# COMP30026 Models of Computation

## Reducibility





Lecture Week 12. Par



Semester 2, 2021

# Undecidable Languages



In the last lecture we saw that

$$A = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$$

is undec



Tute exercise T12.1 asks you to prove that it follows that

$$Halt_{TM} = \left\{ \langle M, w\rangle \;\middle|\; \begin{array}{l} M \text{ is a Turi} \\ M \text{ halts w} \end{array} \right.$$



is also undecidable.

# At Least $A_{TM}$ Is Recognisable

Note that

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$$

is Turin

The reas urng machin

On input $\langle M, w\rangle$ $U$ simulates $M$ on inp

If $M$ enters its accept state, $U$ accepts.

If $M$ enters its reject state, $U$ rejects.

If $M$ never halts, neither does $U$.

# Hilbert's Tenth Problem



Mathematicians have this proud history of inventing algorithms and posing algorithmic challen



Here, for e
problem
Hilbert in 1900:



> *Find an algorithm that determines whethe many variables but with integer coefficient root.*

# Hilbert's Tenth Problem

70 years after Hilbert posed his tenth problem, based on work by J. Robinson, Y. Matiyasevich proved that there is no algorithm for it.

$\{p \mid p$ is a

undecid

binson

However

If the input polynomial $p$ has $k$ variable $x_1, \ldots, x_k$ then we can simply enumerate all integer $k$-tuples $(v_1, \ldots, v_k)$ and evaluate $p(v_1, \ldots, v_k)$, one by one. If $p(v_1, \ldots, v_k) = 0$, accept. We refer to this type of problem as semi-decidable.

Y. Matiyasevich

# Closure Properties



The set of Turing recognisable languages is closed under the regular operatio ment.

The set of d tions, and also u



Week 11 ute exercises explore some of these closur ore detail.

# Relating Decidability and Recognisability

**Theorem:** A language $L$ is decidable iff both $L$ and its complement $L^c$ are Turing recognisable.



**Proof:** If $L$ is decidable, clearly $L$ and also $L^c$ are recognisable.

Assume both $L$ and $L^c$ are recognisable by Turing machines $M_1$ and $M_2$.



A Turing machine $M$ can then take input $w$ in parallel. If $M_1$ accepts, so does $M$. If $M_2$ accepts, so does $M$.



Note that at least one of $M_1$ and $M_2$ is guaranteed to accept.

Hence $M$ decides $L$.

