We are going to show how, given the machine table or flow chart or other suitable presentation of a Turing machine, and any n, we can effectively write down a finite set of sentences and a sentence D such that implies D if and only if the machine in question does eventually halt when started with input n, that is, when started in its initial state scanning the leftmost of a block of n strokes on an otherwise blank tape. It follows that if the decision problem for logical implication could be solved, that is, if an effective method could be devised that, applied to any finite set of sentences and sentence D, would in a finite amount of time tell us whether or not implies D, then the halting problem for Turing machines could be solved, or in other words, an effective method would exist that, applied to any suitably presented Turing machine and number n,would in a finite amount of time tell us whether or not that machine halts when started with input n

Since we have seen in Chapter 4 that, assuming Turing's thesis, the halting problem is not solvable, it follows that, again assuming Turing's thesis, the decision problem is unsolvable, or, as is said, that logic is undecidable. In principle this section requires only the material of Chapters 3–4 and 9–10. In practice some facility at recognizing simple logical implications will be required: we are going to appeal freely to various facts about one sentence implying another, leaving the verification of these facts largely to the reader. We begin by introducing simultaneously the language in which the sentences in and the sentence D will be written, and its standard interpretationM. The language interpretation will depend on what machine and what input n we are considering. The domain of M will in all cases be the integers, positive and zero and negative. The nonnegative integers will be used to number the times when the machine is operating: the machine starts at time 0. The integers will also be used to number the squares on the tape: the machine starts at square 0, and the squares to the left and right are numbered as in Figure 11-1.

There will be a constant 0, whose denotation in the standard interpretation will be zero, and two-place predicates S and <, whose denotations will be the successor relation (the relation an integer n bears to n + 1 and nothing else) and the usual order relation, respectively. To save space, we write Suv rather than S(u, v), and similarly for other predicates. As to such other predicates, there will further be, for each of the (nonhalted) states of the machine, numbered let us say from 1 (the initial state) to k, a one-place predicate. In the standard interpretation, Qi will denote the set of t ≥ 0 such that at the time numbered t the machine is in the state numbered i . Besides this we need two more two-place predicates@andM. The denotation of the former will be the set of pairs of integers t ≥ 0 and x such that at the time number t, the machine is at the square numbered x. The denotation of the latter will be the set of t ≥ 0 and x such that at time t, square x is 'marked', that is, contains a stroke rather than a blank. (We use t as the variable when a time is intended, and x and y when squares are intended, as a reminder of the standard interpretation. Formally, the function of a variable is signalled by its position in the first or the second place of the predicate @orM.) It would be easy to adapt our construction to the case where more symbols than just the stroke and the blank are allowed, but for present purposes there is no reason to do so. We must next describe the sentences that are to go into and the sentence D. The sentences in will fall into three groups. The first contains some 'background information' about S and < thatwould be the same for any machine and any input. The second consists of a single sentence specific to the input n we are considering. The third consists of one sentence for each 'normal' instruction of the specific machine we are considering, that is, for each instruction except for those telling us to halt. ◦*Isfirst − orderlogicdecidablei.e.foranygivenproof, isitpossibletostatebyfinitestepsthat*"φ*followsfrom*"Γ?*No.Theundecidabilityoffirst− orderimplicationisalsoknownasChurch′sTheorem.*

Two ways of proviing the results: using the Turing-computability, by showing that decidability proble is really the

11.1 Church's Theorem via Turing-computability and the halting problem.

11.2 Church's Theorem proven by recursive functions and the nullity problem.

General strategy of both proofs:
(Thm.0.1) 11.2 **Church's (Undecidability) Theorem**: The decision problem for logical implication is

unsolvable.

1 | Syntax

Logical Symbols

Connectives: $\neg, \vee, \wedge$

Individual variables: $v_0, v_1, \ldots$

Quantifiers: $\forall, \exists$

Identity: $=$, a 2 place predicate

Non-Logical Symbols

Constants or individual symbols: $f_1^0, f_2^0, f_3^0, \ldots$(denumerable)

k-place predicates or relation symbols:

$$\begin{array}{cccc} A_0^1 & A_1^1 & A_2^1 & \cdots \\ A_0^2 & A_1^2 & A_2^2 & \cdots \\ A_0^3 & A_1^3 & A_2^3 & \cdots \\ \vdots & \vdots & \vdots & \end{array}$$

Function symbols: (n and k are any positive integers):

$$\begin{array}{cccc} f_0^1 & f_1^1 & f_2^1 & \cdots \\ f_0^2 & f_1^2 & f_2^2 & \cdots \\ f_0^3 & f_1^3 & f_2^3 & \cdots \\ \vdots & \vdots & \vdots & \end{array}$$

Vocabulary

Dfn **1.1.** *Language: an enumerable set of non-logical symbols*

$\circ Any language is a subset of this vocabulary.$

Empty language ($L_\varnothing$): no non-logical symbols.
This language has infinitely many formulas because formulas can be constructed with identity (a logical symbol) and quantifiers. $(\forall x)(x = x)$, $(\forall x)(x = x \vee (x = x)$, and so on.

