



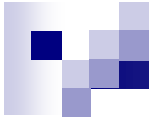
# Decidable and Undecidable Theories

Robert Keller  
March 2011



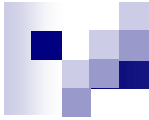
# Predicate Calculus is Generally Undecidable

- There is no algorithm that will determine whether or not an arbitrary predicate calculus formula is derivable from a given set of formulas.
- You showed this with your reduction of a Turing machine to Otter. The construction is (or should be) completely general.



## Predicate Calculus is Recognizable

- There is an algorithm for **enumerating** all the formulas that can be derived from a given set of formulas.
- In other words, the set of **derivable** formulas is recursively-enumerable, aka **recognizable**.



## Predicate Calculus is not Co-Recognizable

- There is no algorithm for determining that a given formula is **not** derivable.
- If there were, then the set of derivable formulas would be decidable, which it isn't, as already mentioned.



# Restricted Subsets of the Predicate Calculus

- For example, suppose we require that there be **no predicate symbols**, other than 0-ary ones.
- Then the calculus would effectively turn into propositional calculus, which is decidable.
- This raises the question of whether there are interesting **in-between** cases.



# Spectrum of Decidability

Arbitrary predicates,  
constants, functions

Only 0-ary  
predicates



undecidable

decidable



# Decidable Theories

- However, for certain **special categories of formulas** (cf. W. Ackerman, W. Solvable Cases of the Decision Problem. North-Holland Publishing Co., 1954),  
  
or for **certain theories**, the question derivability question **can** be decided algorithmically.
- What Makes a Theory Decidable?
  - **Restricting the language** to a specific set of function, predicate, and constant symbols.
  - **Axioms** that constrain the possible **models** to ones where questions can be answered by computation.



# What is a Theory?

- A theory restricts the language of predicate calculus to use specific function and predicate symbols  
(e.g.  $+$ ,  $x$ ,  $=$ ,  $<$ ).
- A theory specifies certain **axioms** and focuses on **theorems**, i.e. the formulas derived from those axioms using the rules of the framework.



## Decidability of a Theory

- This is distinct from the decidability of the framework. It is generally more restrictive.
- A **theory** is **decidable** iff there is an algorithm that decides whether any formula in the language of the theory is a theorem or not.



# Completeness of a Theory

- Completeness is related to, but not the same as decidability.
- It is also **not the same** as completeness of a framework. (For example, the predicate calculus *is* complete.)
- **A theory is complete** iff for any closed formula  $\varphi$  in the language of the theory, either  $\varphi$  is a theorem or  $\neg\varphi$  is a theorem. [Sometimes this called “negation-complete”.]



# Completeness of a Framework

- Natural deduction, or other systems equivalent to it, are usually **complete** in the sense that:

If a formula  $\varphi$  is true for **every model**, then  $\varphi$  is derivable.

- However, there can be formulas in the language of a **theory** that are not true in **every** model (even though they **might** be true in models of interest). If this happens, the **theory** is incomplete.
- Hence completeness of a **theory** speaks to **how the well the axioms capture the model(s) of interest.**



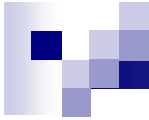
# More on Theory Completeness

- Why closed formula?
  - Closed formulas (ones with no free variables) evaluate to **true or false** under a given interpretation.
  - Open formulas require an assignment (of values to variables) to give them a truth value.
- Mainly the focus on closed formulas is a convenience, because the truth of a non-closed formula can be equated to the truth of its **closure** (just add an outer  $\forall$  for each free variable).



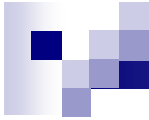
## Models, by the way

- Some authors, including Sipser, use the word “model” for “interpretation”.
- Others, including these notes, use “model” only for an interpretation in which the formula is **true**.



## Why is completeness good?

- Given an **arbitrary closed formula** in the language of the theory, we'd like to be able to know it is either true or false for all interpretations that satisfy the axioms (i.e. all models).
- That is the main point in having a theory in the first place.



