

Thinking Recursively

Part V

Outline for Today

- ***Recursive Backtracking***
 - Finding a needle in a haystack.
- ***On Tenacity***
 - Computational grit!
- ***Optional<T>***
 - Sending data out of functions.
- ***CHeMoWiZrDy***
 - Having some fun with the periodic table.

A Warm-Up Exercise

What's Wrong With This Code?

```
bool containsE(const string& str) {  
    for (char ch: str) {  
        return ch == 'e' || ch == 'E';  
    }  
    return false;  
}
```

Answer at

<https://pollev.com/cs106bwin23>

What's Wrong With This Code?

```
bool containsE(const string& str) {  
    for (char ch: str) {  
        return ch == 'e' || ch == 'E';  
    }  
    return false;  
}
```

It's exceedingly rare to have an unconditional return statement in a for loop. This almost certainly indicates the presence of a bug.

Specifically, this code makes its final decision based on the first character of the string.

Recap from Last Time

A Little Word Puzzle

“What nine-letter word can be reduced to a single-letter word one letter at a time by removing letters, leaving it a legal word at each step?”

One Solution

S	T	A	R	T	L	I	N	G
---	---	---	---	---	---	---	---	---

One Solution

S	T	A	R	T	I	N	G
---	---	---	---	---	---	---	---

One Solution

S	T	A	R	I	N	G
---	---	---	---	---	---	---

One Solution

S	T	R	I	N	G
---	---	---	---	---	---

One Solution

S	T	I	N	G
---	---	---	---	---

One Solution

S	I	N	G
---	---	---	---

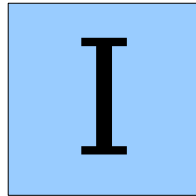
One Solution

S	I	N
---	---	---

One Solution

I	N
---	---

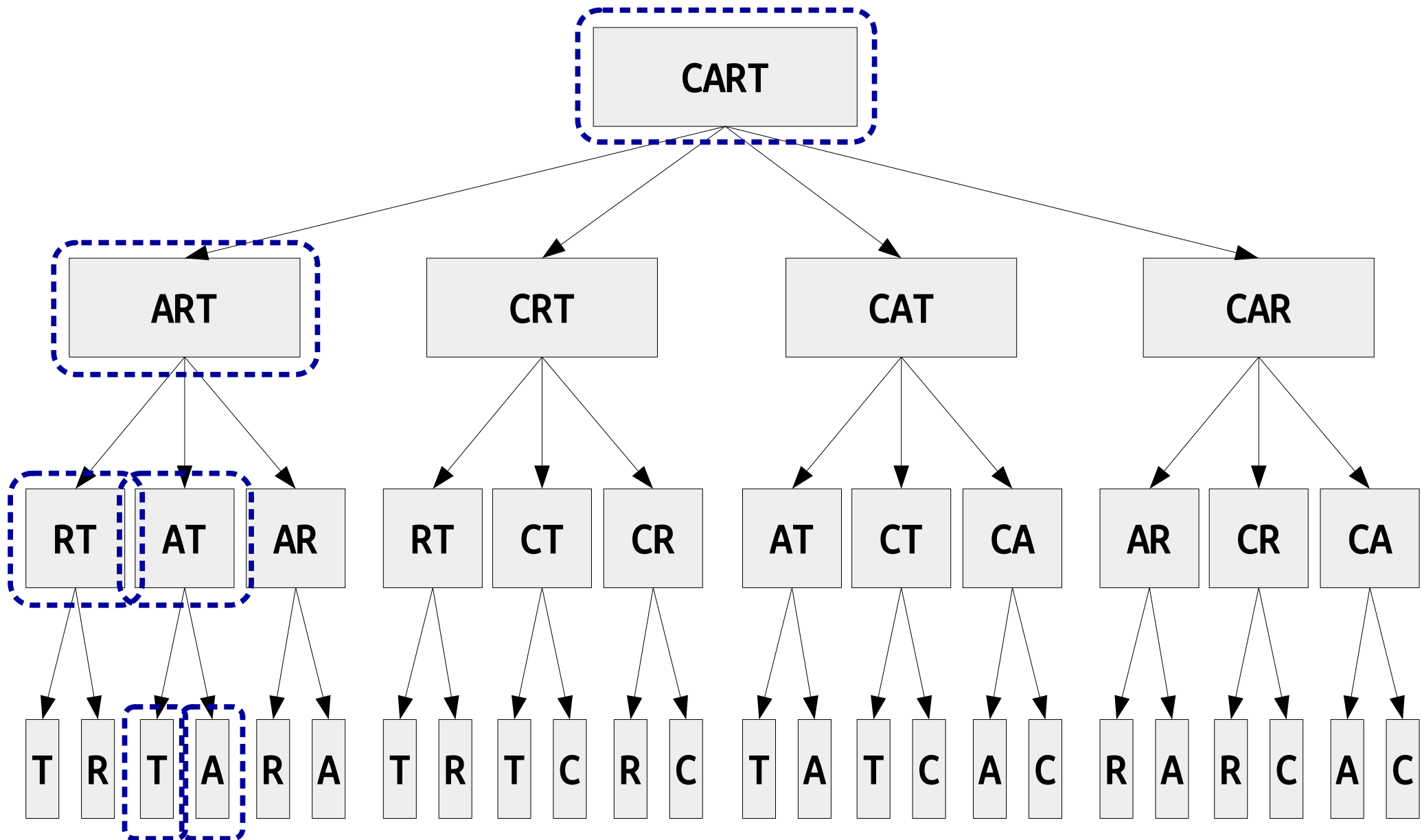
One Solution



New Stuff!

Our Solution, In Action

The Incredible Shrinking Word



```
bool isShrinkableWord(const string& word,
                     const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    for (int i = 0; i < word.length(); i++) {
        string shrunken = word.substr(0, i) + word.substr(i + 1);
        if (isShrinkableWord(shrunken, english)) {
            return true;
        }
    }
    return false;
}
```

```
bool isShrinkableWord(const string& word,
                     const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    for (int i = 0; i < word.length(); i++) {
        string shrunken = word.substr(0, i) + word.substr(i + 1);
        if (isShrinkableWord(shrunken, english)) {
            return true;
        }
    }
    return false;
}
```

```
bool isShrinkableWord(const string& word,
                      const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    for (int i = 0; i < word.length(); i++) {
        string shrunken = word.substr(0, i) + word.substr(i + 1);
        return isShrinkableWord(shrunken, english); // Bad idea!
    }
    return false;
}
```

