University of Central Florida

Complexity Theory More Computability

Charles E. Hughes

COT6410 – Spring 2020 Notes

2SAT

A Subset of 3SAT
How hard?

2SAT

- We showed that 3SAT is NP Complete
- What about 2SAT (two variable per clause)?

Attacking 2SAT

First we need to convert a 2SAT instance to a different form, the so-called implicative normal form. Note that the expression $a \vee b$ is equivalent to

$\neg a \Rightarrow b \ \wedge \ \neg b \Rightarrow a$

(if one of the two variables is false, then the other one must be true).

We now construct a directed graph of these implications: for each variable x there will be two vertices x and $\neg x$. The edges will correspond to the implications.

2SAT Example

Let's look at an example in 2-CNF form:

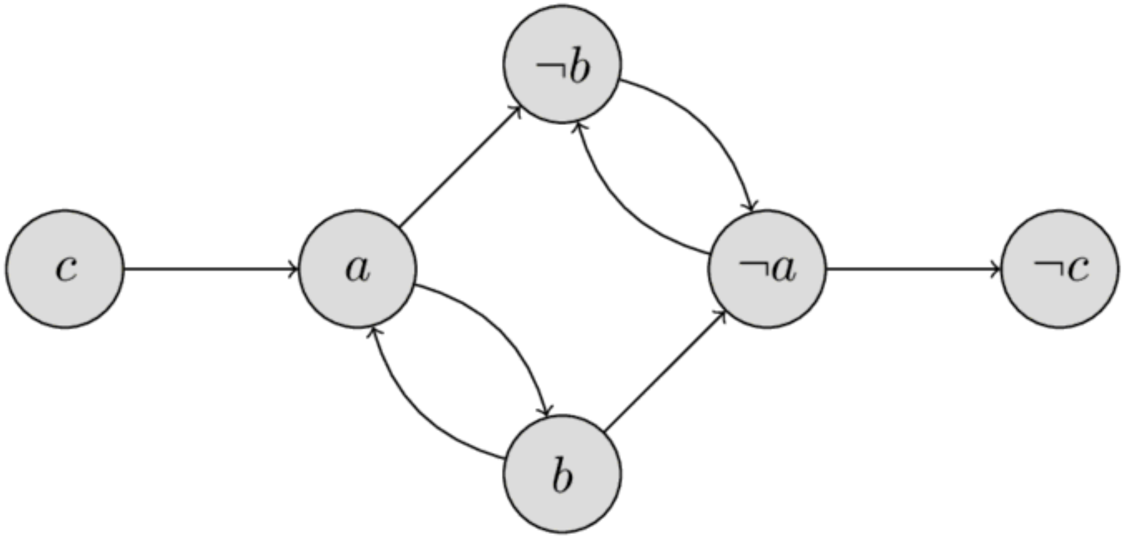
$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

The oriented graph will contain the following vertices:

$(a \vee \neg b)$	$(\neg a \vee b)$	$(\neg a \vee \neg b)$	$(a \vee \neg c)$
$\neg a \Rightarrow \neg b$	$a \Rightarrow b$	$a \Rightarrow \neg b$	$\neg a \Rightarrow \neg c$
$b \Rightarrow a$	$\neg b \Rightarrow \neg a$	$b \Rightarrow \neg a$	$c \Rightarrow a$

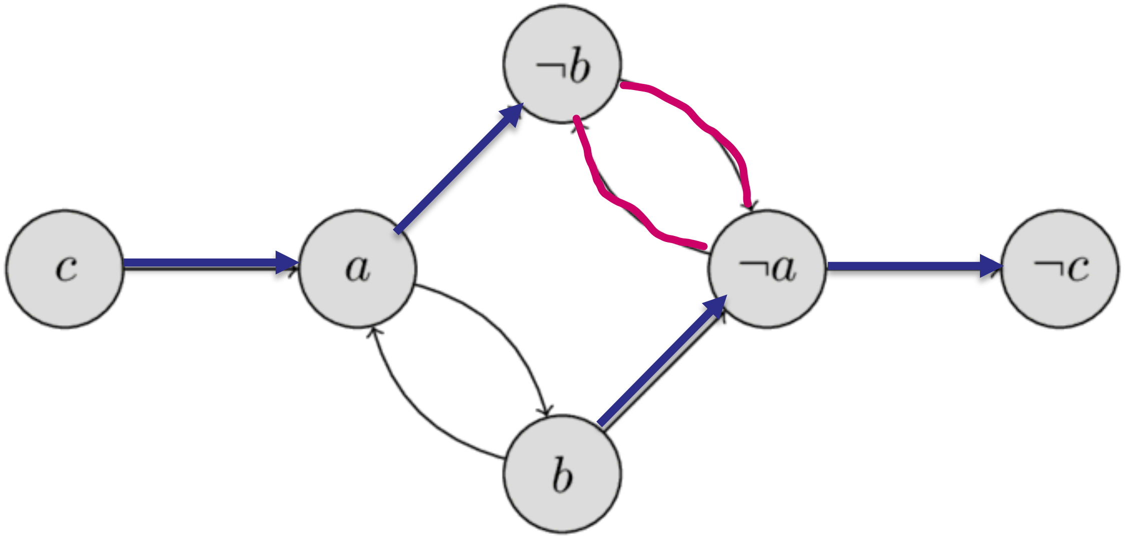
Graph from 2SAT Example

- If there is an edge a⇒b, then there also is an edge ¬b⇒¬a
- A contradiction exists if there is a cycle, for any variable x, that involves x and ¬x (means x ⇔ ¬x, which is a self-contradiction)
- What if there is path from some variable x to ¬x or vice versa?
- x⇒¬x can only be satisfied if x is false (¬x true)


$$\begin{tikzcd}
 & & \neg b \arrow[dr, bend left] & & \\
c \arrow[r] & a \arrow[ur] \arrow[dr, bend left] & & \neg a \arrow[ul, bend left] \arrow[r] & \neg c \\
 & & b \arrow[ul, bend left] \arrow[ur] & &
\end{tikzcd}$$

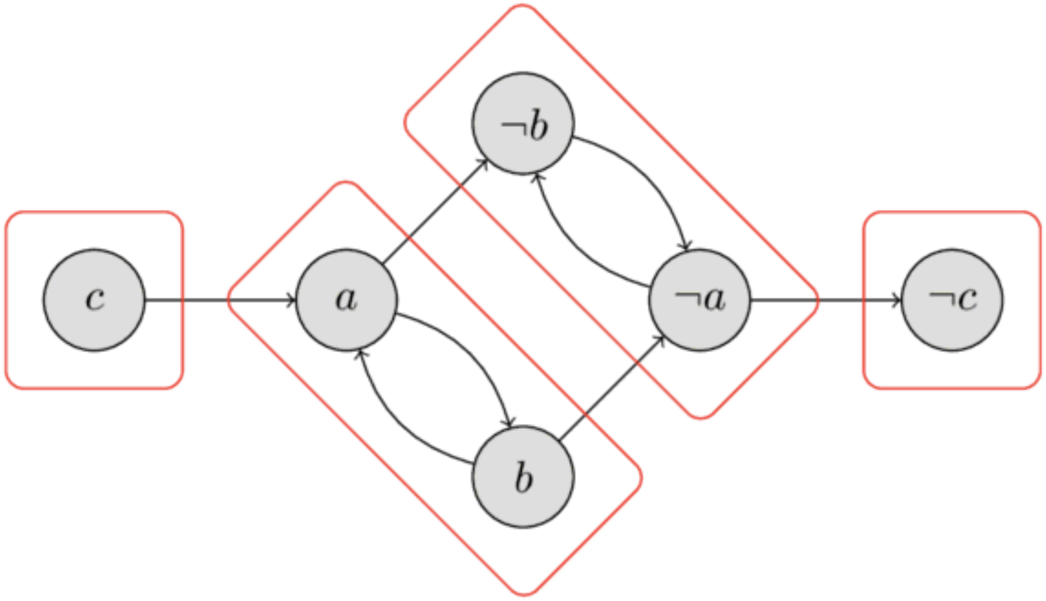
Finding a Solution for 2SAT

- Looking at our graph, c must be false, but so must a and b, as each has a path to its complement
- Note that, if a is true, b is true, and if a is false, b is false
- Fortunately, there are no cycles involving a variable and its complement, so we have a solution <a = F; b = F; c = F)
- The trick now is to discover that solution in an algorithmic manner



Strongly Connected Components (SCC)

- A directed graph is strongly connected if there is a path between all pairs of vertices.
- A strongly connected component (**SCC**) of a directed graph is a maximal strongly connected subgraph. For example, there are 3 SCCs in the graph we have been investigating.