# Why is incompleteness not fatal?

- If the theory is incomplete, we just can't know whether an arbitrary formula is true for all interpretations that satisfy the axioms.
- It doesn't mean that there isn't a proof for many true formulas.
- It might be that:
  - We only have **some** of the axioms needed to characterize the interpretations of interest.
  - A formula might not be derivable, but its negation might not be either. At the same time, it could be true, or its negation could be true.



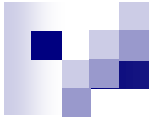
# Axiomatizability

- A theory is **finitely-axiomatizable** iff all of its theorems can be derived using only a finite set of axioms.
- A theory is **recursively-axiomatizable** iff all of its theorems can be derived using a decidable set of axioms.
- Obviously the latter are of great practical interest. If we can't tell whether something is an axiom, what good is the theory?



# Examples

- Group theory is **finitely**-axiomatizable (3 axioms)
- Peano arithmetic is **recursively**-axiomatizable, but not finitely-axiomatizable (due to the induction axiom schema).



# Meta-Theorem

- If a theory is both:
  - complete
  - recursively-axiomatizablethen it is also
  - decidable.
- Proof? Given formula  $\varphi$ , how do we tell whether it is  $\varphi$  or  $\neg\varphi$  that is a theorem?



# Corollary

- If a theory is both:
  - recursively-axiomatizable
  - undecidablethen it is also
  - incomplete.
- Peano arithmetic is such a theory.



# Decidable does not imply Complete

- For example:
  - ACF: Theory of algebraically-closed fields



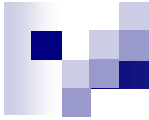
# Consistency

- A theory is **consistent** iff  $\perp$  is not a theorem.
- In other words, a theory is **inconsistent** iff  $\perp$  is derivable,
- If we can derive  $\perp$  then we can derive **all** formulas, so the theory is worthless: there is no information value in saying something is a theorem.
- Inconsistency is equivalent to there being a  $\varphi$  such that **both**  $\varphi$  and  $\neg\varphi$  are derivable.



# Trivial Matters

- An inconsistent theory is complete.
- An inconsistent theory is decidable.



# Some Known Decidable Theories

- **Theory of Equality**, with no predicate symbols
- **One unary function**, with equality
- **Various limitations on quantifiers**
- **Presburger Arithmetic** (natural numbers with just  $+$ , no  $\times$ , and with a form of induction)
- **Robinson Arithmetic** (natural numbers with  $+$ ,  $\times$ ,  $<$ , but **no induction**)
- **Dense Linear Orders**
- **Real Closed Fields**
- **Boolean Algebras**



# Presburger Arithmetic

- $(\forall x) \neg(0 = x + 1)$
- $(\forall x)(\forall y) \neg(x = y) \rightarrow \neg(x + 1 = y + 1)$
- $(\forall x) x + 0 = x$
- $(\forall x)(\forall y) (x + y) + 1 = x + (y + 1)$
- If  $P$  is any formula involving a single free variable  $x$ :

$$( P[0/x] \wedge (\forall x) (P \rightarrow P[(x + 1)/x]) ) \rightarrow (\forall x) P$$

- Formulas can be decided based on a **finite automata-like** construction, as shown by M. Presburger in 1929!
- This theory has since been more of interest to computer scientists than logicians: In 1974, Fischer and Rabin proved that any decision procedure requires **at least double exponential time** ( $2^{2^n}$ ) where  $n$  is the length of the formula. In 1978, Oppen showed a triple exponential upper bound.



From [http://en.wikipedia.org/wiki/Presburger\\_arithmetic](http://en.wikipedia.org/wiki/Presburger_arithmetic)

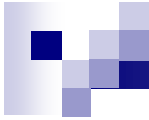
Mojżesz Presburger [1929] proved Presburger arithmetic to be:

- **consistent**
- **complete**: For each statement in Presburger arithmetic, either it is possible to deduce it from the axioms or it is possible to deduce its negation.
- **decidable**: There exists an algorithm which decides whether any given statement in Presburger arithmetic is true or false.



How can a theory for numbers be **complete** if it does not include such things as  $\times$  (multiply)?