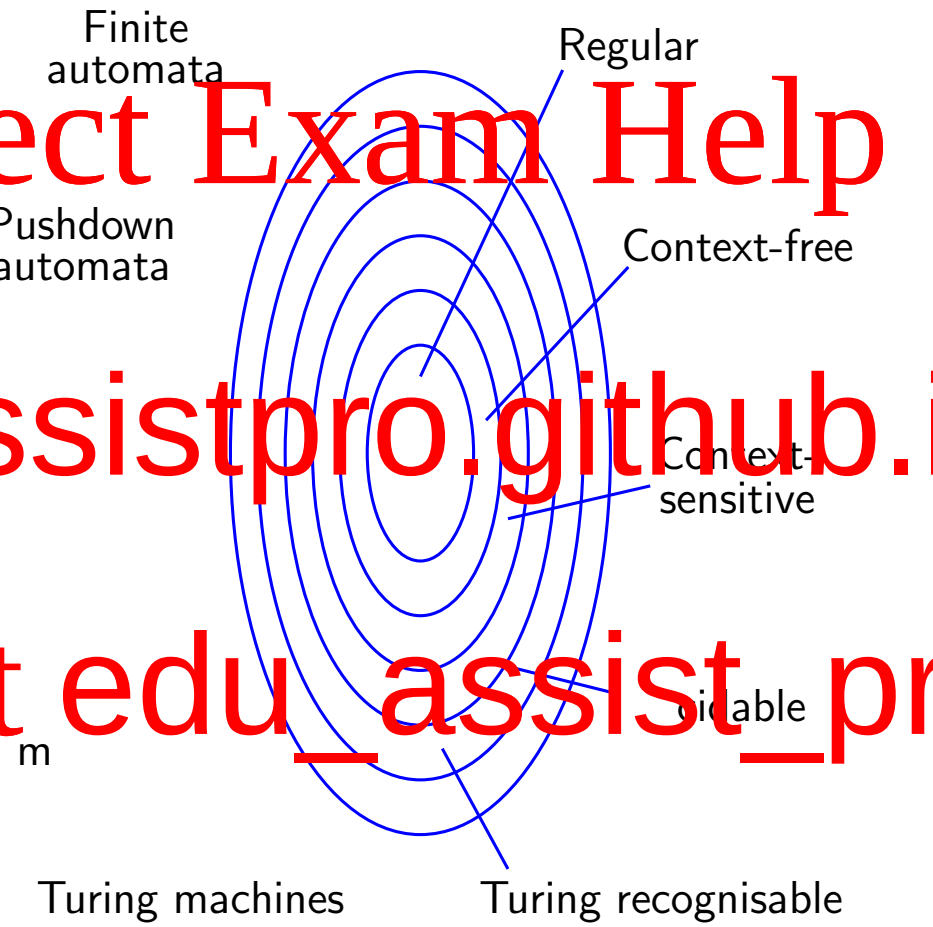
# A Non-Turing Recognisable Language

This gives us an example of a language which is not Turing recognisable: $(A_{TM})^c$, that is, the complement of $A$ .

Namely, recognis

If $(A_{TM})^c$ was also Turing recognisable, $A_{TM}$ would be decidable.

But we have shown that it isn't.



# Too Many Languages



As we have seen, the set $\{0,1\}^*$ of finite binary strings is a countable set.

Proble $\mathcal{B}$ of all infinite



That is interesting, because it follows that the set of all l ges over any finite non-empty alphabet $\Sigma$ is also uncoun



Namely, the set of all languages over $\Sigma$ is in a one-to-on correspondence with $\mathcal{B}$, as we now show.

# Too Many Languages

Let $s_1, s_2, s_3, \ldots$ be the standard enumeration of $\Sigma^*$.



For each language $A$ over $\Sigma$, there's a unique characteristic sequence $\chi_A \in \mathcal{B}$, whose $i$th bit is 1 iff $s_i \in A$:



$$\chi_A : \quad 0 \; 1 \; 0 \; 1 \; 1 \; 0 \; 0 \; 1 \; 0 \; \ldots$$



Hence we cannot put the set of all languages into a one-correspondence with the set of all Turing machine

That is, we could never hope to have a recogniser for each possible language.

# Reducibility

Let $P$ and $P'$ be decision problems with instances $p_i$ and $p'_i$.

$P$ can be reduced to $P'$ iff there is a Turing machine $M$:

---

such that

- $P$ reducible to $P'$ and $P'$ decidable
- $P$ reducible to $P'$ and $P$ undecida

So reducibility is useful for proving decidability and undecidability results.

# TM Emptiness Is Undecidable

**Theorem:**

$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

is undeci

**Proof:** $\langle M, w \rangle$

a Turing $M$

into $M'$ which recognises $L(M) \cap \{w\}$.

Here is what the new machine $M'$ does

1. If input $x$ is not $w$, reject.
2. Otherwise run $M$ on $w$ and accept if $M$ does.

# TM Emptiness Is Undecidable

Notice how $w$ has been "hard-wired" into $M'$: This machine is like $M$, but it has extra states added to perform the test against $w$.