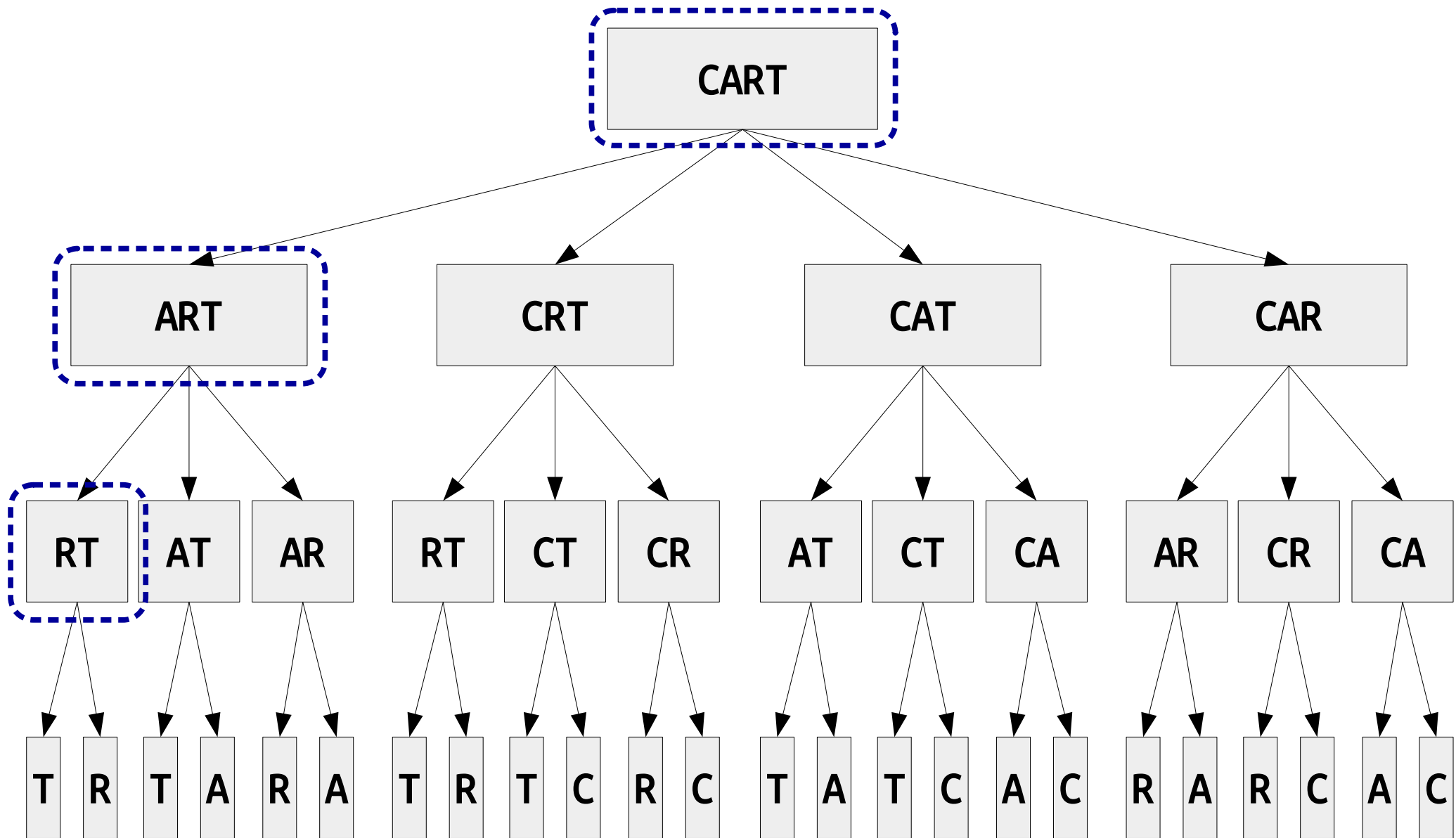
```
bool isShrinkableWord(const string& word,
                     const Lexicon& english) {
    if (!english.contains(word)) {
        return false;
    }
    if (word.length() == 1) {
        return true;
    }

    for (int i = 0; i < word.length(); i++) {
        string shrunken = word.substr(0, i) + word.substr(i + 1);
        return isShrinkableWord(shrunken, english); // Bad idea!
    }
    return false;
}
```

It's exceedingly rare to have an unconditional return statement in a for loop. This almost certainly indicates the presence of a bug.

Specifically, this code makes its final decision based on the first character it tries removing.

Tenacity is a Virtue



When backtracking recursively,
don't give up if your first try fails!

Hold out hope that something else will
work out. It very well might!

Recursive Backtracking

```
if (problem is sufficiently simple) {  
    return whether the problem is solvable  
} else {  
    for (each choice) {  
        try out that choice  
        if (that choice leads to success) {  
            return success;  
        }  
    }  
} return failure;  
}
```

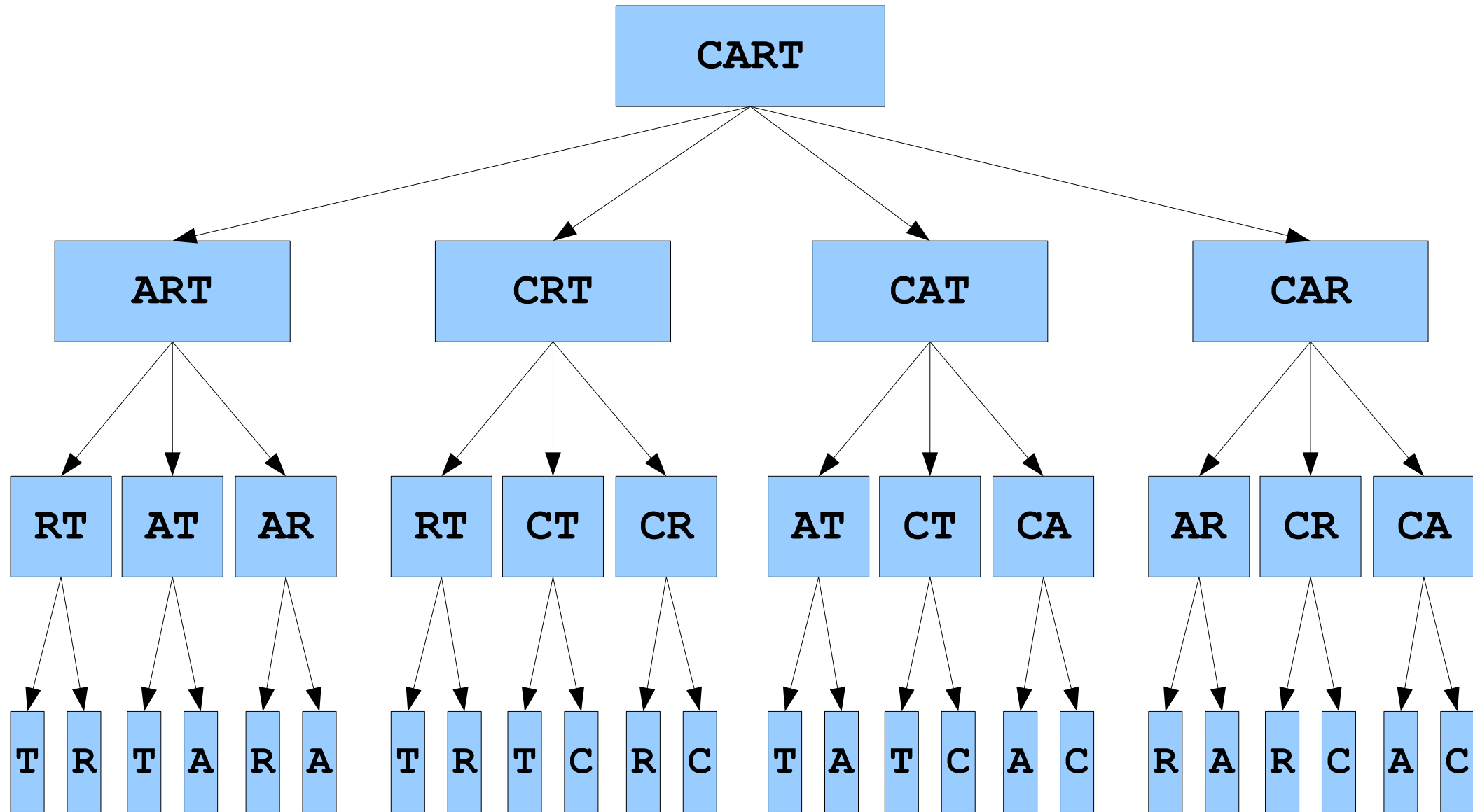
Note that if the recursive call succeeds, then we return success. If it doesn't succeed, that doesn't mean we've failed - it just means we need to try out the next option.

How do we know we're correct?