- Although the theory applies to numbers, it can apply to other models as well.
- If it doesn't include  $\times$ , it only makes statements about the functions it does include.
- In other words, the language of the theory must be taken into account when discussing completeness.



# Techniques for Decidability

- often involve some form of **Quantifier-Elimination**:
- Put the formula in **Prenex** form, then come up with some kind of algorithmic analysis based on the matrix and how the quantifiers are stacked.
- A nice example, illustrating the application of computability ideas to logic, follows.

# Decidability of arithmetic with only + (no x)

- Based on how **DFAs can check addition** (Sipser prob 1.32):

1.32 Let

$$\Sigma_3 = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}.$$

$\Sigma_3$  contains all size 3 columns of 0s and 1s. A string of symbols in  $\Sigma_3$  gives three rows of 0s and 1s. Consider each row to be a binary number and let

$$B = \{w \in \Sigma_3^* \mid \text{the bottom row of } w \text{ is the sum of the top two rows}\}.$$

For example,

LSB first

$$\begin{array}{r} 3 \\ +1 \\ \hline 4 \end{array} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \in B, \quad \text{but} \quad \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \notin B. \quad \begin{array}{r} 1 \\ +0 \\ \hline 3 \text{ not!} \end{array}$$

Show that  $B$  is regular. (Hint: Working with  $B^R$  is easier. You may assume the result claimed in Problem 1.31.)

# Decidable Theory, continued

Here we use a generalization of this method to present  $i$ -tuples of numbers in parallel using an alphabet with  $2^i$  symbols.

We give an algorithm that can determine whether its input, a sentence  $\phi$  in the language of  $(\mathcal{N}, +)$ , is true in that model. Let

$$\phi = Q_1x_1 Q_2x_2 \cdots Q_lx_l [\psi],$$

where  $Q_1, \dots, Q_l$  each represent either  $\exists$  or  $\forall$  and  $\psi$  is a formula without quantifiers that has variables  $x_1, \dots, x_l$ . For each  $i$  from 0 to  $l$ , define formula  $\phi_i$  to be

$$\phi_i = Q_{i+1}x_{i+1} Q_{i+2}x_{i+2} \cdots Q_lx_l [\psi].$$

Thus  $\phi_0 = \phi$  and  $\phi_l = \psi$ .

Formula  $\phi_i$  has  $i$  free variables. For  $a_1, \dots, a_i \in \mathcal{N}$  write  $\phi_i(a_1, \dots, a_i)$  to be the sentence obtained by substituting the constants  $a_1, \dots, a_i$  for the variables  $x_1, \dots, x_i$  in  $\phi_i$ .

# Decidable Theory, continued

For each  $i$  from 0 to  $l$ , the algorithm constructs a finite automaton  $A_i$  that recognizes the collection of strings representing  $i$ -tuples of numbers that make  $\phi_i$  true. The algorithm begins by constructing  $A_l$  directly, using a generalization of the method in the solution to Problem 1.32. Then, for each  $i$  from  $l$  down to 1, it uses  $A_i$  to construct  $A_{i-1}$ . Finally, once the algorithm has  $A_0$ , it tests whether  $A_0$  accepts the empty string. If it does,  $\phi$  is true and the algorithm accepts.

**PROOF** For  $i > 0$  define the alphabet

# of rows =  
# of free variables  
in  $\phi_i$

$$\Sigma_i = \left\{ \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 1 \end{bmatrix}, \dots, \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} \right\}.$$

Hence  $\Sigma_i$  contains all size  $i$  columns of 0s and 1s. A string over  $\Sigma_i$  represents  $i$  binary integers (reading across the rows). We also define  $\Sigma_0 = \{[]\}$ , where  $[]$  is a symbol.

# Decidable Theory, continued

We now present an algorithm that decides  $\text{Th}(\mathcal{N}, +)$ . On input  $\phi$  where  $\phi$  is a sentence, the algorithm operates as follows. Write  $\phi$  and define  $\phi_i$  for each  $i$  from 0 to  $l$ , as in the proof idea. For each such  $i$  construct a finite automaton  $A_i$  from  $\phi_i$  that accepts strings over  $\Sigma_i^*$  corresponding to  $i$ -tuples  $a_1, \dots, a_i$  whenever  $\phi_i(a_1, \dots, a_i)$  is true, as follows.

