Computability Summary

Recursive Languages

- The following are all equivalent:
  - A language *B* is recursive iff *B* = L(*M*) for some total TM *M*.
  - A language *B* is (Turing) computable iff some total TM M computes *B*.
  - A language *B* is decidable iff *B* = L(*m*) for some decision method *m*.
  - A language is recursive iff it is computable.
  - A language is recursive iff it is decidable.

Note: 'iff' = 'if and only if'

Recursively Enumerable Languages

- The following are equivalent:
  - A language *B* is RE iff *B*=L(*M*) for some TM *M.*
  - A language *B* is recognizable iff *B*=L(*m*) for some recognition method *m.*
  - A language is RE iff it is recognizable.

RE Languages

- L_u = {(**p**,**in**) | **p** is a recognition method and **in** $\in L$(**p**)}
  - We have shown that L_u is not decidable
  - We can show that it is recognizable, for each (**p**,**in**) $\in L_u$ we can apply the run method:
    run(**p**,**in**)
    which will halt if **in** $\in L$(**p**)
  - This means the property of p-accepts-in is not decidable.
- L_h = {(**p**,**in**) | **p** is a recognition method that halts on **in**}
  - We have shown that L_h is not decidable
  - We can show that it is recognizable, for each (**p**,**in**) $\in L_h$ we can apply the run method:
    run(**p**,**in**)
    which will halt if **in** $\in L$(**p**)
  - This means the property of p-halts-on-in is not decidable.

Theorem 18.6: Rice's Theorem

For all nontrivial properties α, the language
$\{$**`p`** $|$ **`p`** is a recognition method and $L($**`p`**$)$ has property $\alpha\}$
is not recursive.

- To put it another way: *all nontrivial properties of the RE languages are undecidable*
- Some examples of languages covered by the Rice's Theorem…

Rice's Theorem Examples

L_e = $\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $L(\mathbf{p})$ is empty$\}$

L_r = $\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $L(\mathbf{p})$ is regular$\}$

$\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $L(\mathbf{p})$ is context free$\}$

$\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $L(\mathbf{p})$ is recursive$\}$

$\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $|L(\mathbf{p})| = 1\}$

$\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $|L(\mathbf{p})| \geq 100\}$

$\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $hello \in L(\mathbf{p})\}$

$\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $L(\mathbf{p}) = \Sigma^*\}$

What “Nontrivial” Means

- A property is *trivial* if no RE languages have it, or if all RE languages have it
- Rice’s theorem does not apply to trivial properties such as these:

$\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $L(\mathbf{p})$ is RE$\}$

$\{\mathbf{p} \mid \mathbf{p}$ is a recognition method and $L(\mathbf{p}) \supset \Sigma^*\}$

Languages That Are Not RE

- We've seen examples of nonrecursive languages like L_h and L_u
- Although not recursive, they are still RE: they can be defined using recognition methods (but not using decision methods)
- Are there languages that are not even RE?
- Yes, and they are easy to find…

Theorem 18.9

If a language is RE but not recursive, its complement is not RE.

- Proof is by contradiction
- Let *L* be any language that is RE but not recursive
- Assume by way of contradiction that the complement of *L* is also RE
- Then both *L* and its complement have recognition methods; call them **`lrec`** and **`lbar`**
- We can use them to implement a *decision* method for *L*…

Theorem 18.9, Continued

> If a language is RE but not recursive, its complement is not RE.

```
boolean ldec(String s) {
  for (int j = 1; ; j++) {
    if (runLimited(lrec,s,j)) return true;
    if (runLimited(lbar,s,j)) return false;
  }
}
```

- For some `j`, one of the two `runLimited` calls must return true
- So this is a decision method for *L*
- This is a contradiction; *L* is not recursive
- By contradiction, the complement of *L* is not RE

Closure Properties

- So the RE languages are not closed for complement
- But the recursive languages are
- Given a decision method **`ldec`** for *L*, we can construct a decision method for *L*'s complement:

```
boolean lbar(String s) {return !ldec(s);}
```