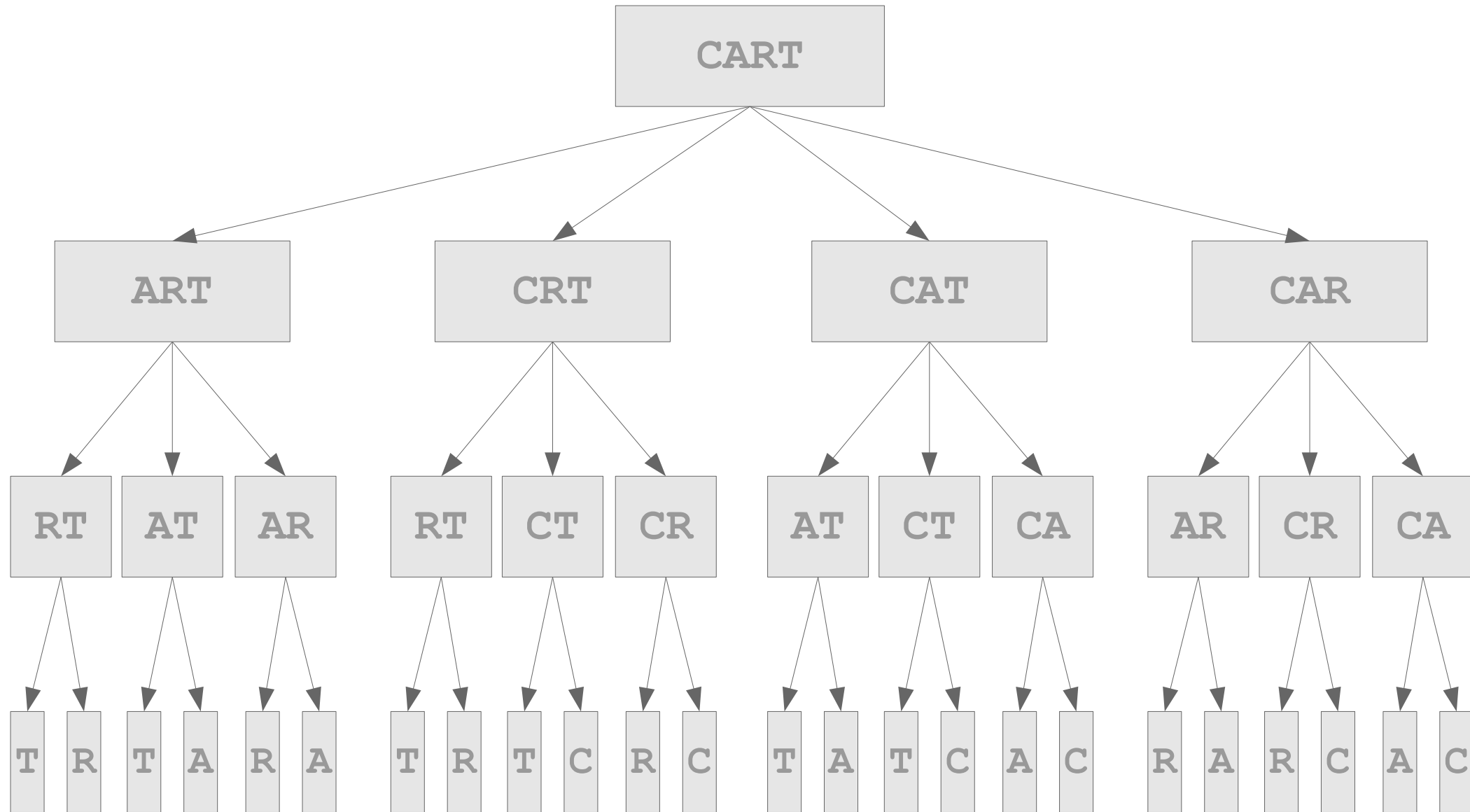
Optional<T>

- The `Optional<T>` type represents either an object of type `T` or is `Nothing` at all.
- It's useful when working with recursive functions that look for something that may or may not exist.
 - If a solution exists, return it as usual.
 - Otherwise, return `Nothing`.
- If the `Optional<T>` is a value of type `T`, you can call the `.value()` function to retrieve the underlying value.