Computing SCC

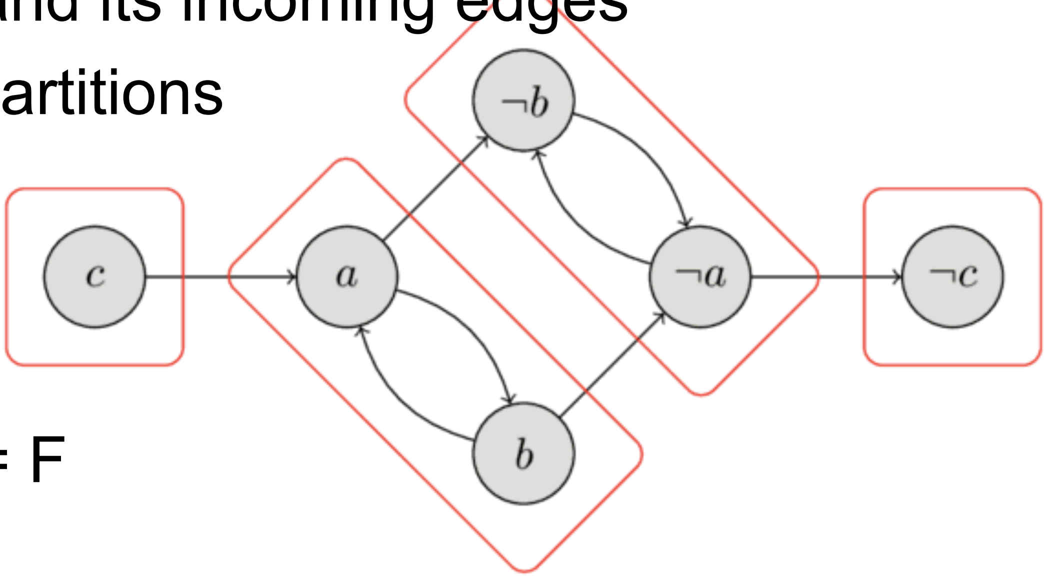
- There are several efficient linear time algorithms for finding the strongly connected components of a graph, based on depth first search
- The one commonly taught in Algorithm Design and Analysis is Tarjan’s

Mapping 2SAT to SCC

- In terms of the implication graph, two literals belong to the same strongly connected component whenever there exist chains of implications from one literal to the other and vice versa.
- Therefore, the two literals must have the same value in any satisfying assignment to the given 2-satisfiability instance. In particular, if a variable and its negation both belong to the same strongly connected component, the instance cannot be satisfied, because it is impossible to assign both of these literals the same value.
- This is a necessary and sufficient condition: a 2-CNF formula is satisfiable if and only if there is no variable that belongs to the same strongly connected component as its negation.

Solving SCC and 2SAT

- While some variable is not yet assigned
  - Start at a partition that has no outgoing edges
  - Assign true to all members of partition
  - Remove partition and its incoming edges
- Can also do DFS of partitions
- Either way, we get
  - ¬c = T
  - ¬a = ¬b = T
  - And so, a = b = c = F



Any Hard Problems Here?

- Minimum-ones 2SAT problem: Provide a satisfying assignment that sets a minimum number of variables to true.
- Uniform Min-Ones-2SAT is the restriction of Min-Ones-2SAT to input instances without mixed clauses (must be all positive or all negative literals in each clause)
- Positive Min-Ones-2SAT is the restriction of Uniform Min-Ones-2SAT to inputs containing only positive clauses (no negations)

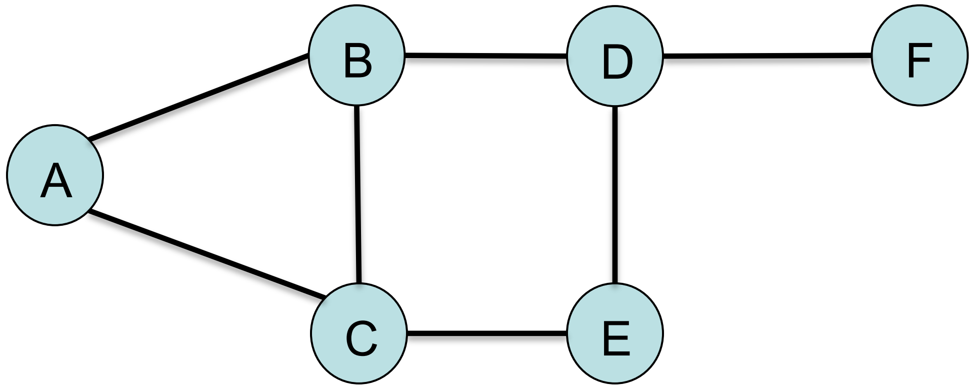
Uniform Min-Ones-2SAT

- Uniform Min-Ones-2SAT is NP-Hard as we can reduce Min-Vertex-Cover to it
- In fact, Uniform Min-Ones-2SAT is NP-Equivalent
- The best known (to me) uniform minimum-ones 2SAT problem algorithm has a running time of $O(1.21061^n)$ on a satisfiable 2SAT formula with n variables

Positive Min-Ones-2SAT

- Positive Min-Ones-2SAT is also equivalent to Min-Vertex-Cover and therefore NP-Equivalent as well
- This is interesting as the problem of determining haplotype classifications and propensity for certain genetic diseases can be mapped onto Positive Min-Ones-2SAT

VC to Positive Min-Ones-2SAT



Can we cover all edges with just 3 vertices?

$(A\vee B)(A\vee C),(B\vee C),(B\vee D),(C\vee E),(D\vee E),(D\vee F)$
Is minimum positive assignment 3 or fewer?
B,C,D works and tells us which vertices to choose to 3 cover above
If we added edge between E and F, the min would be 4 and we would require 4 vertices to cover all edges and 4 variables set to true
This shows Positive Min-Ones-2SAT is NP-Hard

Positive Min-Ones-2SAT to VC

- Associate every variable with a vertex
- If ($v1 \vee v2$) is a clause, add an edge between v1 and v2 in graph
- Now to find min, start with n/2, where we have n variables and do a binary search for min using oracle for VC
- Max number of queries of VC oracle is just $\log_2 n$ so this is NP-Easy and therefore NP-Equivalent

Propositional Calculus

Axiomatizable Fragments

Propositional Calculus

- Unquantified logical expressions
- Essentially Boolean algebra
- Goal is to reason about propositions
- Often interested in determining
  - Is a well-formed formula (wff) a tautology?
  - Is a wff refutable (unsatisfiable)?
  - Is a wff satisfiable? (classic NP-complete)

Tautology and Satisfiability

- The classic approaches are:
  - Truth Table
  - Axiomatic System (axioms and inferences)
- Truth Table
  - Clearly exponential in number of variables
- Axiomatic Systems Rules of Inference
  - Substitution and Modus Ponens
  - Resolution / Unification

Proving Consequences

- Start with a set of axioms (all tautologies)
- Using substitution and MP
  $(P,\ P \supset Q \Rightarrow Q)$
  derive consequences of axioms (also tautologies, but just a fragment of all)
- Can create complete sets of axioms
- Need 3 variables for associativity, e.g.,
  $(p1 \vee p2) \vee p3 \quad \supset \quad p1 \vee (p2 \vee p3)$

Some Undecidables

- Given a set of axioms,
  - Is this set complete?
  - Given a tautology T, is T a consequent of axiom set?
- The above are even undecidable with one axiom and with only 2 variables. I will show this result shortly.

Refutation

- If we wish to prove that some wff, F, is a tautology, we could negate it and try to prove that the new formula is refutable (cannot be satisfied; contains a logical contradiction).
- This is often done using resolution.