Also note that



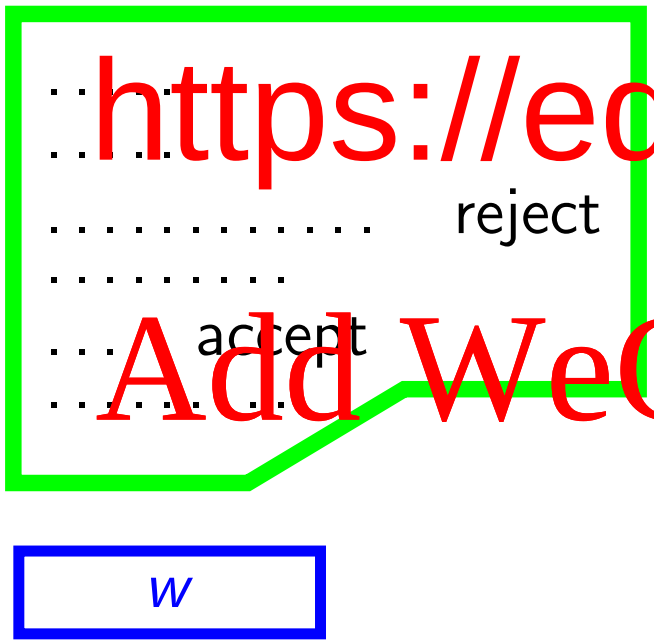
Here then is a decider for $A_{TM}$, using a dec



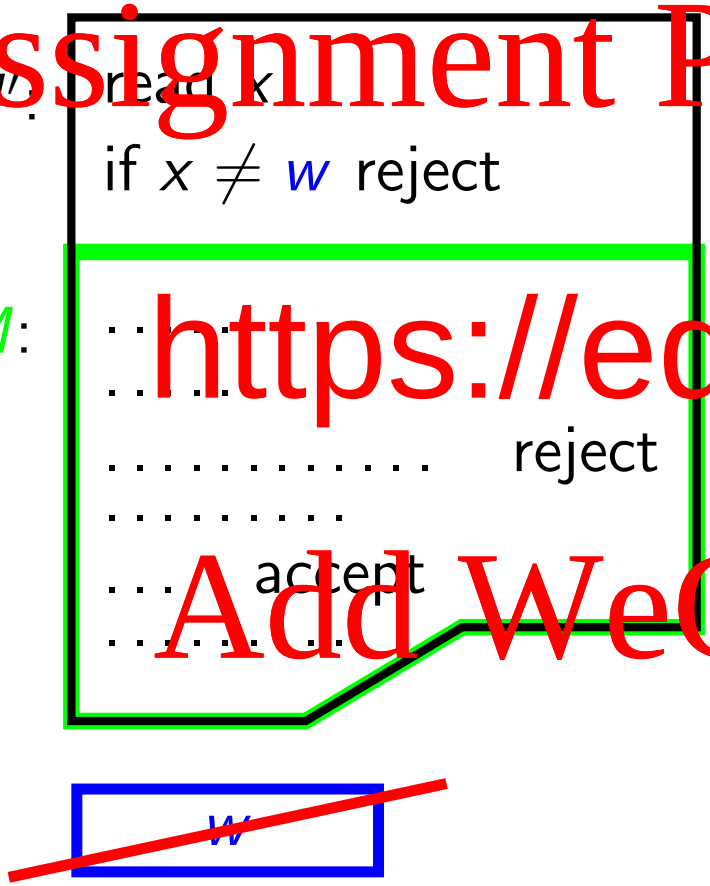
1. From input $\langle M, w \rangle$ construct $\langle M' \rangle$
2. Run $R$ on $\langle M' \rangle$.
3. If $R$ rejects, accept; if it accepts, reject.

