**Theorem**  
CFLs are closed under union (\( U \)), concatenation (\( \cdot \)) and closure (\( * \)).

**Proof**
- Idea: Grammar-based construction.
- Goal: Given CFLs \( L_1 \) and \( L_2 \), show that the following are CFLs:
  - \( L_1 U L_2 \)
  - \( L_1 \cdot L_2 \)
  - \( L_1^* \)

Let CFLs \( L_1 \) and \( L_2 \) have CFGs \( G_1 \) and \( G_2 \).

Assume all variables in \( G_1 \) and \( G_2 \) are distinct with start symbols \( S_1 \) and \( S_2 \).

**New Grammars**
(a) \( L_1 U L_2 \) combine \( G_1 \) and \( G_2 \) via the production:
\[
S \rightarrow S_1 | S_2
\]
(b) \( L_1 \cdot L_2 \) use \( S \rightarrow S_1 . S_2 \)
(c) \( L_1^* \) add \( S \rightarrow S_1 . S | \epsilon \) to \( G_1 \)

**Correctness** Simple Exercise

Textbook proves closure under "substitution"—more general and includes above as special cases, but amounts to same proof.
EXAMPLE

\[ L_1 = \{ 0^i 1^i 2^k \mid i = j \} \]
\[ L_2 = \{ 0^j 1^i 2^k \mid j = k \} \]

\[ G_1 \]
\[ S_1 \rightarrow AB \]  \hspace{1cm} \[ G_2 \]
\[ S_2 \rightarrow CD \]
\[ A \rightarrow 0A1 \mid e \]
\[ C \rightarrow 0C \mid e \]
\[ B \rightarrow 2B \mid e \]
\[ D \rightarrow 1D2 \mid e \]

CONSIDER
\[ L = L_1 \cup L_2 = \{ 0^i 1^i 2^k \mid i = j \text{ OR } j = k \} \]

GRAMMAR \( G \)
- USE ALL OF \( G_1, G_2 \)
- ADD \( S \rightarrow S_1 \mid S_2 \).

THEOREM
CFLS NOT CLOSED UNDER INTERSECTION

PROOF
CONSIDER \( L_1, L_2 \) IN ABOVE EXAMPLE
DEFINE \( L = L_1 \cap L_2 \)

\[ \Rightarrow L = \{ 0^i 1^i 2^k \mid i = j \text{ AND } j = k \} \]
\[ = \{ 0^n 1^n 2^n \mid n \geq 0 \} \]

BUT PUMPING LEMMA SHOWS \( L \) IS NOT CFL.

THEOREM
CFLS NOT CLOSED UNDER COMPLEMENT.

PROOF
OTHERWISE WOULD GET CLOSURE UNDER \( \cap \)

SINCE \( L_1 \cap L_2 = \overline{L_1 \cup L_2} \)
AND CLOSED UNDER UNION.

THEOREM
CFLS CLOSED UNDER REVERSAL

IDEA
REVERSE R.H.S. OF EACH PRODUCTION

E.G.
\[ A \rightarrow BCD \]
BECOMES \[ A \rightarrow DCB \]
**Theorem**  
CFLs closed under intersection with regular languages  

**OR**  
L is CFL and R is regular \(\Rightarrow L \cap R \) is CFL.

**Idea**

**Given**  
PDA \(P\) for \(L\), DFA \(D\) for \(R\).

**Make**  
PDA \(M\) for \(L \cap R\).

**States of \(M\)**

- \(q_1, p\)  
- \(D's\) state \(\rightarrow\) \(P's\) state

**Transitions**

**Suppose**

\[
\begin{align*}
\text{IN } D & \quad s(q, a) \rightarrow q' \\
\text{IN } P & \quad s(p, a, x) \rightarrow (p', \delta')
\end{align*}
\]

**Then**

\[
\begin{align*}
\text{IN } M & \quad s([q, p], a, x) \rightarrow ([q', p'], \delta)
\end{align*}
\]

If \(a = \epsilon\) then \(q' = q\).

**Accept?**

Assume \(L\) is final state language of \(P\)

Final states of \(M\) \([q, p]\) with \(q\) final in \(D\) and \(p\) final in \(P\).
**CFL Decision Problems.**

**Emptiness**
Given CFL $L$ (as PDA/CFG), is $L = \emptyset$?

**Idea**
$L = \emptyset \iff$ in CFG, start $S$ is useless

**Thus**
Use algorithm for "useless") elimination.

**Membership**
Given $w$ and CFL $L$, is $w \in L$?

**Idea**
Assume given CFG $G$ for $L$

**Step 1**
If $w = \epsilon$, check if $S$ is nullable

**Step 2**
Construct CNF grammar $G'$ for $L - \{\epsilon\}$

**Step 3**
Try all possible derivations in $G'$ of length $\leq 2n-1$, where $n = |w|$

At most $|P|^2n-1$ possibilities

Why $2n-1$?
For $|w| = n$ and CNF grammar $G'$

$S \xrightarrow{*} w$

In exactly $2n-1$ steps

**Finiteness**
Similar to Regular — homework problem.

**Equality?**
Given CFL's $L_1, L_2$, is $L_1 = L_2$?

**Ambiguity?**
Given CFG $G$, is $G$ ambiguous?

Claim such problems cannot be solved — need Turing machines to prove that!