Resolution

- Put formula in Conjunctive Normal Form (CNF)
- If have terms of conjunction
  $(P \lor Q)$, $(R \lor \sim Q)$
  then can determine that $(P \lor R)$
- If we ever get a null conclusion, we have refuted the proposition
- Resolution is not complete for derivation, but it is for refutation

Example Resolution

1. Premise: **A** $\vee$ $\neg$**I**
2. Premise: $\neg$ **W** $\vee$ **I**
3. Premise: $\neg$ **A**
4. Premise: **W** $\vee$ **I** $\vee$ **C**
5. Negation of Conclusion. $\neg$ **C** (prove **C** from above)

6. **W** $\vee$ **I**	L4, L5, resolution
7. **I**	L2, L6, resolution, idempotence
8. **A**	L1, L7, resolution
9. (**A** $\vee$ $\neg$ **A**)	L3, L8, resolution

*#9 is a contradiction and so we can conclude **FALSE** from above, meaning **C** follows logically from Premises*

Axioms for a System that Can Deduce Only Tautogies

- Must be tautologies
- Can be incomplete
- Might have limitations on them and on WFFs, e.g.,
  - Just implication
  - Only n variables
  - Single axiom

Simulating Machines

- Linear representations require associativity, unless all operations can be performed on prefix only (or suffix only)
- Prefix and suffix-based operations are single stacks and limit us to CFLs
- Can simulate Post normal Forms with just 3 variables – this is minimum when start with a string-based system

Dyadic PIPC

- Dyadic limits us to two variables
- PIPC means Partial Implicational Propositional Calculus, and limits us to implication as the only connective
- Partial just means we get a fragment
- Problems
  - Is fragment complete?
  - Can some chosen tautology **C** be derived by substitution and MP?

Living without Associativity

- Consider a two-stack model of a TM
- Might be able to use one variable for left stack and other for right
- Must find a way to encode a sequence as a composition of forms – that's the key to this simulation

Composition Encoding

- Consider $(p \supset p)$, $(p \supset (p \supset p))$, $(p \supset (p \supset (p \supset p)))$, …
  - No form is a substitution instance of any of the other, so they can't be confused
  - All are tautologies
- Consider $((X \supset Y) \supset Y)$
  - This is just $X \vee Y$

Encoding Stack

- Use $(p \supset p)$ as form of bottom of stack
- Use $(p \supset (p \supset p))$ as form for letter 0
- Use $(p \supset (p \supset (p \supset p)))$ as form for 1
- Etc.
- String 01 (reading top to bottom of stack) is
  - $(\; ((p \supset p) \supset ((p \supset p) \supset ((p \supset p) \supset (p \supset p)))) \supset$
    $((p \supset p) \supset ((p \supset p) \supset ((p \supset p) \supset (p \supset p)))) \supset$
    $((p \supset p) \supset ((p \supset p) \supset ((p \supset p) \supset (p \supset p)))))$

TM to Encode

- Tape alphabet {0,1}   0 is blank
- State set $\{q_1, q_2, \dots, q_m\}$
  - q_1 is start state
  - Machine halts if we reach a discriminant (state, scanned symbol) with no associated action

Encoding Functions

$\mathrm{I}(p)$ abbreviates $[p \supset p]$ // stack bottom

$\Phi_0(p)$ is $[p \supset \mathrm{I}(p)]$ which is $[p \supset [p \supset p]]$ // symbol 0

$\Phi_1(p)$ is $[p \supset \Phi_0(p)]$ // symbol 1

$\xi_1(p)$ is $[p \supset \Phi_1(p)]$ // helper 1

$\xi_2(p)$ is $[p \supset \xi_1(p)]$ // helper 2

$\xi_3(p)$ is $[p \supset \xi_2(p)]$ // helper 3

$\psi_1(p)$ is $[p \supset \xi_3(p)]$ // symbol q_1

$\psi_2(p)$ is $[p \supset \psi_1(p)]$ // symbol q_2

…

$\psi_m(p)$ is $[p \supset \psi_{m-1}(p)]$ // symbol q_m

Example TM ID

- Let a TM's ID be
  110 q_5 1101
- Tape could be represented by two stacks
  Stack 1 is right side, reading left to right
  q_5 1101
- Stack 2 is left side, reading right to left
  011

Excoded Example TM ID

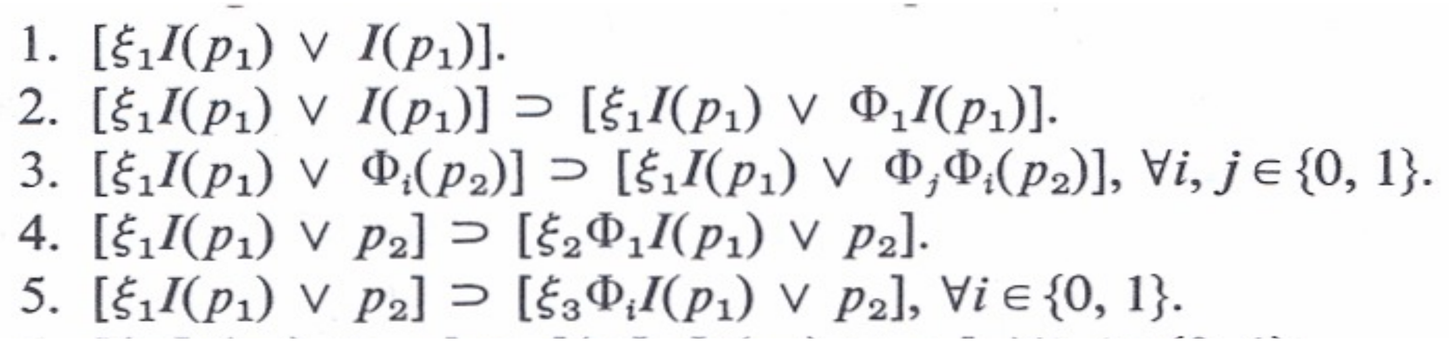
- q_5 1101 (stack 1, read left to right)
- 011 (stack 2, read left to right)
- $\psi_5(\Phi_1(\Phi_1(\Phi_0(\Phi_1(\mathbf{I} (p_1)))))) \vee \Phi_0(\Phi_1(\Phi_1(\mathbf{I} (p_2))))$
- Consider a Turing Table entry q_5 1 L q_2
- We could have an implication like
  $[\psi_5(\Phi_1(p_1)) \vee \Phi_0(p_2)] \supset [\psi_2(\Phi_0(\Phi_1(p_1))) \vee p_2]$
- Using substitution (see red) and MP
  $[\psi_5(\Phi_1(\Phi_1(\Phi_0(\Phi_1(\mathbf{I}(p_1)))))) \vee \Phi_0(\Phi_1(\Phi_1(\mathbf{I}(p_1))))] \Rightarrow$
  $[\psi_5(\Phi_0(\Phi_1(\Phi_1(\Phi_0(\Phi_1(\mathbf{I}(p_1))))))) \vee \Phi_1(\Phi_1(\mathbf{I}(p_1)))]$
- This mimics one step of forward computation

Running Backwards

- The simulation we show actually will mimic the TM running backwards so the rule on the previous page will actually be $[\psi_2(\Phi_0(\Phi_1(\mathbf{p}_1))) \vee \mathbf{p}_2] \supset [\psi_5(\Phi_1(\mathbf{p}_1)) \vee \Phi_0(\mathbf{p}_2)]$
- To kick things off, my rules want to allow me to deduce any arbitrary halting ID
- We use three helper forms to do this; they are $\xi_1(\mathbf{p})$, $\xi_2(\mathbf{p})$, and $\xi_3(\mathbf{p})$

ξ_1 Sets Up Stack 2

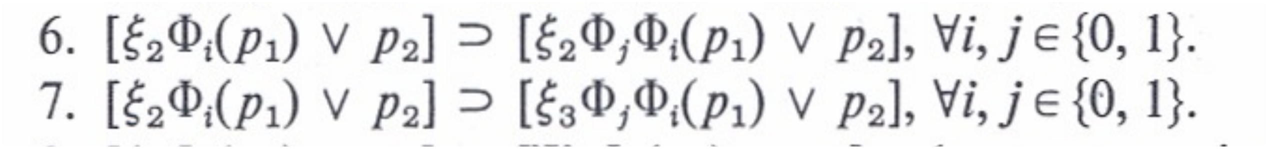
- The only axiom that does not involve a form for which MP can be applied is
  1. $[\xi_1 I(p_1) \vee I(p_1)]$
- The above reflects two empty stacks
- Using ξ_1 rules, we generate any and all possible left-hand sides of tape in stack 2
- This guarantees that left side is either empty (rule 4) or starts with a 1 (rule 2)
- If we apply rule 2 then rule 3 can expand the left side



1. $[\xi_1 I(p_1) \vee I(p_1)]$.
2. $[\xi_1 I(p_1) \vee I(p_1)] \supset [\xi_1 I(p_1) \vee \Phi_1 I(p_1)]$.
3. $[\xi_1 I(p_1) \vee \Phi_i(p_2)] \supset [\xi_1 I(p_1) \vee \Phi_j \Phi_i(p_2)], \forall i, j \in \{0, 1\}$.
4. $[\xi_1 I(p_1) \vee p_2] \supset [\xi_2 \Phi_1 I(p_1) \vee p_2]$.
5. $[\xi_1 I(p_1) \vee p_2] \supset [\xi_3 \Phi_i I(p_1) \vee p_2], \forall i \in \{0, 1\}$.