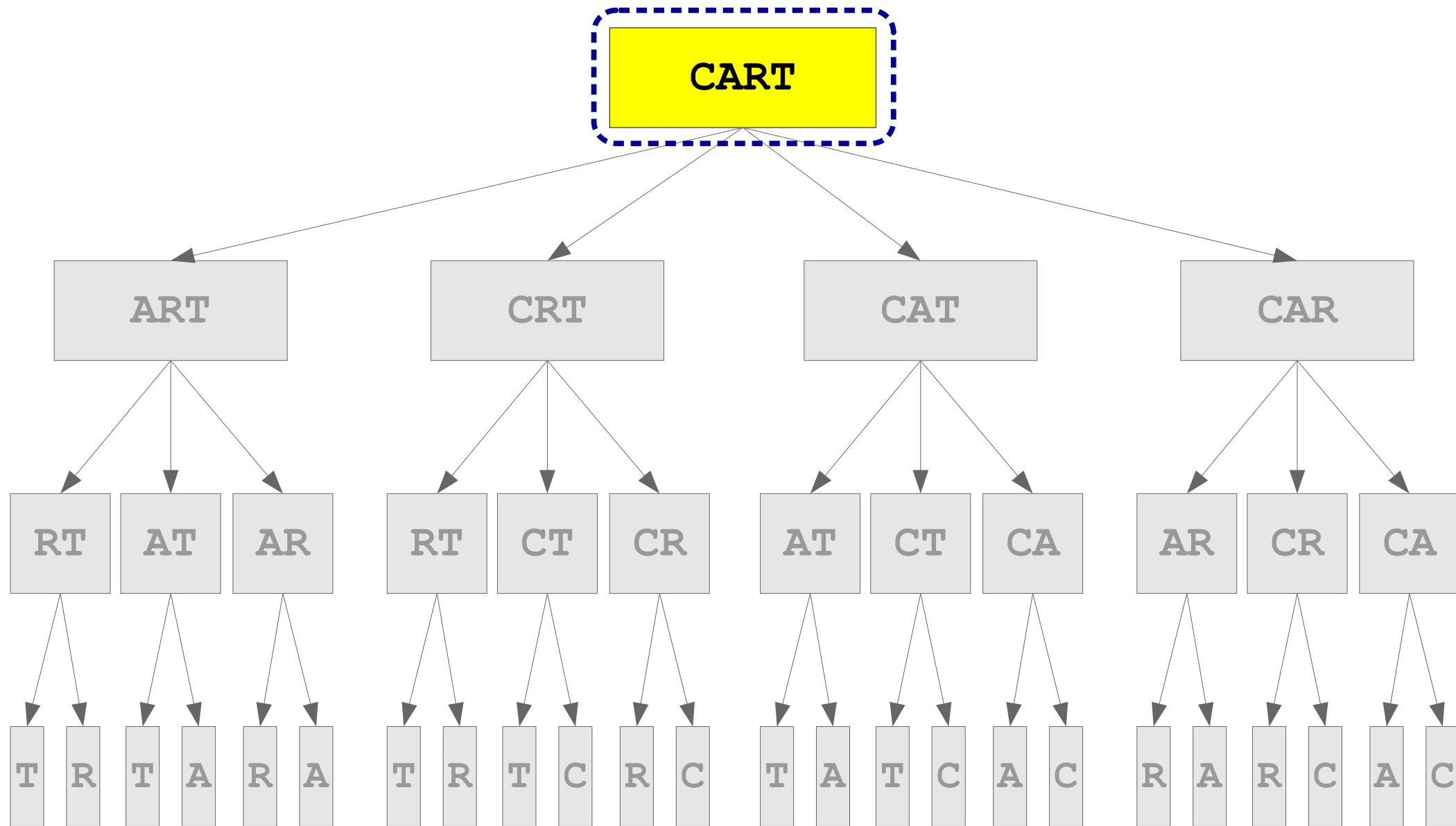
Generating the Answer



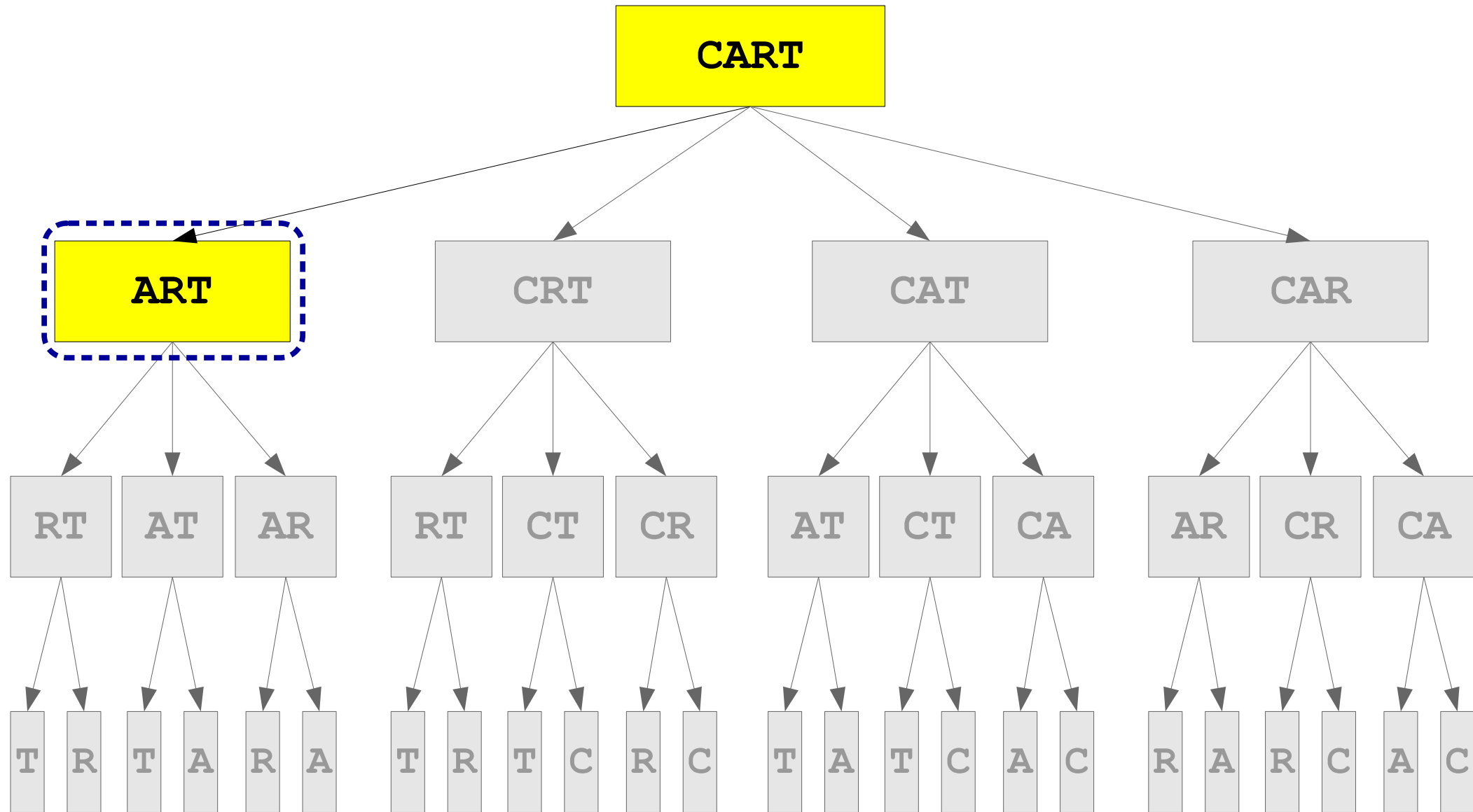
Generating the Answer