To construct the first machine  $A_l$ , observe that  $\phi_l = \psi$  is a Boolean combination of atomic formulas. An atomic formula in the language of  $\text{Th}(\mathcal{N}, +)$  is a single addition. Finite automata can be constructed to compute any of these individual relations corresponding to a single addition and then combined to give the automaton  $A_l$ . Doing so involves the use of the regular language closure constructions for union, intersection, and complementation to compute Boolean combinations of the atomic formulas.

# Decidable Theory, continued

Next, we show how to construct  $A_i$  from  $A_{i+1}$ . If  $\phi_i = \exists x_{i+1} \phi_{i+1}$ , we construct  $A_i$  to operate as  $A_{i+1}$  operates, except that it nondeterministically guesses the value of  $a_{i+1}$  instead of receiving it as part of the input.

More precisely,  $A_i$  contains a state for each  $A_{i+1}$  state and a new start state. Every time  $A_i$  reads a symbol

$$\begin{bmatrix} b_1 \\ \vdots \\ b_{i-1} \\ b_i \end{bmatrix},$$

where every  $b_i \in \{0,1\}$  is a bit of the number  $a_i$ , it nondeterministically guesses  $z \in \{0,1\}$  and simulates  $A_{i+1}$  on the input symbol

$$\begin{bmatrix} b_1 \\ \vdots \\ b_{i-1} \\ b_i \\ z \end{bmatrix}.$$

# Decidable Theory, continued

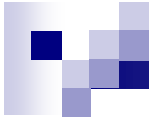
Initially,  $A_i$  nondeterministically guesses the leading bits of  $z$  corresponding to suppressed leading 0s in  $b_1$  through  $b_i$  by nondeterministically branching from its new start state to all states that  $A_{i+1}$  could reach from its start state with input strings of the symbols

$$\left\{ \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \right\}$$

in  $\Sigma_{i+1}$ . Clearly,  $A_i$  accepts its input  $(a_1, \dots, a_i)$  if some  $a_{i+1}$  exists where  $A_{i+1}$  accepts  $(a_1, \dots, a_{i+1})$ .

If  $\phi_i = \forall x_{i+1} \phi_{i+1}$ , it is equivalent to  $\neg \exists x_{i+1} \neg \phi_{i+1}$ . Thus we can construct the finite automaton that recognizes the complement of the language of  $A_{i+1}$ , then apply the preceding construction for the  $\exists$  quantifier, and finally apply complementation once again to obtain  $A_i$ .

Finite automaton  $A_0$  accepts any input iff  $\phi_0$  is true. So the final step of the algorithm tests whether  $A_0$  accepts  $\epsilon$ . If it does,  $\phi$  is true and the algorithm accepts; otherwise, it rejects.



## Some Known **U**ndecidable Theories

- **Equality and 2 unary function symbols**, otherwise unrestricted
- **Equality and a 2-ary function symbol** , otherwise unrestricted
- **Peano Arithmetic** (natural numbers with  $+$ ,  $\times$ , but **with induction**)
- **Rationals**, with  $+$ ,  $\times$ , and equality



# Three (Meta-)Theorems by Gödel

- **Completeness Theorem:**
  - Relates to deductive **framework** being complete
- **Incompleteness Theorem:**
  - Relates to **Number Theory** being incomplete
- **Second Incompleteness Theorem:**
  - Relates to theories being unable to prove their own consistency



# Gödel's Completeness Theorem

- The completeness of first-order predicate calculus as a **deductive system** was first established by Kurt Gödel in his dissertation in 1929.
- Kurt Gödel, "Über die Vollständigkeit des Logikkalküls", doctoral dissertation, University Of Vienna, 1929.
- Kurt Gödel, "Die Vollständigkeit der Axiome des logischen Funktionen-kalküls", Monatshefte für Mathematik und Physik 37 (1930), 349-360. This article contains the same material as the doctoral dissertation, in a rewritten and shortened form.