ξ_2 Starts Up Stack1

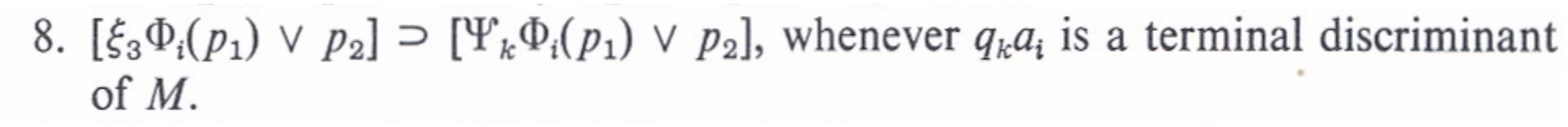
- Two possibilities follow
  Either Rule 4 replaces ξ_1 with ξ_2 and assures that the right side of tape (stack 1) has a 1 as its leftmost symbol
  Or Rule 5 replaces ξ_1 with ξ_3 and assures that the right side of tape (stack 1) has just a scanned symbol (can be a 0 or 1)
- If we use ξ_2 then rule 6 can expand the right side but at some point we use rule 7 to switch to ξ_3



6. $[\xi_2\Phi_i(p_1) \vee p_2] \supset [\xi_2\Phi_j\Phi_i(p_1) \vee p_2], \forall i, j \in \{0, 1\}.$
7. $[\xi_2\Phi_i(p_1) \vee p_2] \supset [\xi_3\Phi_j\Phi_i(p_1) \vee p_2], \forall i, j \in \{0, 1\}.$

ξ_3 Insures Terminal Discriminamt

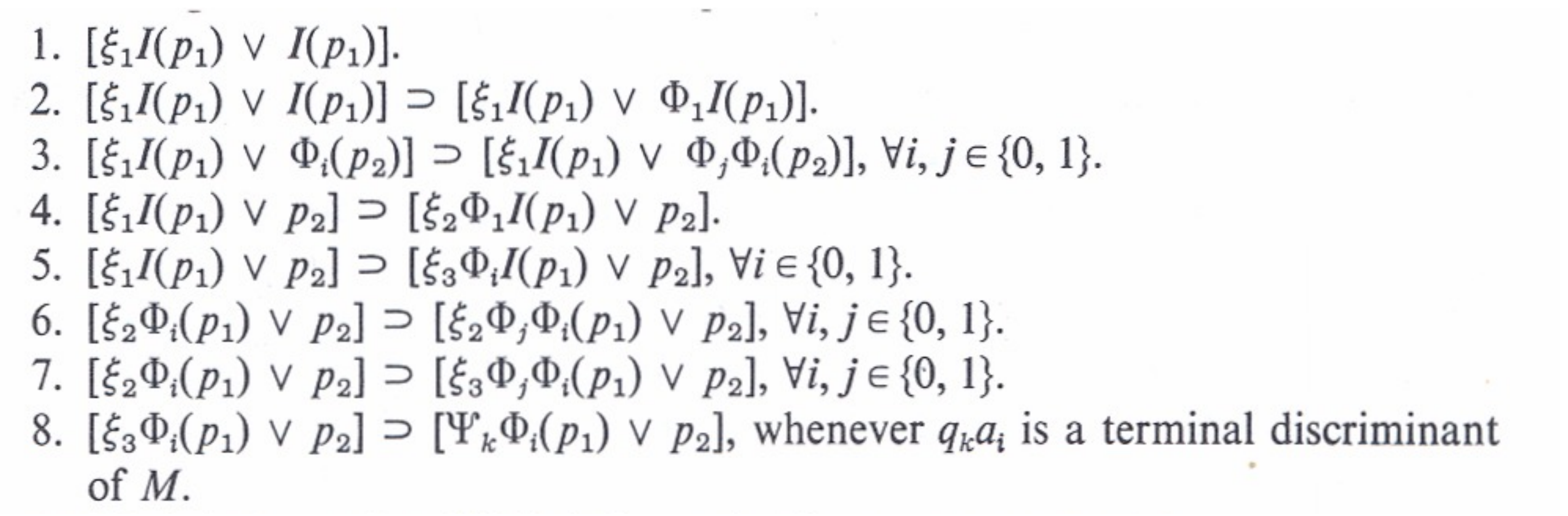
- Rule 8 replaces ξ_3 with any ψ_k such that $\mathbf{q_k i}$ halts (no rule) and i, represented by Φ_i, is on the top of stack 1 (new wff will be of form $\psi_k(\Phi_i(\mathbf{p_1})) \vee \mathbf{p_2}$
- This is the point where the simulation of the TM begins, except we run TM in reverse via rules 9-13 (and their subparts)
- While these rules can be a bit complex at first they are just the same ones we used to map a TM to a semi-Thue system or a PSG


8. $[\xi_3\Phi_i(p_1) \vee p_2] \supset [\Psi_k\Phi_i(p_1) \vee p_2]$, whenever $q_k a_i$ is a terminal discriminant of M.

Putting it Together

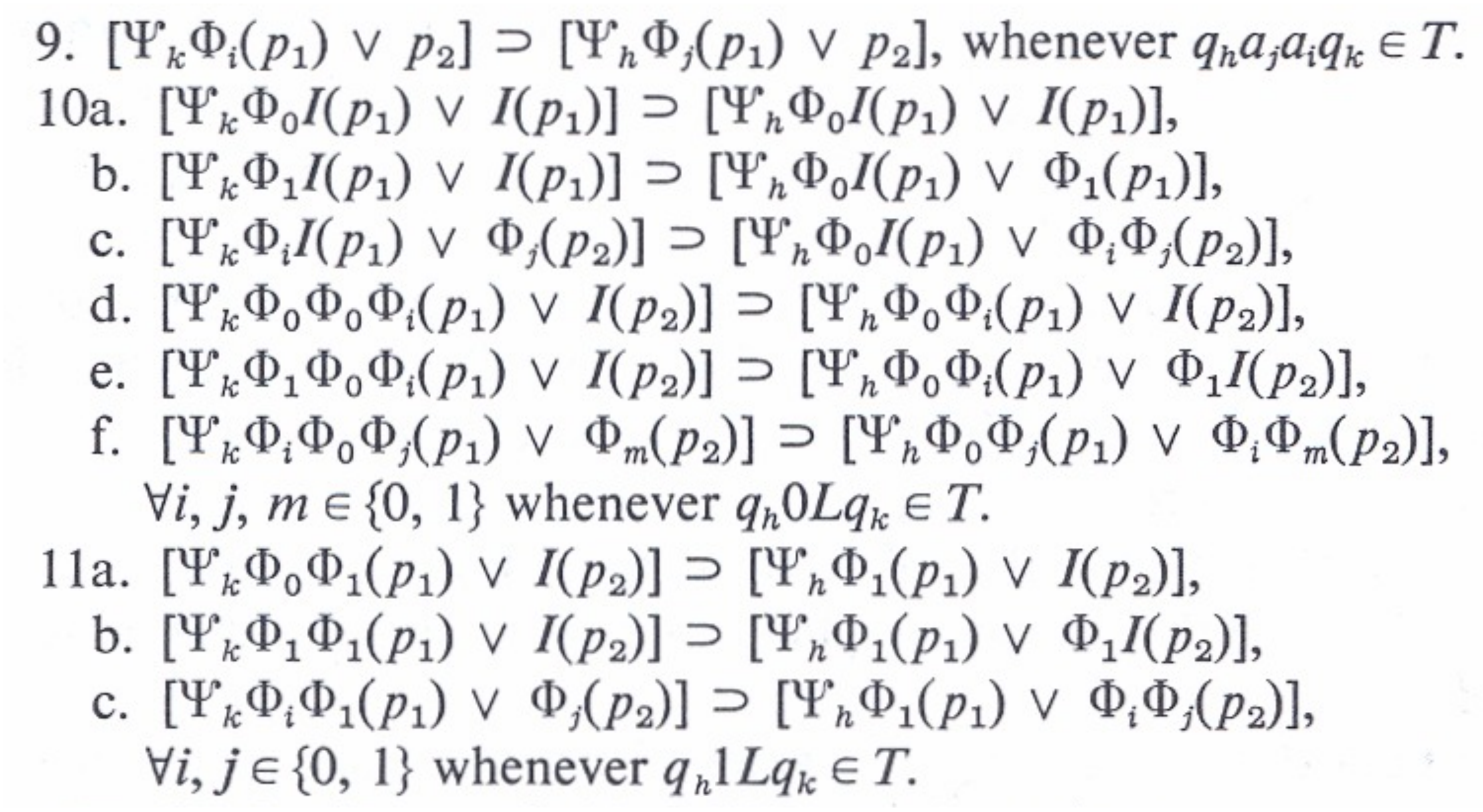
- The main point is that the axioms produce a bunch of items that are easy to check for validity (the stuff involving the forms ξ_1, ξ_2, and ξ_3 plus exactly those representations of starting IDs for which the TM halts
- If we could decide what Tautologies are producible by this Propositional System then we would be able to solve the Halting Problem for TMs
- This proves the deducibility problem for Fragments of the 2-Variable Implicational Calculus (PIPC) is unsolvable
- This is true even though two variables are insufficient to represent the basic notion of associativity!!!