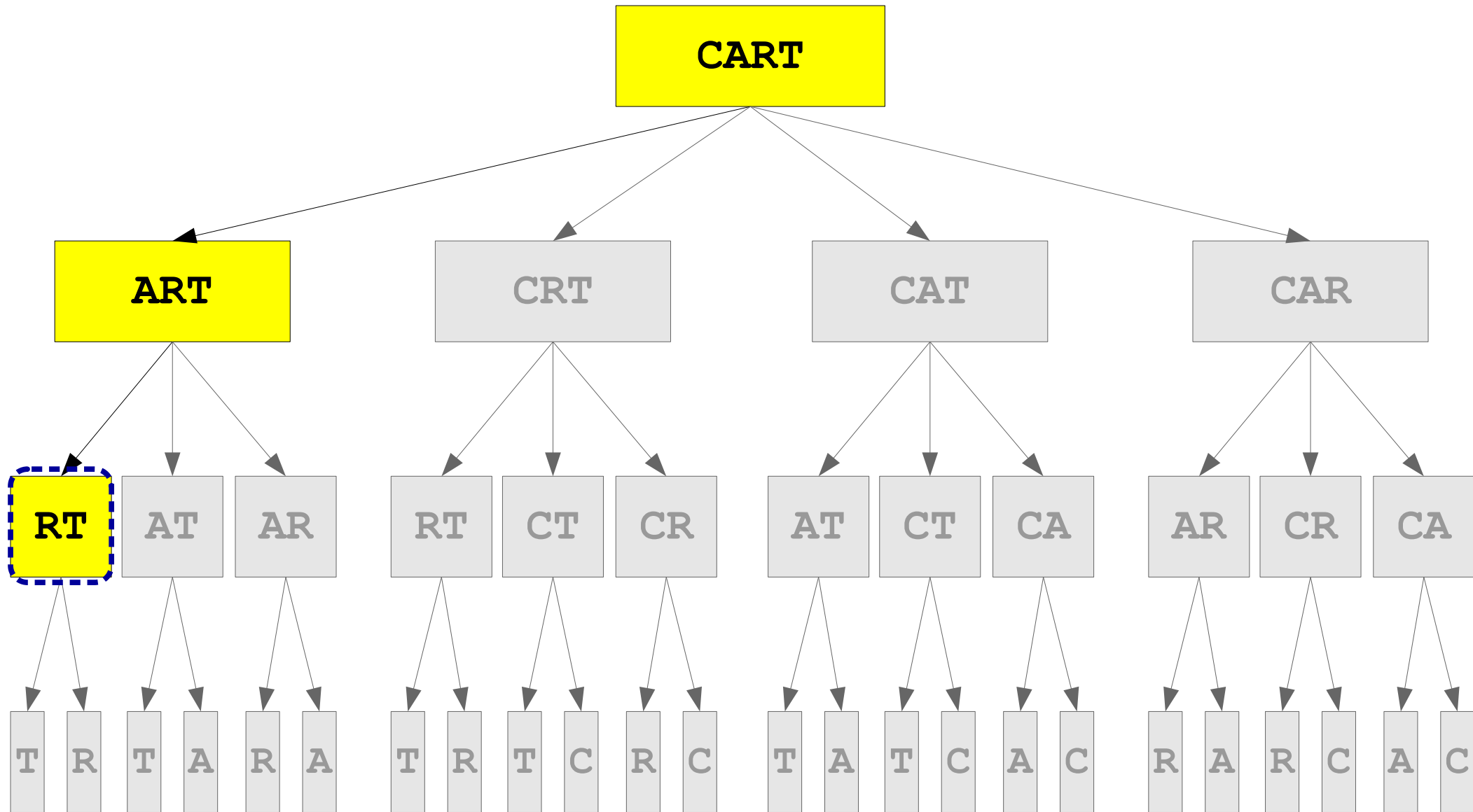
Generating the Answer



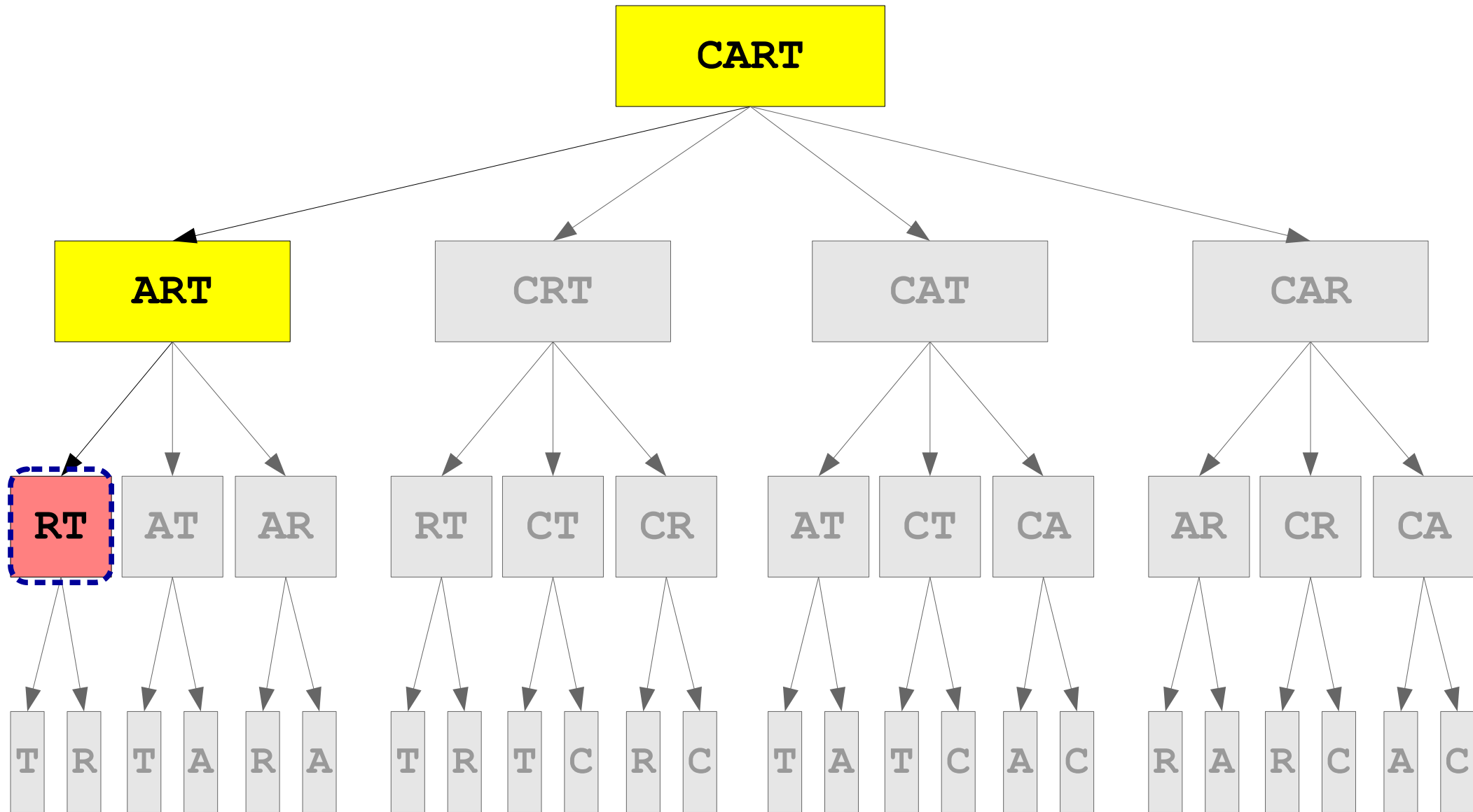
Generating the Answer



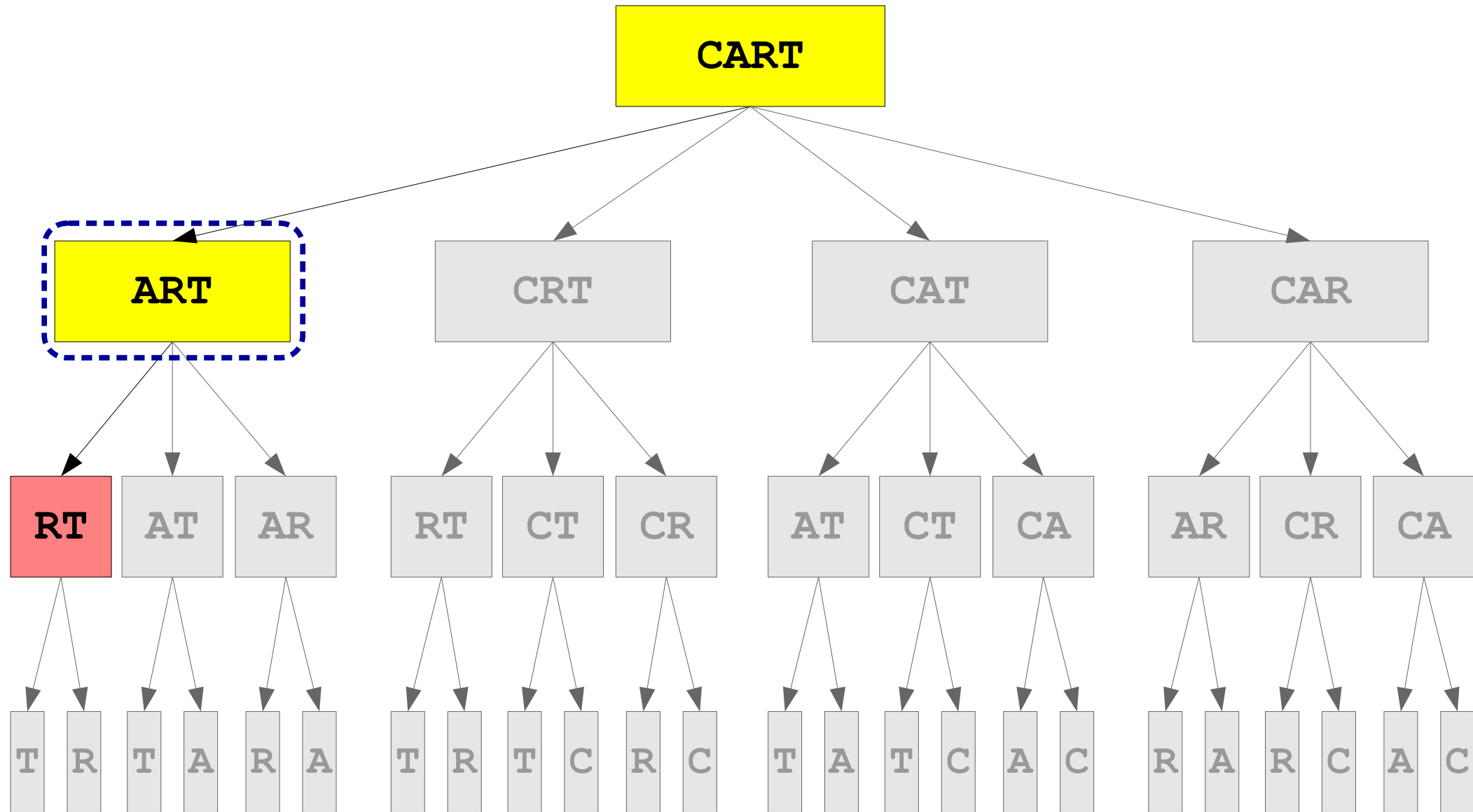