Language of Arithmetic (L^*):
individual constant: f_4^0 (say) for 0
two-place predicate: A_0^2 for $<$
one-place function symbol: f_1^2 for $'$
the two-place function symbols: f_1^3,f_2^3 for $+$,$\bullet$

Dfn **1.2.** *Term: atomic term or built up from atomic terms in a sequence of finitely many steps (in a* formation sequence*) by applying function symbols to simpler terms*

$\circ Atomic\ terms : all individual variables or individuals constants are terms.$

Formation Rules: if f is an n-place function symbol and $t_1, \ldots t_n$ are terms, $f(t_1, \ldots, t_n)$ is a term

Dfn **1.3.** *Formulas: A (first-order) formula is anything that is an atomic formula or can be build up* *from atomic formulas in a sequence of finitely many steps by applying formation rules.*

$\circ$*Atomic Formula*(for) : *astringofsymbols*R$(\mathrm{t}_1, \ldots, t_n)$ consisting of a predicate, followed by a left parenthesis, followed by *n* constants or variables, where *n* is the number of places of the predicate, with commas separating the successive terms, all followed by a right parenthesis

Formation Rules: (- means 'followed by')

if F is a formula, '¬' $-$ F is a formula

if F and G are formulas, '(' $-$ F $-$' $\vee'$ $-$ G $-$ ')', '(' $-$ F $-$' $\wedge'$ $-$ G $-$ ')' are formulas.

if F is a formula and x is a variable, '∀'$-x-$ F, '$\exists'$ $- x -$ F are formulas.

If t_1 and t_2 are terms, $= (t1, t2)$ is a formula.

Some more notions

Dfn 1.4. *subterm*: A consecutive string of symbols inside a term is a *subterm* if it is itself a term.

Dfn 1.5. *formation sequence* A sequence of finite steps by applying formation rules.

Dfn 1.6. A consecutive string of symbols inside a formula is a *subformula* if it is itself a formula.

Dfn 1.7. Terms that contain variables are *open* terms, while terms that do not are *closed* terms.

Dfn 1.8. A formula is a **sentence** if it contains no free variables.

Dfn 1.9. A **subsentence** is a subformula that is a sentence.

Rem 1.10. A formula need not be bound, and the variables associated with quantifiers need not appear in the formula. $(\forall x)Fx$, $(\forall x)Fxy$, $(\forall y)Fxy$, $(\forall z)Fxy$, $(\forall x)(\forall y)Fxy$

Abbreviations in Logic

Symbol	Official	Unofficial
$<$	$<(x,y)$	$x<y$
P	$P(x,y)$	Pxy
$\wedge$	$(A \wedge (B \wedge (C \wedge D)))$	$(A \wedge B \wedge C \wedge D)$
$\rightarrow$	$(\neg G \vee F)$	$(F \leftrightarrow G)$
$\leftrightarrow$	$((\neg G \vee F) \wedge (\neg F \vee G))$	$(F \rightarrow G)$
$\forall y < x$	$\forall y(y<x) \rightarrow \ldots)$	$\forall y < x$
$\exists y < x$	$\exists y(y<x) \rightarrow \ldots$	$\exists y < x$

Abbreviations in Language of Arithmetic (L*))

Official	Unofficial
v_0	x
f_0^0	0
$f_0^1(f_0^0)$	1
$f_0^1(f_0^1(f_0^0))$	2
$f_1^2(f_0^1(f_0^1(f_0^0)), v_0)$	$2 \bullet x$
$f_0^2(f_1^2(f_0^1(f_0^1(f_0^0)), v_0), f_1^2(f_0^1(f_0^1(f_0^0)), v_0))$	$2 \bullet x + 2 \bullet x$

2 | Syntactic Properties

Lem **2.1.** *Parenthesis Lemma (BJ-L9.4): When formulas are written in official notation the following hold:*

Every formula ends in a right parenthesis.

Every formula has equally many left and right parenthesis.

If a formula is divided into a left part and a right part, both nonempty, then there are at least as many left as right parentheses in the left part, and more if that part contains at least one parenthesis.

Lem **2.2.** *Unique Readability Lemma (BJ-L9.5)*

The only subformula of an atomic formula $R(t_1, \ldots, t_n)$ *or* $=(t_1, t_2)$ *is itself.*

The only subformulas of $\neg F$ *are itself and the subformulas of F*

The only subformulas of $(F \wedge G)$ *or* $(F \vee G)$ *are itself and the subformulas of F and G.*

The only subformulas of $\forall xF$ *or* $\exists xF$ *are itself and subformulas of F*

Lem **2.3.** *(BJ-P9.4) In a formation sequence for a formula F, every subformula of F must appear*

Lem **2.4.** *(BJ-P9.5) Every formula F has a formation sequence in which the only formulas that appear are subformulas of F, adn the number of formulas that appear is no greater than the number of symbols in F*