Creating Terminal IDs



1. $[\xi_1 I(p_1) \vee I(p_1)]$.
2. $[\xi_1 I(p_1) \vee I(p_1)] \supset [\xi_1 I(p_1) \vee \Phi_1 I(p_1)]$.
3. $[\xi_1 I(p_1) \vee \Phi_i(p_2)] \supset [\xi_1 I(p_1) \vee \Phi_j \Phi_i(p_2)], \forall i, j \in \{0, 1\}$.
4. $[\xi_1 I(p_1) \vee p_2] \supset [\xi_2 \Phi_1 I(p_1) \vee p_2]$.
5. $[\xi_1 I(p_1) \vee p_2] \supset [\xi_3 \Phi_i I(p_1) \vee p_2], \forall i \in \{0, 1\}$.
6. $[\xi_2 \Phi_i(p_1) \vee p_2] \supset [\xi_2 \Phi_j \Phi_i(p_1) \vee p_2], \forall i, j \in \{0, 1\}$.
7. $[\xi_2 \Phi_i(p_1) \vee p_2] \supset [\xi_3 \Phi_j \Phi_i(p_1) \vee p_2], \forall i, j \in \{0, 1\}$.
8. $[\xi_3 \Phi_i(p_1) \vee p_2] \supset [\Psi_k \Phi_i(p_1) \vee p_2]$, whenever $q_k a_i$ is a terminal discriminant of M.

Reversing Print and Left



9. $[\Psi'_k\Phi_i(p_1) \vee p_2] \supset [\Psi'_h\Phi_j(p_1) \vee p_2]$, whenever $q_h a_j a_i q_k \in T$.

10a. $[\Psi'_k\Phi_0 I(p_1) \vee I(p_1)] \supset [\Psi'_h\Phi_0 I(p_1) \vee I(p_1)]$,

b. $[\Psi'_k\Phi_1 I(p_1) \vee I(p_1)] \supset [\Psi'_h\Phi_0 I(p_1) \vee \Phi_1(p_1)]$,

c. $[\Psi'_k\Phi_i I(p_1) \vee \Phi_j(p_2)] \supset [\Psi'_h\Phi_0 I(p_1) \vee \Phi_i\Phi_j(p_2)]$,

d. $[\Psi'_k\Phi_0\Phi_0\Phi_i(p_1) \vee I(p_2)] \supset [\Psi'_h\Phi_0\Phi_i(p_1) \vee I(p_2)]$,

e. $[\Psi'_k\Phi_1\Phi_0\Phi_i(p_1) \vee I(p_2)] \supset [\Psi'_h\Phi_0\Phi_i(p_1) \vee \Phi_1 I(p_2)]$,

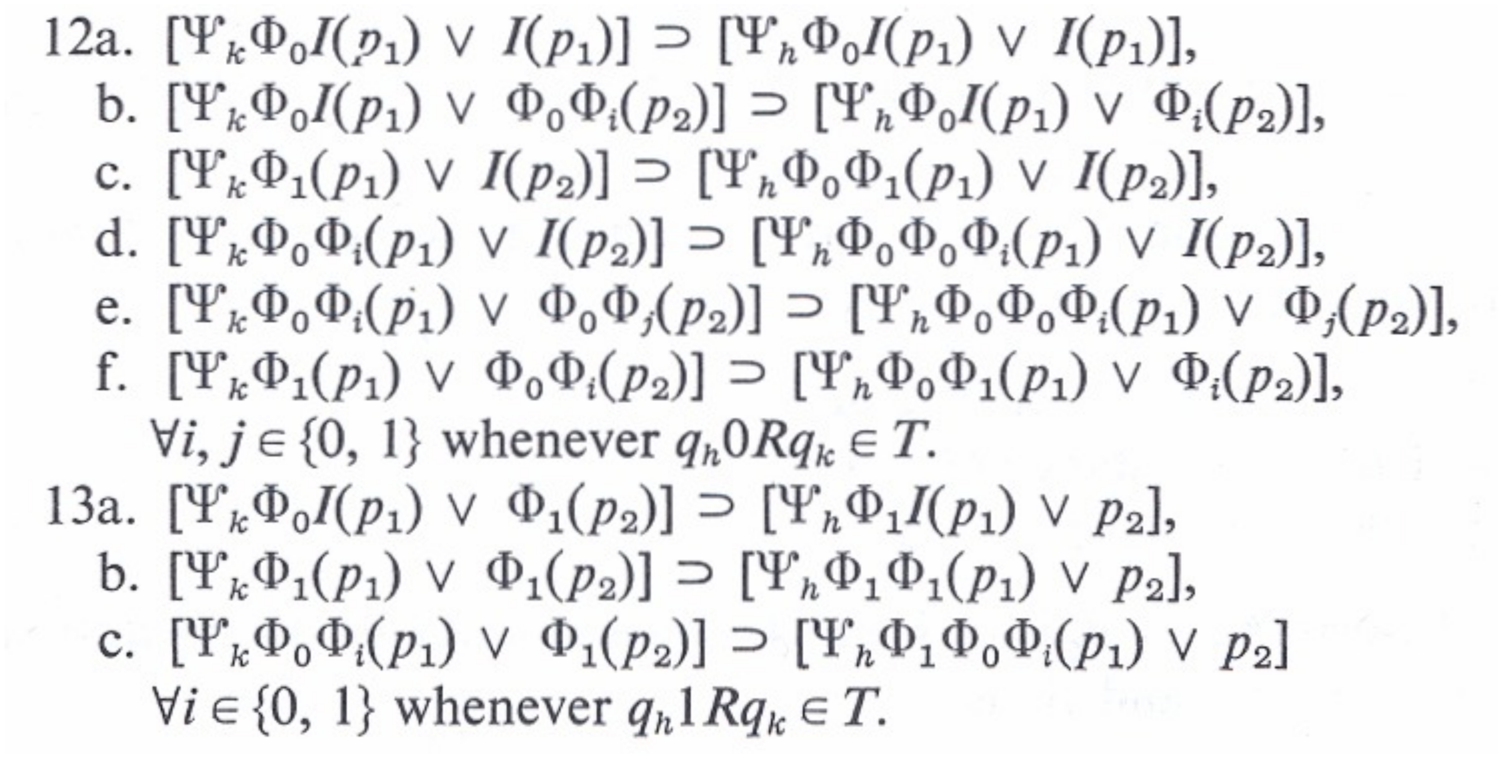
f. $[\Psi'_k\Phi_i\Phi_0\Phi_j(p_1) \vee \Phi_m(p_2)] \supset [\Psi'_h\Phi_0\Phi_j(p_1) \vee \Phi_i\Phi_m(p_2)]$,
$\forall i, j, m \in \{0, 1\}$ whenever $q_h 0 L q_k \in T$.

11a. $[\Psi'_k\Phi_0\Phi_1(p_1) \vee I(p_2)] \supset [\Psi'_h\Phi_1(p_1) \vee I(p_2)]$,

b. $[\Psi'_k\Phi_1\Phi_1(p_1) \vee I(p_2)] \supset [\Psi'_h\Phi_1(p_1) \vee \Phi_1 I(p_2)]$,

c. $[\Psi'_k\Phi_i\Phi_1(p_1) \vee \Phi_j(p_2)] \supset [\Psi'_h\Phi_1(p_1) \vee \Phi_i\Phi_j(p_2)]$,
$\forall i, j \in \{0, 1\}$ whenever $q_h 1 L q_k \in T$.