Recall equality—testing for REG. LANG. Used closure under complement/intersection

$$(L_1 = L_2 \iff L = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset)$$
DECIDABILITY THEORY

**Question**
How do we show that there is no algorithm to decide equality/ambiguity of CFLs or any other problem which is not computable?

**Problem**
We don't have a formal, mathematical definition of algorithms.

**Simpler Goal?**
Let us try to prove that there is no C program to solve such problems.

---

**Hello-World Problem.**

**Program P₀**

```c
main ()
{
    printf ("Hello, World \n");
}
```

Clearly, the first 12 characters output by P₀ is "Hello, World"

---

**Hello-World Problem**

Given an arbitrary C program P and an input I for P, does P(I) print "Hello, World" as its first 12 characters?

**Consider** a solution H (itself a C program)

```
I    H
    /
   / /
  ?   YES
```

**Question**
Does there exist such C program H?

**Of course**
You might try to write H by having it scan P's printf statements, but we need to know if those will be executed on input I.
FERMATS LAST THEOREM. THE EQUATION \( x^n + y^n = z^n \) HAS NO INTEGER SOLUTION FOR \( n \geq 3 \).

OBSERVE FOR \( n = 2 \), ONE SOLUTION IS \( (x=3, y=4, z=5) \)

\[ 3^2 + 4^2 = 5^2. \]

BUT FOR \( n \geq 3 \) FOR 300 YEARS NO PROOF WAS KNOWN UNTIL RECENTLY, WHEN WYLES GAVE A PROOF.

BUT HIS FIRST PROOF WAS WRONG, AND SECOND PROOF IS LONG & COMPLEX AND NOT YET FULLY VERIFIED.

CONSIDER C PROGRAM \( P_1 \) (GIVEN FULLY IN BOOK).

\[ P_1 \]
1) TAKE INPUT \( n \)
2) FOR ALL POSSIBLE \( (x, y, z) \) DO
   IF \( x^n + y^n = z^n \)
   THEN PRINTF ("HELLO, WORLD\n")

SUPPOSE INPUT = 3, THEN \( P_1 \) PRINTS "HELLO, WORLD" ONLY IF F.L.T. IS FALSE, ELSE IT LOOPS FOREVER.

QUESTION HOW CAN YOU (OR, C PROGRAM \( H \)) DECIDE IF \( P_1 \) PRINTS "HELLO, WORLD"?

THEOREM THERE CANNOT EXIST A C-PROGRAM \( H \) WHICH IS A HELLO-WORLD TESTER.

PROOF ASSUME \( H \) EXISTS
SHOW CONTRADICTION.
Recall on input \((P,I)\), program \(H\) outputs "YES" if \(P(I)\) prints "HELLO, WORLD", else \(H\) prints "\(\)"

Modify \(H\) to C program \(H_1\), which behaves exactly like \(H\) but prints "HELLO, WORLD" instead of "\(\)"

\[
\begin{array}{c}
P's \text{ input } I \rightarrow \text{H}_1 \\
\text{C program } P \rightarrow \text{YES} \rightarrow \text{HELLO, WORLD}.
\end{array}
\]

Note obtaining \(H_1\) requires modifying \(H's\) printf("\(\)"

Modify \(H_1\) to C program \(H_2\), which is same as \(H_1\) but takes only input \(P\) and makes \(I\) same as \(P\).

Idea write a "buffer" function which reads in "\(P\)", buffers it, and then feeds \(P/I\) to \(H_1\)

\[
\begin{array}{c}
\text{C program } P \rightarrow \text{buffer} \rightarrow H_1 \\
\rightarrow \text{YES} \rightarrow \text{HELLO, WORLD}.
\end{array}
\]

Question what is the output of \(H_2(P)\) when \(P=H_2\)?

Suppose \(H_2(H_2)=\text{YES} \Rightarrow P(P)=\text{HELLO, WORLD}\)

Suppose \(H_2(H_2)=\text{HELLO, WORLD} \Rightarrow P(P)=\text{HELLO, WORLD}\)

But \(P=H_2\)

Get contradiction \(\Rightarrow H, H_1, H_2\) cannot exist!
REDUCTIONS

So far we showed hello-world problem is undecidable.