- That approach does not work for nonrecursive RE languages
- If the recognition method **`lrec(s)`** runs forever, **`!lrec(s)`** will too

Examples

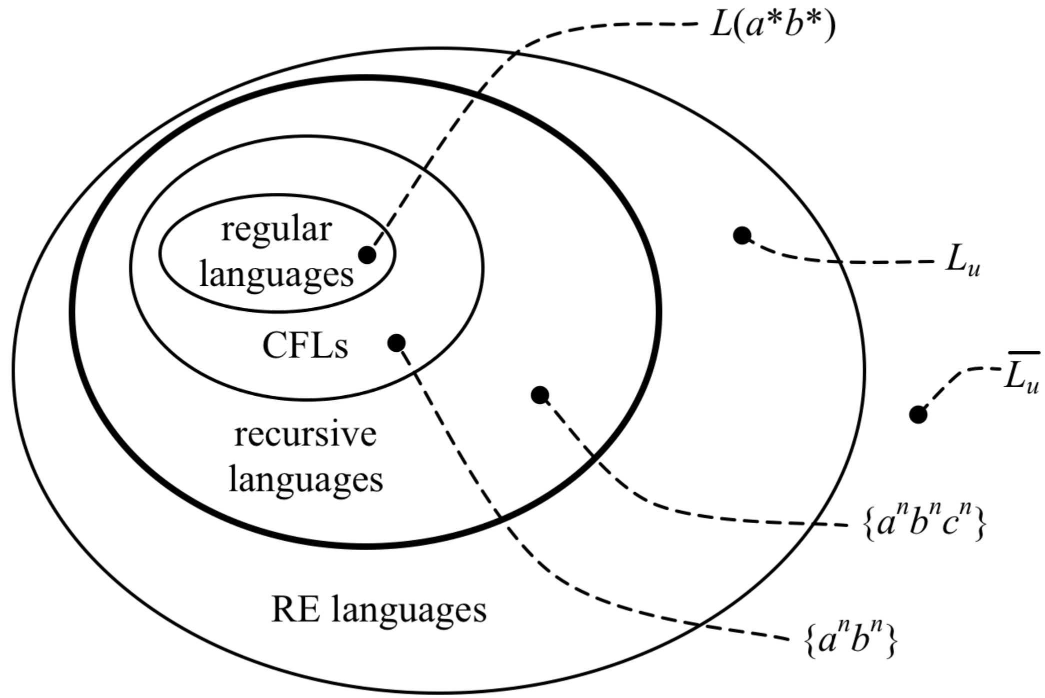
- L_h and L_u are RE but not recursive
- By Theorem 18.9, their complements are not RE:

$$\overline{L_u}$$

$$\overline{L_h}$$

- These languages cannot be defined as $L(M)$ for any TM M, or with any Turing-equivalent formalism

The Big Picture



Recursive

- When a language is *recursive*, there is an effective computational procedure that can definitely categorize all strings
  - Given a positive example it will decide yes
  - Given a negative example it will decide no
- A language that is *recursive*, a property that is *decidable*, a function that is *computable*
- All these terms refer to total-TM-style computations, computations that always halt

RE But Not Recursive

- There is a computational procedure that can effectively categorize positive examples:
  - Given a positive example it will decide yes
  - Given a negative example it may decide no, or may run forever
- A property like this is called *semi-decidable*
- Like the property of $(\mathbf{p},\mathbf{s}) \in L_h$
  - If $\mathbf{p}$ halts on $\mathbf{s}$, a simulation can answer yes
  - If not, neither simulation nor any other approach can always answer with a definite no

Not RE

- There is no computational procedure for categorizing strings that gives a definite yes answer on all positive examples
- Consider $(\mathbf{p},\mathbf{s}) \in \overline{L_h}$
- One kind of positive example would be a recognition method **`p`** that runs forever on **`s`**
- But there is no algorithm to identify such pairs
- Obviously, you can't simulate **`p`** on **`s`**, see if it runs forever, and then say yes