Reversing Right



12a. $[\Psi_k \Phi_0 I(p_1) \vee I(p_1)] \supset [\Psi_h \Phi_0 I(p_1) \vee I(p_1)]$,

b. $[\Psi_k \Phi_0 I(p_1) \vee \Phi_0 \Phi_i(p_2)] \supset [\Psi_h \Phi_0 I(p_1) \vee \Phi_i(p_2)]$,

c. $[\Psi_k \Phi_1(p_1) \vee I(p_2)] \supset [\Psi_h \Phi_0 \Phi_1(p_1) \vee I(p_2)]$,

d. $[\Psi_k \Phi_0 \Phi_i(p_1) \vee I(p_2)] \supset [\Psi_h \Phi_0 \Phi_0 \Phi_i(p_1) \vee I(p_2)]$,

e. $[\Psi_k \Phi_0 \Phi_i(p_1) \vee \Phi_0 \Phi_j(p_2)] \supset [\Psi_h \Phi_0 \Phi_0 \Phi_i(p_1) \vee \Phi_j(p_2)]$,

f. $[\Psi_k \Phi_1(p_1) \vee \Phi_0 \Phi_i(p_2)] \supset [\Psi_h \Phi_0 \Phi_1(p_1) \vee \Phi_i(p_2)]$,

$\forall i, j \in \{0, 1\}$ whenever $q_h 0 R q_k \in T$.

13a. $[\Psi_k \Phi_0 I(p_1) \vee \Phi_1(p_2)] \supset [\Psi_h \Phi_1 I(p_1) \vee p_2]$,

b. $[\Psi_k \Phi_1(p_1) \vee \Phi_1(p_2)] \supset [\Psi_h \Phi_1 \Phi_1(p_1) \vee p_2]$,

c. $[\Psi_k \Phi_0 \Phi_i(p_1) \vee \Phi_1(p_2)] \supset [\Psi_h \Phi_1 \Phi_0 \Phi_i(p_1) \vee p_2]$

$\forall i \in \{0, 1\}$ whenever $q_h 1 R q_k \in T$.

Constant time:
Not amenable to Rice's

Constant Time

- **CTime = { M | ∃K [M** halts in at most **K** steps independent of its starting configuration **] }**
- **RT** cannot be shown undecidable by Rice's Theorem as it breaks property 2
  - Choose **M1** and **M2** to each Standard Turing Compute (STC) **ZERO**
  - **M1** is **R** (move right to end on a zero)
  - **M2** is $\mathcal{L}\ \mathcal{R}$ **R** (time is dependent on argument)
  - **M1** is in **CTime; M2** is not , but they have same I/O behavior, so **CTime** does not adhere to property 2

Quantifier Analysis

- **CTime = { M | ∃K ∀C [STP(M, C, K)] }**
- This would appear to imply that **CTime** is not even re. However, a TM that only runs for **K** steps can only scan at most **K** distinct tape symbols. Thus, if we use unary notation, **CTime** can be expressed
- **CTime = { M | ∃K** $\forall C_{|C| \le K}$ **[STP(M, C, K)] }**
- We can dovetail over the set of all TMs, **M**, and all **K**, listing those **M** that halt in constant time.

Mortal Turing Machines

- A TM, ***M***, is **mortal** if it halts on all initial IDs, whether the tape is finitely or infinitely marked.
- A TM is **immortal** if it is not mortal, that is, if there some starting configuration, with the tape either finitely or infinitely marked, on which it does not halt
- The possibility of infinitely marked tapes is essential to the idea of mortality

Complexity of CTime

- Can show it is equivalent to the **Mortality Problem** for TM's with **Infinite Tapes** (not unbounded but truly infinite and potentially infinitely marked)
- This was shown in 1966 to be undecidable*.
- It was also shown to be re, just as we have done so for **CTime**.
- Details Later

*P.K. Hooper, The undecidability of the Turing machine immortality problem, ***J. Symbolic Logic 31*** (1966) 219-234.

Finite Convergence for Concatenation of Context-Free Languages

Relation to Real-Time (Constant Time) Execution

Powers of CFLs

Let G be a context free grammar.

Consider $L(G)^n$

Question1: Is $L(G) = L(G)^2$?

Question2: Is $L(G)^n = L(G)^{n+1}$, for some finite $n>0$?

These questions are both undecidable.

Think about why question1 is as hard as whether or not L(G) is Σ^*.

Question2 requires much more thought.

$L(G) = L(G)^2$?

- **The problem to determine if $L = \Sigma^*$ is Turing reducible to the problem to decide if $L \bullet L \subseteq L$, so long as L is selected from a class of languages C over the alphabet Σ for which we can decide if $\Sigma \cup \{\lambda\} \subseteq L$.**
- **Corollary 1:**
  The problem "is $L \bullet L = L$, for L context free or context sensitive?" is undecidable

$L(G) = L(G)^2$? is undecidable

- Question: Does $L \bullet L$ get us anything new?
  - i.e., Is $L \bullet L = L$?
- Membership in a CFL is decidable.
- Claim is that $L = \Sigma^*$ iff
  (1) $\Sigma \cup \{\lambda\} \subseteq L$; and
  (2) $L \bullet L = L$
- Clearly, if $L = \Sigma^*$ then (1) and (2) trivially hold.
- Conversely, we have $\Sigma^* \subseteq L^* = \cup_{n\geq 0} L^n \subseteq L$
  - first inclusion follows from (1); second from (2)

Finite Power Problem

- The problem to determine, for an arbitrary context free language L, if there exist a finite n such that $L^n = L^{n+1}$ is undecidable.
- Let M be some Turing Machine
- $L_1 = \{ C_1\# C_2^R \$ \mid C_1, C_2$ are configurations $\}$,
- $L_2 = \{ C_1\#C_2^R\$C_3\#C_4^R \ \dots \ \$C_{2k-1}\#C_{2k}^R\$ \mid$ where $k \geq 1$ and, for some i, $1 \leq i < 2k$, $C_i \Rightarrow_M C_{i+1}$ is false $\}$,
- $L = L_1 \cup L_2 \cup \{\lambda\}$.

Undecidability of $\exists n\ L^n = L^{n+1}$

- L is context free.
- Any product of L_1 and L_2, which contains L_2 at least once, is L_2. For instance, $L_1 \bullet L_2 = L_2 \bullet L_1 = L_2 \bullet L_2 = L_2$.
- This shows that $(L_1 \cup L_2)^n = L_1^n \cup L_2$.
- Thus, $L^n = \{\lambda\} \cup L_1 \cup L_1^2 \ldots \cup L_1^n \cup L_2$.
- Analyzing L_1 and L_2 we see that $L_1^n \cup L_2 \neq L_2$ just in case there is a word $C_1 \# C_2^R \$ C_3 \# C_4^R \ldots \$ C_{2n-1} \# C_{2n}^R \$$ in L_1^n that is not also in L_2.
- But then there is some valid trace of length 2n.
- L has the finite power property iff M executes in constant time.

Missing Step

- We have that **CT** (Constant-Time) is many-one reducible to Finite Power Problem (**FPC**) for CFLs
- This means that if **CT** is unsolvable, so is **FPC** for CFLs.
- However, we still lack a proof that **CT** is unsolvable. To achieve that we actually start with the 1966 result* that the mortality problem for TMs with potentially **infinite** initial tape markings is re/non-recursive
  Note that the uniform halting problem for TMs with **finite** initial tape markings is not even re – This is **TOTAL**

*P.K. Hooper, The undecidability of the Turing machine immortality problem, ***J. Symbolic Logic 31*** (1966) 219-234.