Next we show new problems can be shown undecidable via reduction to hello-world, without having to repeat entire proof structure used for hello-world.

**FOO Problem**

Given program R and its input z, does R ever call a function named foo while executing on input z?

**Idea**

We show that if it's possible to solve the foo problem on \((R, z)\), then we can solve the hello-world problem on \((Q, y)\) — for any program Q with input y.

Since latter is undecidable, so is the former.

**Suppose**

Some program F can take as input \((R, z)\) and decide the foo problem.

We show F can be used to construct H for hello-world decision on \((Q, y)\).

**Idea**

Apply series of modifications on Q:

1) **GET Q**₁ if Q contains a function named "foo" then change its name throughout Q

2) **GET Q**₂ add a dummy function "foo" to Q₁

3) **GET Q**₃ modify Q₂ to store every character it prints in some array A.
4) Get $Q_4$ modify $Q_3$ so that after every "print" statement it checks array A to see if "Hello, World" has been printed yet — if so, call function foo.

Now let $R = Q_4$ and $Z = Y$

Clearly $(Q, Y)$ prints "Hello, World" $\iff (R, Z)$ calls function foo.

Thus we can use program $F$ to solve foo-problem on $(R, Z)$; thereby getting solution to hello-world problem on $(Q, Y)$

In effect given $F$, we can construct $H$ as follows:

![Diagram]

But since $H$ cannot exist, $F$ cannot exist!

**DONE**

**Reduction**

Problem $P_1$ takes input $I_1$ — known to be undecidable

Problem $P_2$ takes input $I_2$ — we would like to show is undecidable
**Reduction Idea**: Convert $I_1$ to $I_2$ such that answer to $P_1$ on input $I_1$ is "yes" if and only if answer to $P_2$ on input $I_2$ is "yes".

Now given solution program $S_2$ for $P_2$, can construct solution program $S_1$ for $P_1$.

![Diagram]

Of course since $S_1$ cannot exist, we obtain that $S_2$ cannot exist and $P_2$ is undecidable.

**Tricky Part** converting $I_1$ to $I_2$ — both are program Turing Machines.

**Problem**: Developing computation theory in terms of programs written in C is not elegant or even possible in general:

A. Have to worry about run-time environment and run-time errors

B. Language constructs are too complex

C. Need to worry about finite memory

D. "State" is too complicated.

E. You might complain that results apply only to C — what if there is a more general language where...
Turing Machines

A. Simple and universal programming language
B. Easy to describe state and runtime configuration
C. Can simulate any known computer or language
D. All attempts to describe powerful models of computation end up as special cases of Turing M/C.

Church-Turing Hypothesis: There is no more powerful model of computation than Turing Machines

Thus it models anything we can compute.

The Machine

![Diagram of a Turing Machine]

Finite state control

Read/write head

Infinite tape

Programming language? — Specify transitions

Move depends on

- Current state
- Symbol read by tape head

Effect

- New state
- Overwrite tape cell
- Move head — left/right.
OBSEERVE RELATIONSHIP TO REAL COMPUTERS
- CPU = FINITE STATE CONTROL
- MEMORY = TAPE

CAVEATS ABSTRACTION LOSES SOME ASPECTS
1) MEMORY IS NO LONGER RANDOM ACCESS
2) INSTRUCTION SET IS RATHER LIMITED
3) VARIOUS OTHER FEATURES ARE OMITTED.

HOWEVER T.M. CAPTURE ALL ESSENTIAL ASPECTS

IN FACT CHAPTER 8-6 SHOWS HOW T.M. CAN SIMULATE A REAL COMPUTER — AT SOME LOSS OF EFFICIENCY

PAYOFF SIMPLER ABSTRACTION ALLOWS US TO REASON BETTER!

FORMALLY

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, B, F) \]

- \( Q \) — FINITE STATES
- \( \Sigma \) — INPUT ALPHABET
- \( F \subseteq Q \) — FINAL STATES
- \( q_0 \in Q \) — INITIAL STATE
- \( \Gamma \) — TAPE ALPHABET
- \( B \in \Gamma \setminus \Sigma \) — BLANK SYMBOL

OBSEERVE

- \( Q, \Sigma, \Gamma \) — FINITE SETS
- \( \Sigma \subseteq \Gamma \) — SINCE INPUT \( w \) IS PROVIDED ON TAPE
- \( B \in \Gamma \setminus \Sigma \) — REST OF TAPE IS INITIALLY BLANK

INITIALLY — STATE \( q_0 \)
- TAPE CONTAINS \( w \) SURROUNDED BY BLANK.