# The Argument in Pictures

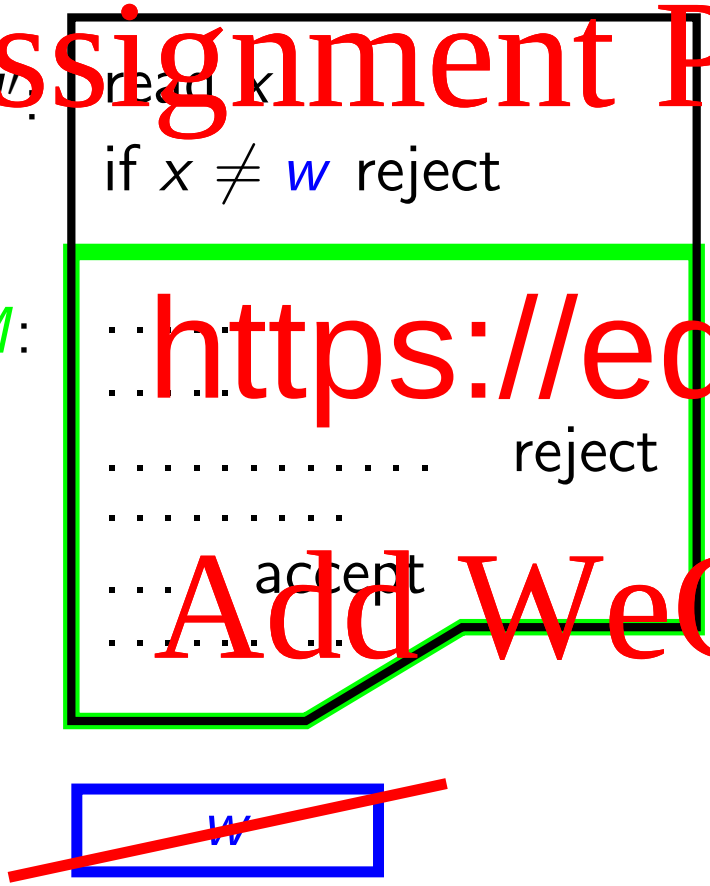


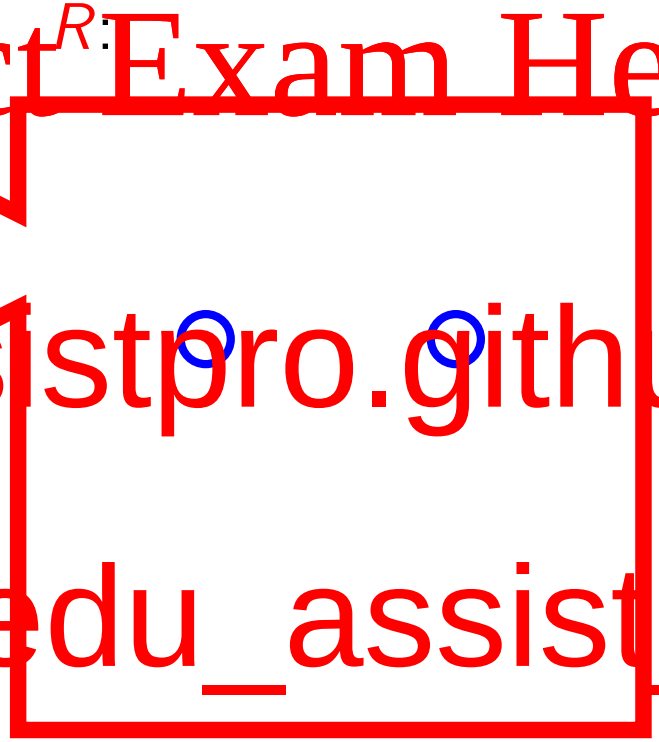


# The Argument in Pictures

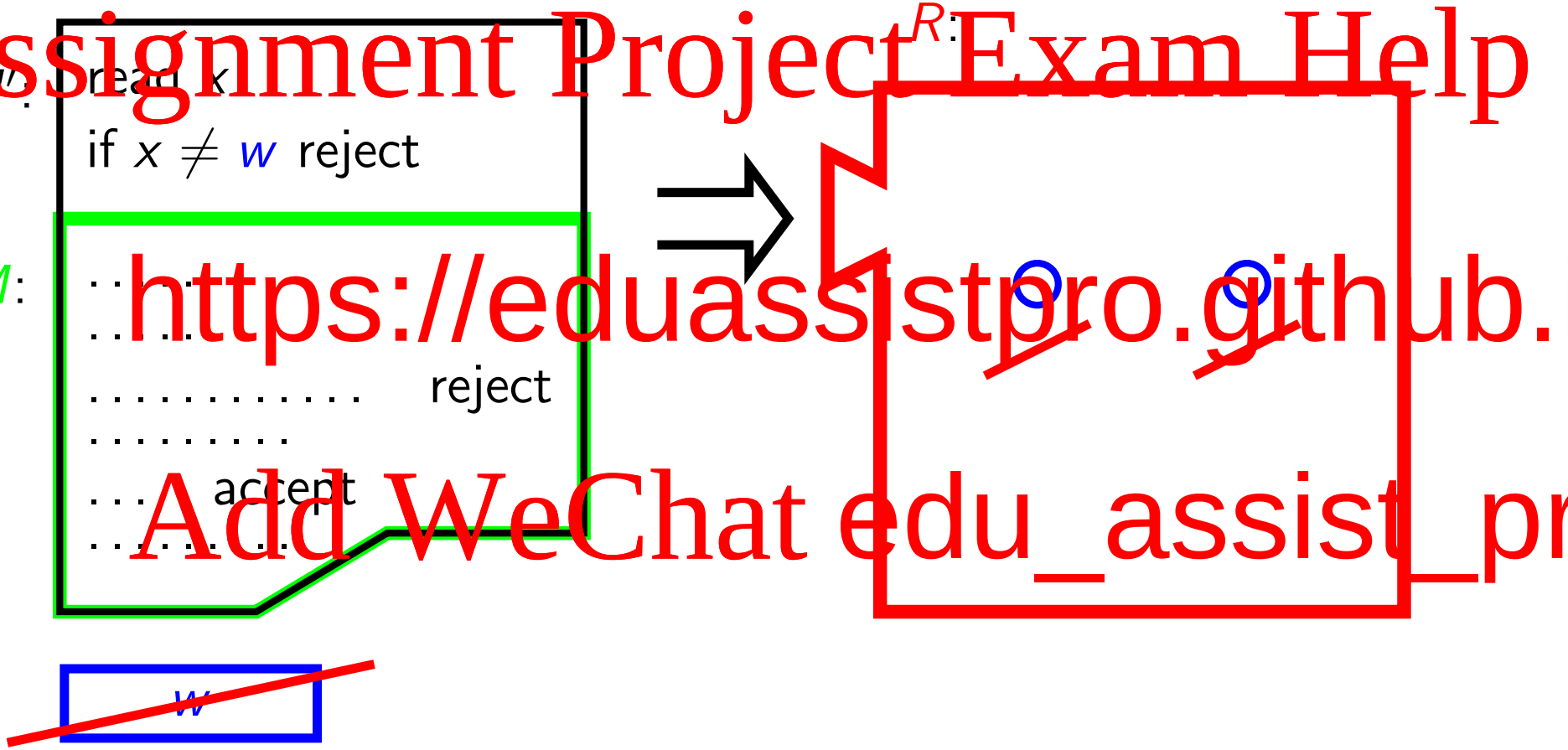


# The Argument in Pictures



# The Argument in Pictures



# TM Equivalence Is Undecidable

**Theorem:**

$$EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

is undecidable.



**Proof:**



Assume t $TM$ $TM$:



1. Input is $\langle M \rangle$.
2. Construct a Turing machine $M_\emptyset$ w
3. Run $S$ on $\langle \langle M \rangle, \langle M_\emptyset \rangle \rangle$.
4. If $S$ accepts, accept; if it rejects, reject.

But we know that $E_{TM}$ is undecidable. So $EQ_{TM}$ is undecidable.

# Valid and Invalid Computations

Recall how we captured a Turing machine configuration as a string (such as $\mathtt{ba}q_5\mathtt{bb}$).