Infinite Tape Markings

- If a TM halts for all tape markings, even if the TM’s initial tape is infinitely marked, then there is some fixed maximum amount of the tape that the machine can traverse
- Why is the above so?
- Well, informally, if there was no bound built into the TM’s table then it would be at the mercy of its data to decide when to stop and that would lead a search for a zero (a divider between items on the tape) to take an infinite amount of time

Uniformly Halting

- A TM, ***M***, uniformly halts if there is some **n**, dependent only on ***M***, such that ***M*** halts in at most **n** steps no matter what initial finite input it is given
- Note that this concepts is restricted to normal TMs that start with a finitely marked tape
- Clearly, a TM that uniformly halts runs in constant time

T uniformly halts iff T is mortal

- Let ***T*** be a TM that does not uniformly halt. If any finite ID does not lead to a halt, then clearly ***T*** is immortal.
- Assume then that ***T*** does not uniformly halt but all finite ID's cause it to halt.
- Let $\mathcal{J}$ be the set of all ID's such that, for each $I \in \mathcal{J}$, when ***T*** starts in ***I*** it will eventually scan each square of the tape containing a symbol of ***I*** before it scans a square not containing a symbol of ***I***.
- Let $\{\mathbf{q_1}, \ldots, \mathbf{q_m}\}$ be the states of ***T***. We define a forest of **m** trees, one for each state of ***T***, such that the **j**-th tree has root $\mathbf{q_j}$.
- If $I_0, I_1 \in \mathcal{J}$, and $\mathbf{q_j}$ is a symbol of I_0 and I_1, and $I_1 = \sigma I_0$ or $I_1 = I_0 \sigma$, where σ is a tape symbol, then I_0 is a parent of I_1 in the **j**-th tree.
- Note that when ***T*** starts in I_1, the square containing σ is scanned after every other square of I_1 but before any square not in I_1.

T uniformly halts iff T is mortal

- Since ***T*** does not uniformly halt but every finite ID causes it to halt, at least one of the trees of the forest must be infinite.
- The degree of each vertex in each tree is finite (it is bounded by the number of tape symbols). By Koenig's Infinity Lemma, at least one of the trees must have an infinite branch. Therefore, there exists an infinite ID which causes ***T*** to travel an infinite distance on the tape. It follows that ***T*** is immortal.

Undecidability of Finite Convergence for Operators on Formal Languages

Relation to Real-Time (Constant Time) Execution

Simple Operators

- Concatenation
  - $A \bullet B = \{ xy \mid x \in A \;\&\; y \in B \}$
- Insertion
  - $A \triangleright B = \{ xyz \mid y \in A, xz \in B, x, y, z \in \Sigma^* \}$
  - Clearly, since x can be λ, $A \bullet B \subseteq A \triangleright B$

K-insertion

- $A \triangleright^{[k]} B = \{ x_1y_1x_2y_2 \ldots x_ky_kx_{k+1} \mid y_1y_2 \ldots y_k \in A, x_1x_2 \ldots x_kx_{k+1} \in B, x_i, y_j \in \Sigma^* \}$

- Clearly, $B \bullet A \subseteq A \triangleright^{[k]} B$, for all $k>0$

Iterated Insertion

- $A\ (1) \triangleright^{[n]} B = A \triangleright^{[n]} B$
- $A\ (k+1) \triangleright^{[n]} B = A \triangleright^{[n]} (A\ (k) \triangleright^{[n]} B)$

Shuffle

- Shuffle (product and bounded product)
  - $A \diamond B = \cup_{j \geq 1} A \triangleright^{[j]} B$
  - $A \diamond^{[k]} B = \cup_{1 \leq j \leq k} A \triangleright^{[j]} B = A \triangleright^{[k]} B$

- One is tempted to define shuffle product as $A \diamond B = A \triangleright^{[k]} B$ where

  $k = \mu\, y\, [\, A \triangleright^{[j]} B = A \triangleright^{[j+1]} B\,]$

  but such a k may not exist – in fact, we will show the undecidability of determining whether or not k exists

More Shuffles

- Iterated shuffle
  - $A \diamond^{0} B = A$
  - $A \diamond^{k+1} B = (A \diamond^{[k]} B) \diamond B$
- Shuffle closure
  - $A \diamond^{*} B = \cup_{k \geq 0} (A \diamond^{[k]} B)$

Crossover

- Unconstrained crossover is defined by $A \otimes_u B = \{ wz, yx \mid wx \in A \text{ and } yz \in B\}$
- Constrained crossover is defined by $A \otimes_c B = \{ wz, yx \mid wx \in A \text{ and } yz \in B, |w| = |y|, |x| = |z| \}$

Who Cares?

- People with no real life (me?)
- Insertion and a related deletion operation are used in biomolecular computing and dynamical systems
- Shuffle is used in analyzing concurrency as the arbitrary interleaving of parallel events
- Crossover is used in genetic algorithms

Some Known Results

- Regular languages, A and B
  - A • B is regular
  - A $\triangleright^{[k]}$ B is regular, for all k>0
  - A ◇ B is regular
  - A ◇* B is not necessarily regular
    - Deciding whether or not A ◇* B is regular is an open problem

More Known Stuff

- CFLs, A and B
  - A • B is a CFL
  - A ▷ B is a CFL
  - A $\triangleright^{[k]}$ B is not necessarily a CFL, for k>1
    - Consider A=a^nb^n; B = c^md^m and k=2
    - Trick is to consider (A $\triangleright^{[2]}$ B) ∩ a*c*b*d*
  - A ◇ B is not necessarily a CFL
  - A ◇* B is not necessarily a CFL
    - Deciding whether or not A ◇* B is a CFL is an open problem

Immediate Convergence

- $L = L^2$?
- $L = L \triangleright L$?
- $L = L \diamond L$?
- $L = L \diamond^* L$?
- $L = L \otimes_c L$?
- $L = L \otimes_u L$?

Finite Convergence

- $\exists k>0\ L^k = L^{k+1}$
- $\exists k\geq 0\ L\ (k) \triangleright L = L\ (k+1) \triangleright L$
- $\exists k\geq 0\ L \triangleright^{[\,k\,]} L = L \triangleright^{[\,k+1\,]} L$
- $\exists k\geq 0\ L \diamond^{k} L = L \diamond^{k+1} L$
- $\exists k\geq 0\ L\ (k) \otimes_c L = L\ (k+1) \otimes_c L$
- $\exists k\geq 0\ L\ (k) \otimes_u L = L\ (k+1) \otimes_u L$

- $\exists k\geq 0\ A\ (k) \triangleright B = A\ (k+1) \triangleright B$
- $\exists k\geq 0\ A \triangleright^{[\,k\,]} B = A \triangleright^{[\,k+1\,]} B$
- $\exists k\geq 0\ A \diamond^{k} B = A \diamond^{k+1} B$
- $\exists k\geq 0\ A\ (k) \otimes_c B = A\ (k+1) \otimes_c B$
- $\exists k\geq 0\ A\ (k) \otimes_u B = A\ (k+1) \otimes_u L$

Finite Power of CFG

- Let G be a context free grammar.
- Consider $L(G)^n$
- Question1: Is $L(G) = L(G)^2$?
- Question2: Is $L(G)^n = L(G)^{n+1}$, for some finite $n>0$?
- These questions are both undecidable.
- We showed that question1 is as hard as whether or not L(G) is Σ^*.
- Question2 required more work.

1981 Results

- Theorem 1:
  The problem to determine if $L = \Sigma^*$ is Turing reducible to the problem to decide if $L \bullet L \subseteq L$, so long as L is selected from a class of languages C over the alphabet Σ for which we can decide if $\Sigma \cup \{\lambda\} \subseteq L$.
- Corollary 1:
  The problem "is $L \bullet L = L$, for L context free or context sensitive?" is undecidable

Proof #1

- Question: Does $L \bullet L$ get us anything new?
  - i.e., Is $L \bullet L = L$?
- Membership in a CSL is decidable.
- Claim is that $L = \Sigma^*$ iff

  (1) $\Sigma \cup \{\lambda\} \subseteq L$; and

  (2) $L \bullet L = L$
- Clearly, if $L = \Sigma^*$ then (1) and (2) trivially hold.
- Conversely, we have $\Sigma^* \subseteq L^* = \cup_{n \geq 0} L^n \subseteq L$
  - first inclusion follows from (1); second from (2)

Subsuming •

- Let ⊕ be any operation that subsumes concatenation, that is $A \bullet B \subseteq A \oplus B$.
- Simple insertion is such an operation, since $A \bullet B \subseteq A \triangleright B$.
- Unconstrained crossover also subsumes •,
  $A \otimes_c B = \{ wz, yx \mid wx \in A \text{ and } yz \in B\}$

$L = L \oplus L$?

- Theorem 2:
  The problem to determine if $L = \Sigma^*$ is Turing reducible to the problem to decide if $L \oplus L \subseteq L$, so long as $L \bullet L \subseteq L \oplus L$ and L is selected from a class of languages C over Σ for which we can decide if $\Sigma \cup \{\lambda\} \subseteq L$.