Generating the Answer



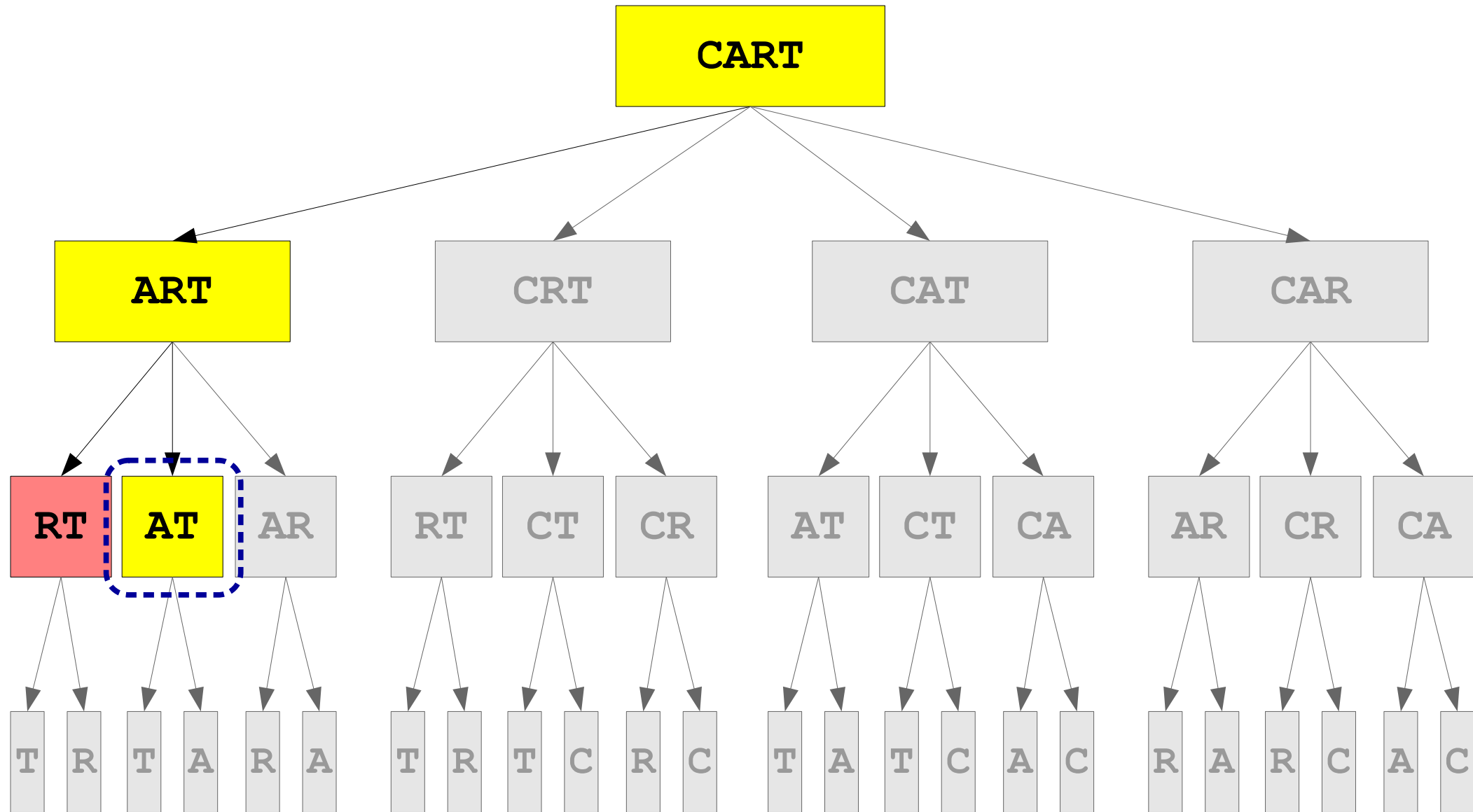
Generating the Answer



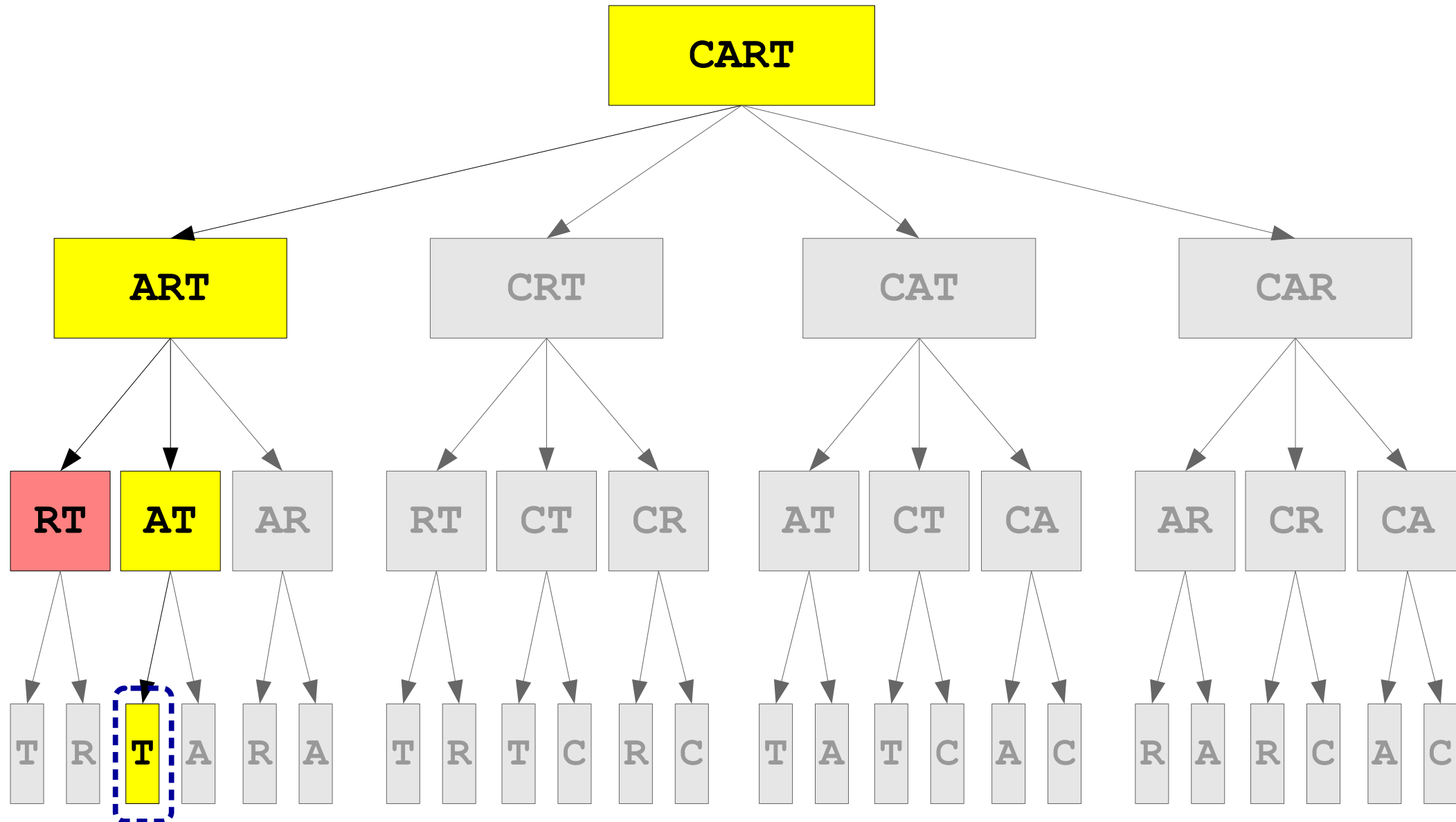