# Gödel's Incompleteness Theorem (as refined by Rosser)

- **No consistent recursively-axiomatized extension of number theory is negation-complete.**
- In other words, we can *try* to build up an all-powerful mathematical theory. At a minimum, it must include number theory, which is not asking very much. But such a theory will always be **incomplete**.
- This result, published in 1931, meant that Hilbert's idea of mechanizing all of mathematics could **never** be achieved.
- Kurt Gödel, *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I*. Monatshefte für Mathematik und Physik, 38 (1931), pp. 173-198. Translated in van Heijenoort: *From Frege to Gödel*. Harvard University Press, 1971., online at <http://home.ddc.net/ygg/etext/godel/>
- B. Rosser: *Extensions of some theorems of Gödel and Church*. Journal of Symbolic Logic, 1 (1936), N1, pp. 87-91.



# Contrapositive

- A complete, consistent theory must be weak, in that it cannot even derive all true formulas about the natural numbers.



# Number Theory

- Essentially one with  $+$ ,  $*$ , and induction, usually called **Peano Arithmetic**
- $(\forall x) \neg(S(x) = 0)$
- $(\forall x) (\forall y) ((S(x) = S(y)) \rightarrow (x = y))$
- $(\forall x) (x + 0) = x$
- $(\forall x) (\forall y) (x + S(y)) = S(x+y)$
- $(\forall x) (x * 0) = 0$
- $(\forall x) (\forall y) (\forall z) (x * S(y)) = ((x * y) + x)$
- $(\varphi[0/x] \wedge (\forall n) (\varphi[n/x] \rightarrow \varphi[S(n)/x])) \rightarrow (\forall n)\varphi[n/x]$   
for **every formula**  $\varphi$  where  $n$  is free for  $x$ .



# How did Gödel's proof work?

- Similar to the encoding of Turing machine tapes as numbers and Turing machines as partial recursive functions, Gödel first showed how to:
  - encode formulas as numbers
  - encode proofs as numbers
- Such encodings are often called **Gödel numberings**.



# How did Gödel's proof work?

- He then showed how to construct a number-theoretic formula  $\Pi$  (for "provable"), where

$\Pi(p, f, a)$  means

If  $\mathbf{f}$  is the encoding of a formula  $\varphi(v)$  with  $v$  representing the one and only free variable, and

- $\mathbf{p}$  is the encoding of a **proof** of formula  $\varphi(a)$ , and
- $\mathbf{a}$  is an **argument**, substituted for  $v$  in  $\varphi(v)$ .
- That is,  $\Pi(p, f, a)$  means  **$\mathbf{p}$  proves  $\varphi(\mathbf{a})$** .
- The actual structure of  $\Pi$  is quite large and complex.



# How did Gödel's proof work?

- $\Pi(p, f, a)$  means  $p$  is a proof of  $\varphi(a)$ , where **f encodes**  $\varphi$ .
- Consider the following formula  $\Delta(f)$  incorporating formula  $\Pi$ :

$$\Delta(f) \text{ is } (\forall p) \neg \Pi(p, f, f)$$

meaning there is **no** proof of  $\varphi(f)$ .

- Let **d** be the encoding of  $\Delta(f)$ . Then  $\Delta(\mathbf{d})$  says

$\Delta(\mathbf{d})$  is **un**provable.

- **Now is  $\Delta(\mathbf{d})$  provable?** If it is, then by the completeness theorem (for the framework),  $\Delta(\mathbf{d})$  would then be true. But if it is true, then from the very meaning of  $\Delta(\mathbf{d})$ , it is **not** provable, which is a contradiction.
- So  $\Delta(\mathbf{d})$  is **not provable**.



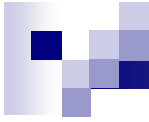
# How did Gödel's proof work?