A valid computation for Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ (on input $w$) is a string of form



where

1. $C_1$ is $M$'s start configuration,
2. $C_k$ is an accepting configuration
3. for each even $i \in \{2, \ldots, k\}$, $C_{i-1}$ $\quad$ $i$
4. for each odd $i \in \{2, \ldots, k\}$, $(C_{i-1})^{\mathcal{R}} \Rightarrow C_i$



A string (over the same alphabet) which is not a valid computation is an invalid computation.

# The Language of Invalid Computations

Rephrasing: A string $w$ is an invalid computation iff one or more of the following properties hold:

0. $w$ is not of the form $C_1\#\cdots\#C_k\#$ with $C_i$ in $\Gamma^* Q \Gamma^*$.
1. $C_1$ $_0\Sigma^*$.
2. $C_k$
3. $C_{i-1} \not\Rightarrow C_i^{\mathcal{R}}$ for some even $i$.
4. $(C_{i-1})^{\mathcal{R}} \not\Rightarrow C_i$ for some odd $i$.



The set of strings satisfying (0)–(2) are regular lang

We claim the set of strings satisfying (3) is context-free (and so is the set satisfying (4)).

# The Language of Invalid Computations



Here is how to build a PDA $P$ for the set of strings for which $C_{i-1} \not\Rightarrow C_i^R$ is false for some even $i$.

1. $P$ s.
2. W bols of th
3. After skipping the next #, $P$ com inst the string found until the following #, if they ar $P$ scans over the remaining input and accepts.





Conclusion: The language of invalid computations is context-free!

# CFG Exhaustiveness is Undecidable

**Theorem:**



$ALL_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$

is undeci

**Proof:** 
construct a CFG $G$ that generates all the invalid computations for $M$.

Clearly $L(G) = \Sigma^*$ iff $L(M) = \emptyset$.



Hence if $ALL_{CFG}$ is decidable, then so is $E_{TM}$.

Since we proved the latter undecidable, $ALL_{CFG}$ must be undecidable as well.

# CFG Equivalence is Undecidable



Our final language undecidability result is an easy reduction from $ALL_{CFG}$.

**Theore**



$E$ $(G')\}$

is undecidable.



**Proof:** Assume we have a decider $E$ fo build a decider for $ALL_{CFG}$. Namely, given input grammar $G$ over alphabet $\Sigma$, construct a CFG $G_{all}$ for $\Sigma^*$. Then run $E$ on $\langle G, G_{all} \rangle$.

# Undecidability in Logic

Around 1930, first-order logic had been found to have a sound and complete axiomatisation (by Kurt Gödel).



What was then considered the foremost outstanding problem of mathem m:

whether t



> *Th*
> *cedure by which one can decide in a finite numb*
> *tions whether a given logical expression is gen or*
> *is satisfiable. The solution [...] is of fundamen*
> *for the theory of all fields, the theores of which are all capable*
> *of logical development from finitely many axioms.*
>
> *David Hilbert and Wilhelm Ackermann, 1928*

# Undecidability in Logic



As Hilbert and Ackermann point out, if the answer to the question "is there a decision procedure for first-order logic" was yes (as was com
hugely im



It would mean that every theory that can be formalised in first-order logic would have an algorithm for deciding the truth of assertions in that theory.



ann

Alan Turing set out to prove that the answer was no.

# Undecidability in Logic



Validity in propositional logic is decidable.

Validity in first-order predicate logic is undecidable, however, it is semi-de



Howeve m.

We cross the boundary to the undecidable only whe fiers appear together with predicates of arity greater th



Monadic logic, that is, the fragment of predicate l t allow predicates of arity 2 and above is decidable.

# Rice's Theorem

We have this rather sweeping result:



**Rice's Theorem:** Every interesting semantic Turing machine property is undecidable!



A propert

$$P(M_1) \text{ and not}$$



for some Turing machines $M_1$ and $M_2$.

It is semantic iff

$$P(M_1) \text{ iff } P(M_2)$$

for all Turing machines $M_1$ and $M_2$ such that $L(M_1) = L(M_2)$.