Generating the Answer



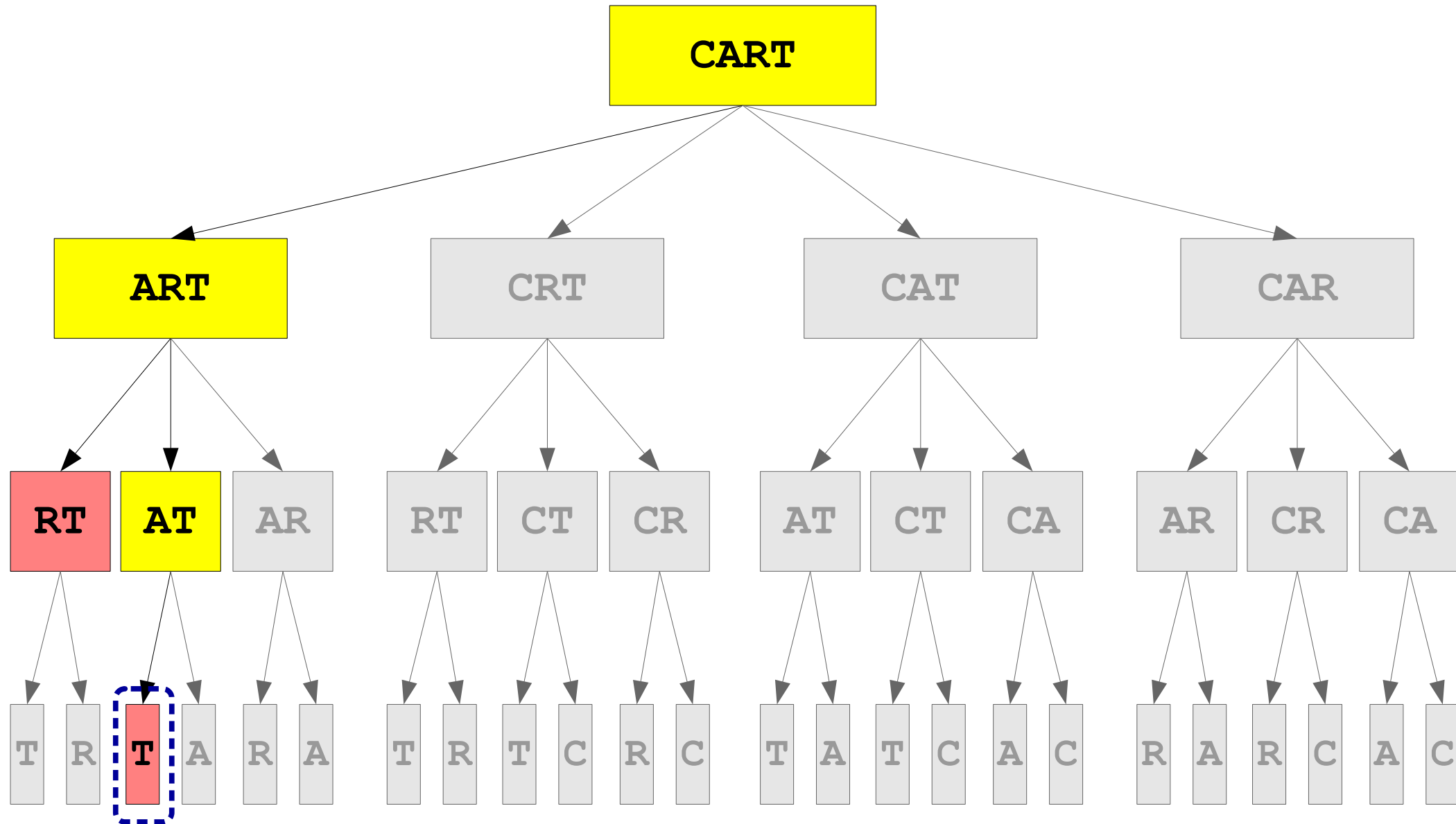
Generating the Answer



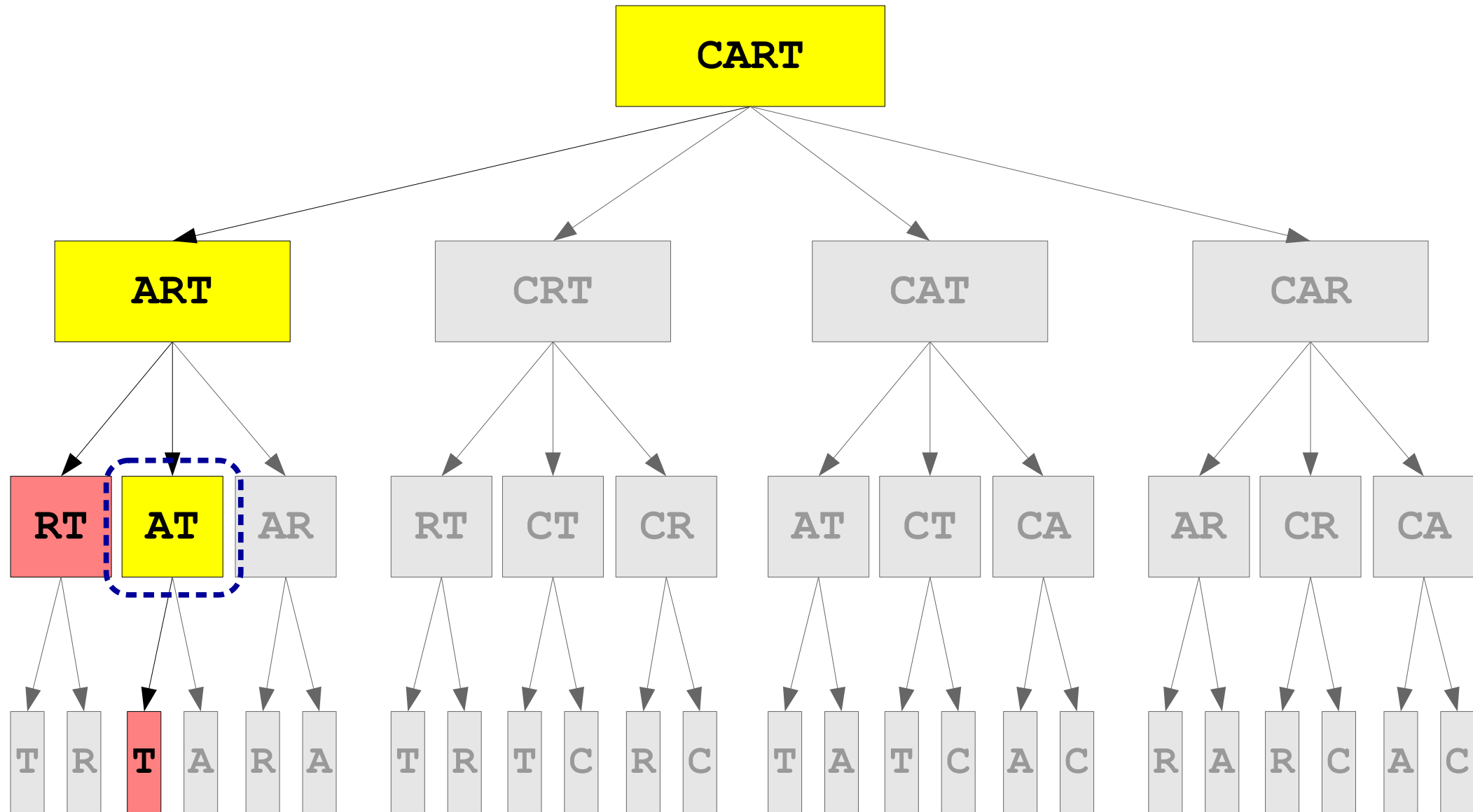