- Is the negation,  $\neg\Delta(d)$ , provable?
- If it were provable, then it would be true (unless the theory is inconsistent), by soundness of the framework.
- But if  $\neg\Delta(d)$  is *true*, then  $\Delta(d)$  is false, meaning:  $\Delta(d)$  *is* not provable, i.e.  $\Delta(d)$  is provable, but we just showed that can't be on the previous page.
- Since neither  $\Delta(d)$  nor  $\neg\Delta(d)$  is provable, we are forced to conclude that **number theory, if consistent, is not (negation-) complete.**
- For a more rigorous proof using the natural deduction framework, see the book by van Dalen, "Logic and Structure".



# Connection with Decidability

- In particular, what does Gödel's incompleteness theorem have to do with **decidability** (= solvability)?
- First, the construction of  $\Delta(d)$  should look roughly similar to the proof of undecidability of the acceptance problem **using the recursion theorem**:
  - $\Delta(d)$  asserts there is **no** proof of  $\Delta(d)$
  - $M(x)$  converges iff  $M(x)$  diverges on  $x$ .
- Second (and this is not obvious in our brief exposition) the functions used to build up  $\Pi$ , from which  $\Delta$  is derived, are the **partial recursive functions**.



# Decidability of Number Theory

- Number theory is incomplete.
- But it still could conceivably be decidable. We haven't shown that it isn't so far.
- We can, however, reduce  $A_{TM}$  to the problem of deciding a formula in number theory.



# Number Theory is Undecidable

- A Turing machine computation can be uniformly encoded as a number theoretic functions. In slides on partial-recursive functions, we stated:
  - Halting in  $i$  steps on  $x_0$  is expressed by:

$$\mu_i [P(T(i, x_0)) = 0]$$

- So the logic formula that expresses whether a machine, coded as primitive-recursive functions  $P$  and  $T$ , halts is:


$$\exists i [P(T(i, x_0)) = 0]$$

- This formula is either true or false. If we could determine which, we'd be solving the halting problem.



## Gödel's Second Incompleteness Theorem

- Within any consistent-extension of number theory, there is a formula that expresses the **consistency** of the theory but which is not provable within the theory.



# Connections Between Logic, Computability, and Complexity

- A decision problem is in **NP** if it can be solved by a **Non-deterministic Turing machine** in time **Polynomial** in the length of the input.
- A decision problem is in **P** if it can be solved by a **deterministic** Turing machine in Polynomial time.
- A problem is **NP-complete** if it is NP and any other problem in NP **reduces** to it by a polynomial-time reduction.
- The theory of NP came, in part, from Stephen Cook's investigation of complexity of **automatic theorem proving**: The problem of determining whether a **propositional** formula is **satisfiable** is NP-complete.
- It is **unknown** whether  $P = NP$ . (It seems **unlikely**, but no one has been able to prove this yet.)
- If  $P = NP$ , every NP complete problem (of which there are many) has a polynomial-time algorithm.
- For a list of 88 NP-complete problems, see:  
[http://www.csc.liv.ac.uk/~ped/teachadmin/COMP202/annotated\\_np.html](http://www.csc.liv.ac.uk/~ped/teachadmin/COMP202/annotated_np.html)



# Logic and Computing Timeline

- 1822 Babbage: Difference Engine
- 1847 Boole: Laws of Thought
- 1866 Boolean Algebra applied to Switching
- 1890 Hollerith Tabulating Machines
- 1900 Fleming invents Vacuum tube
- 1929 Gödel: Completeness Theorem
- 1931 Gödel: Incompleteness Theorem
- 1935 Church: An unsolvable problem of elementary number theory, lambda calculus
- 1936 Kleene: General recursive functions, unsolvability
- 1936 Post: computing processes (essentially equivalent to Turing machines)
- 1937 Turing invents the Turing machine, uncomputability
- 1939 Stored-program electronic computer
- 1947 First transistor
- 1948 Univac: First commercial electronic computer
- 1954 Fortran
- 1958 Integrated circuit
- 1963 Sutherland's Sketchpad
- 1963 LINC minicomputer
- 1969 Arpanet
- 1971 Microprocessor
- 1972 Pocket calculator
- 1973 Xerox alto, point-and-click
- 1975 Ethernet
- 1976 Apple I
- 1981 IBM PC
- 1983 Visicalc
- 1984 Macintosh