equiv \langle \mathsf{sost}(q, \langle y \rangle, q) \rangle$: this makes it obvious that ϕ states "there is no proof in G for ψ". Since $\phi \equiv \gamma(q)$ we have that

$$\boxed{\phi \text{ is generated by replacing the free variable "}y\text{" in "}\gamma(y)\text{" with } q. \qquad (\dagger)}$$

Moreover, By definition of sost, we have that

$$\boxed{\psi \text{ is generated by replacing the free variable "}y\text{" in } \langle q \rangle \text{ with } q. \qquad (\ddagger)}$$

Finally, let us look at ϕ, ψ more closely by comparing their generation processes $(\dagger)$ and $(\ddagger)$ syntactically. The only difference is that ϕ replaces in $\gamma(y)$, whereas ψ replaces in $\langle q \rangle$. But we had defined q to be the Gödel number of $\gamma(y)$, which implies $\langle q \rangle \equiv \gamma(y)$. Therefore $(\dagger)$ and $(\ddagger)$ are the same, which means that $\phi \equiv \psi$. Hence, ϕ actually states "there is no proof in G for ϕ". The last observation is that ϕ was the result of a replacement of the only free variable (y) with an integer (q), and hence ϕ is actually a sentence, not a formula.

The "crucial point" above looks like a magician's trick. Something syntactically trivial like a replacement suddenly becomes, in a sleight of hand, a self-referential sentence that allows the proof of Gödel's incompleteness theorem. Some treatments make it more obvious that there is a fixed-point process of replacement leading to a limit (Eq. (1.9)). In this treatment, I like the fact that the result emerges from the syntactical comparison between two sentences (†)-(‡) that only differ by a single symbol, which turns out to spell out to the same formula. This makes it evident that all the reasoning behind this proof could be verified by a computer.

Bibliography

[1] R. Fourer and D. Gay. *The AMPL Book*. Duxbury Press, Pacific Grove, 2002.
[2] R.M. Smullyan. *Theory of formal systems*. Princeton University Press, Princeton, NJ, 1961.