Proof #2

- Question: Does $L \oplus L$ get us anything new?
  - i.e., Is $L \oplus L = L$?
- Membership in a CSL is decidable.
- Claim is that $L = \Sigma^*$ iff

  (1) $\Sigma \cup \{\lambda\} \subseteq L$; and

  (2) $L \oplus L = L$
- Clearly, if $L = \Sigma^*$ then (1) and (2) trivially hold.
- Conversely, we have $\Sigma^* \subseteq L^* = \cup_{n\geq 0} L^n \subseteq L$
  - first inclusion follows from (1); second from (1), (2) and the fact that $L \bullet L \subseteq L \oplus L$

Quotients of CFLs

Quotients of CFLs (Trace-Like Sequences)

Let L1 = L(G1) = { $ # Y_0 # Y_1 # Y_2 # Y_3 # … # Y_{2j} # Y_{2j+1} # }
where $Y_{2i} \Rightarrow Y_{2i+1}$, $0 \leq i \leq j$.
This checks the even/odd steps of an even length computation.

Now, let L2=L(G2)=
{X_0 $ # X_0 # X_1 # X_2 # X_3 # X_4 # … # X_{2k-1} # X_{2k}# Z_0 #}
where $X_{2i-1} \Rightarrow X_{2i}$, $1 \leq i \leq k$ and Z_0 is a unique halting configuration.
This checks the odd/steps of an even length computation and includes an extra copy of the starting number prior to its $.

If a Turing Machine Trace

Let L1 = L(G1) = { $ # Y_0^R # Y_1 # Y_2^R # Y_3 # … # Y_{2j}^R # Y_{2j+1} # }
where $Y_{2i} \Rightarrow Y_{2i+1}$, $0 \le i \le j$.
This checks the even/odd steps of an even length computation.

Now, let L2=L(G2)=
{X_0 $ # X_0^R # X_1 # X_2^R # X_3 # X_4^R # … # X_{2k-1} # X_{2k}^R# Z_0 #}
where $X_{2i-1} \Rightarrow X_{2i}$, $1 \le i \le k$ and Z_0 is a unique halting configuration.

This checks the odd/steps of an even length computation and includes an extra copy of the starting number prior to its $.

Quotients of CFLs (results)

L1 = $\{ \$ \# Y_0 \# Y_1 \# Y_2 \# Y_3 \# Y_4 \# \ldots \# Y_{2k-1} \# Y_{2j} \# Y_{2j+1} \# \}$

L2 = $\{X0 \$ \# X_0 \# X_1 \# X_2 \# X_3 \# X_4 \# \ldots \# X_{2k-1} \# X_{2k} \# \; Z_0 \; \#\}$

Now, consider the quotient of L2 / L1 . The only way a member of L1 can match a final substring in L2 is to line up the \$ signs. But then they serve to check out the validity and termination of the computation. Moreover, the quotient leaves only the starting point (the one on which the machine halts.) Thus,

L2 / L1 = { X_0 | the system being traced halts}.

Since deciding the members of an re set is in general undecidable, we have shown that membership in the quotient of two CFLs is also undecidable.
Note: Intersection of two CFLs is a CSL but quotient of two CFLs is an re set and, in fact, all re sets can be specified by such quotients.

Quotients of CFLs (precise)

- Let (n, ((a1,b1,c1,d1) , … ,(ak,bk,ck,dk)) be some factor replacement system with residues. Define grammars G1 and G2 by using the 4k+4 rules

  G : $F_i \rightarrow 1^{ai}F_i1^{ci}$ | $1^{ai+bi}\#1^{ci+di}$ $\quad 1 \le i \le k$

  $T_1 \rightarrow \# F_i T_1$ | $\# F_i \#$ $\quad 1 \le i \le k$

  $A \rightarrow 1\ A\ 1$ | $\$ \#$

  $S_1 \rightarrow \$T_1$

  $S_2 \rightarrow A\ T_1 \# 1^{z0} \#$ $\quad Z_0$ is 0 for us

  G1 starts with S_1 and G2 with S_2

- Thus, using the notation of writing Y in place of 1^Y,

  L1 = L(G1) = { $\$ \#Y_0 \# Y_1 \# Y_2 \# Y_3 \# \ldots \# Y_{2j} \# Y_{2j+1} \#$ }

  where $Y_{2i} \Rightarrow Y_{2i+1}$, $0 \le i \le j$.

  This checks the even/odd steps of an even length computation.

  But, L2 = L(G2) = {$X_0 \$ \#X_0 \# X_1 \# X_2 \# X_3 \# X_4 \# \ldots \# X_{2k-1} \# X_{2k}\# Z_0 \#$ }

  where $X_{2i-1} \Rightarrow X_{2i}$, $1 \le i \le k$ and $X = X_0$

  This checks the odd/steps of an even length computation, and includes an extra copy of the starting number prior to its \$.

Summarizing Quotient

Now, consider the quotient **L2 / L1** where **L1** and **L2** are the CFLs on prior slide. The only way a member of **L1** can match a final substring in **L2** is to line up the **$** signs. But then they serve to check out the validity and termination of the computation. Moreover, the quotient leaves only the starting number (the one on which the machine halts.) Thus,

L2 / L1 = { X | the system F halts on zero }.

Since deciding the members of an re set is in general undecidable, we have shown that membership in the quotient of two CFLs is also undecidable.

A Complexity Summary

PSPACE

- PSPACE is set of problems solvable in polynomial space with unlimited time PSPACE = $\cup$ SPACE(n^k)
- PSPACE = co-PSPACE = NPSPACE
- PSPACE is a strict superset of CSLs
- PSPACE-Complete Problem is, given a regular expression E over Σ, does E denote all strings in Σ*?
- The above, while solvable, is potentially hard
- Another PSPACE-Complete problem is QSAT
- PSPACE is suspected to outside the P/NP hierarchy

EXPTIME and EXPSPACE

- EXPTIME is the set of problems solvable in $2^{p(n)}$ where is p is some polynomial.
- NEXPTIME is the set of problems solvable in $2^{p(n)}$ on a non-deterministic TM.
- EXPSPACE is set of problems solvable in $2^{p(n)}$ space and unbounded time

Elementary Functions

$$\begin{aligned}\mathsf{ELEMENTARY} &= \bigcup_{k\in\mathbb{N}} \text{k-EXP} \\ &= \mathsf{DTIME}(2^n) \cup \mathsf{DTIME}(2^{2^n}) \cup \mathsf{DTIME}(2^{2^{2^n}}) \cup \cdots\end{aligned}$$

Alternating TM (ATM)

- ATM adds to NDTM notation the notion where, for each state q, q has one of the following properties: (accept, reject, $\vee$, $\wedge$)
  - $\vee$ means mean accept the string if <u>any</u> final state reached after q is accepting
  - $\wedge$ means mean accept the string if <u>all</u> final states reached after q are accepting
- AP = PSPACE where AP is class of problems solvable in polynomial time on an ATM

QSAT, Petri Net, Presburger

- QSAT is solvable by an alternating TM in polynomial time and polynomial space
- As noted, before, QSAT is PSPACE-Complete
- Petri net reachability is EXPSPACE-hard and requires 2-EXPTIME
- Presburger arithmetic is at least in 2-EXPTIME, at most in 3-EXPTIME, and can be solved by an ATM with n alternating quantifiers in doubly exponential time

Complexity Hierarchy

- P ⊆ NP ⊆ PSPACE = NPSPACE ⊆ EXPTIME ⊆ NEXPTIME ⊆ EXPSPACE ⊈ 2-EXPTIME ⊈ 3-EXPTIME ⊈ … ⊈ ELEMENTARY ⊈ PRF ⊈ REC
- What if P ≠ EXPTIME; At least one of these is true
  - P ⊈ NP
  - NP ⊈ PSPACE
  - PSPACE ⊈ EXPTIME
- If NP ≠ NEXPTIME; At least of these is true
  - NP ⊈ PSPACE
  - PSPACE ⊈ EXPTIME
  - EXPTIME ⊈ NEXPTIME
    - Note that EXPTIME = NEXPTIME iff P=NP
    - Note that k-EXPTIME ⊈ (k+1)-EXPTIME, k>0
- What If PSPACE ≠ EXPSPACE; At least one of these is true
  - PSPACE ⊈ EXPTIME
  - EXPTIME ⊈ EXPSPACE