Generating the Answer



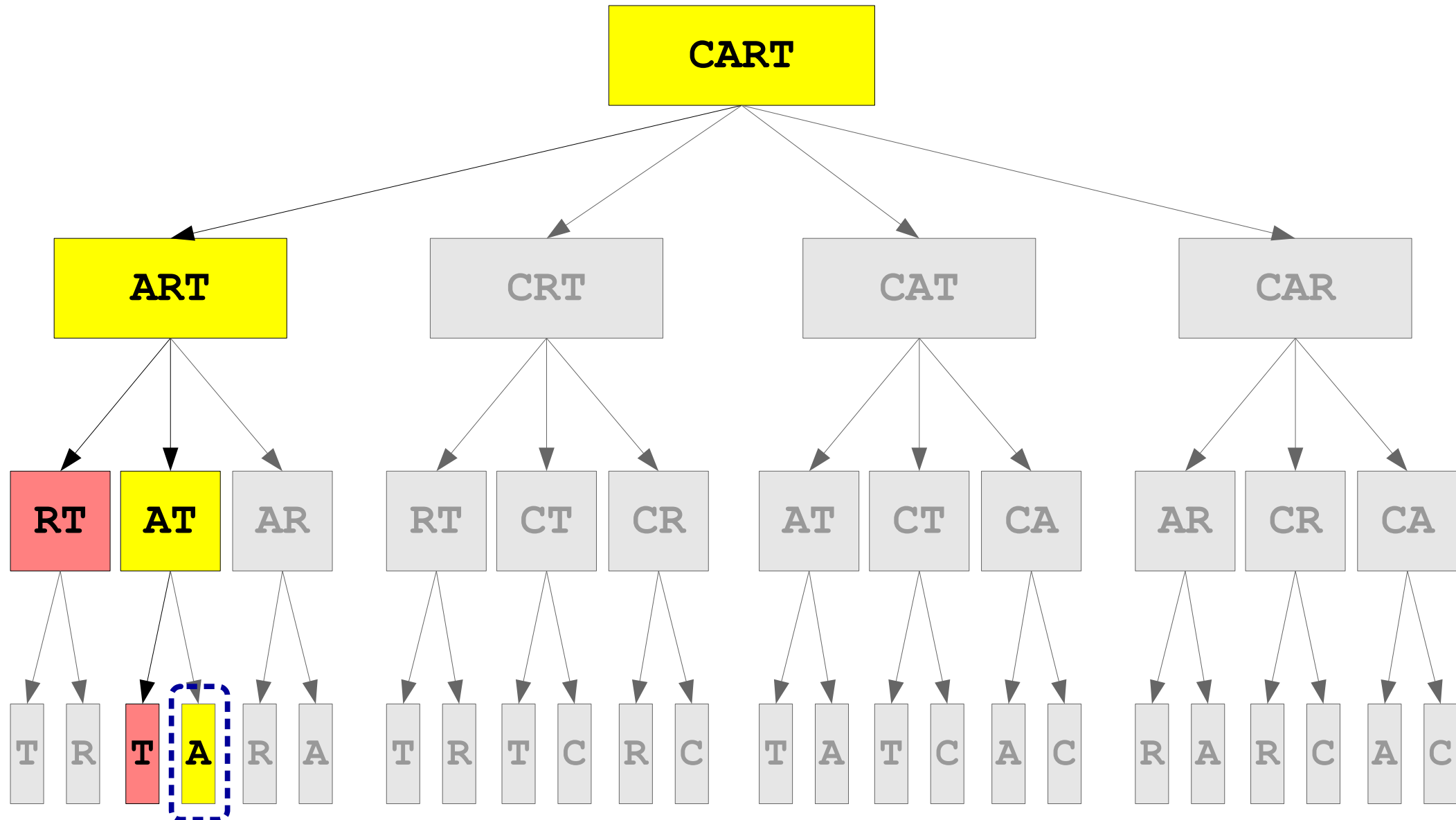
Generating the Answer



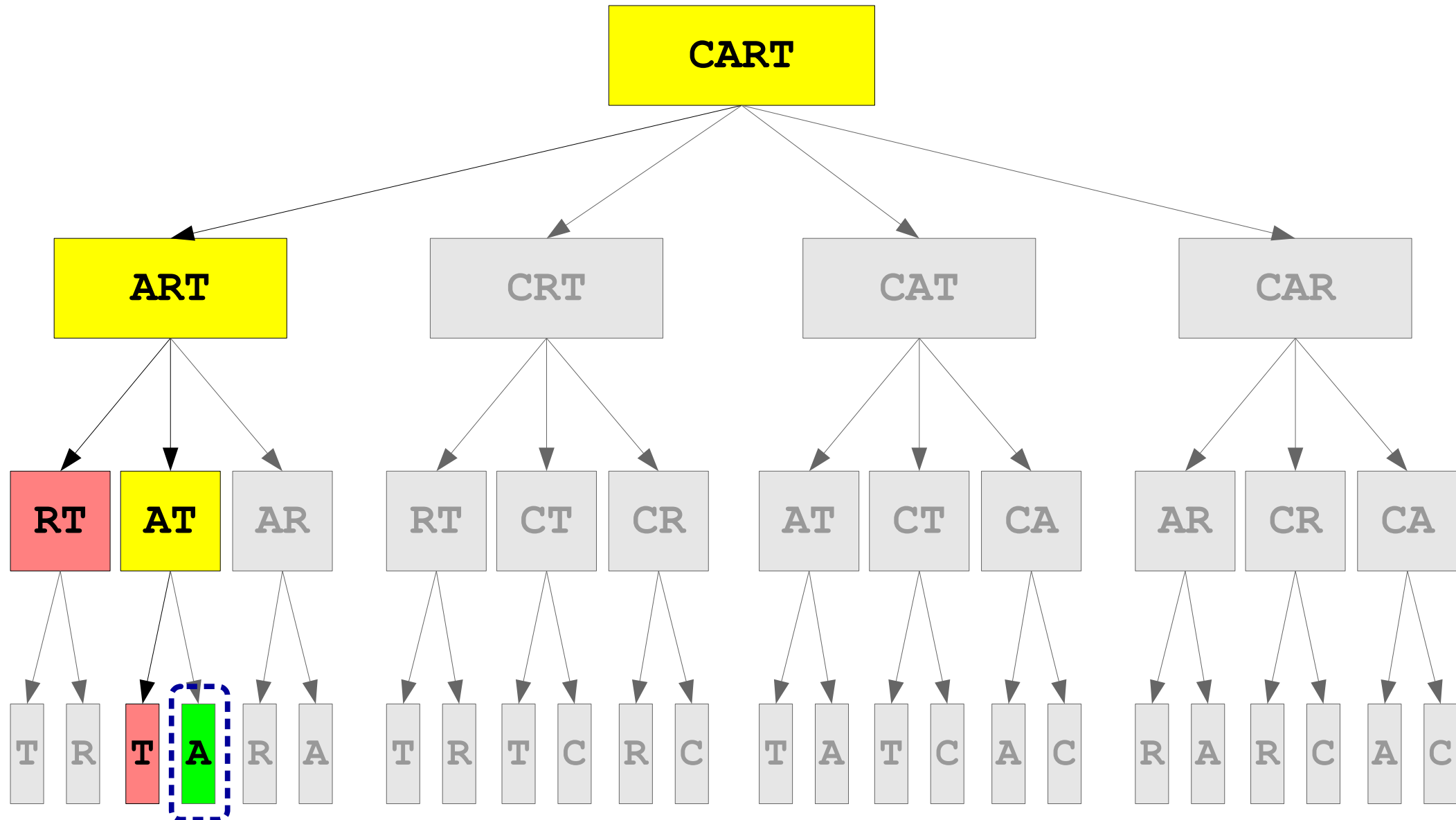
Generating the Answer



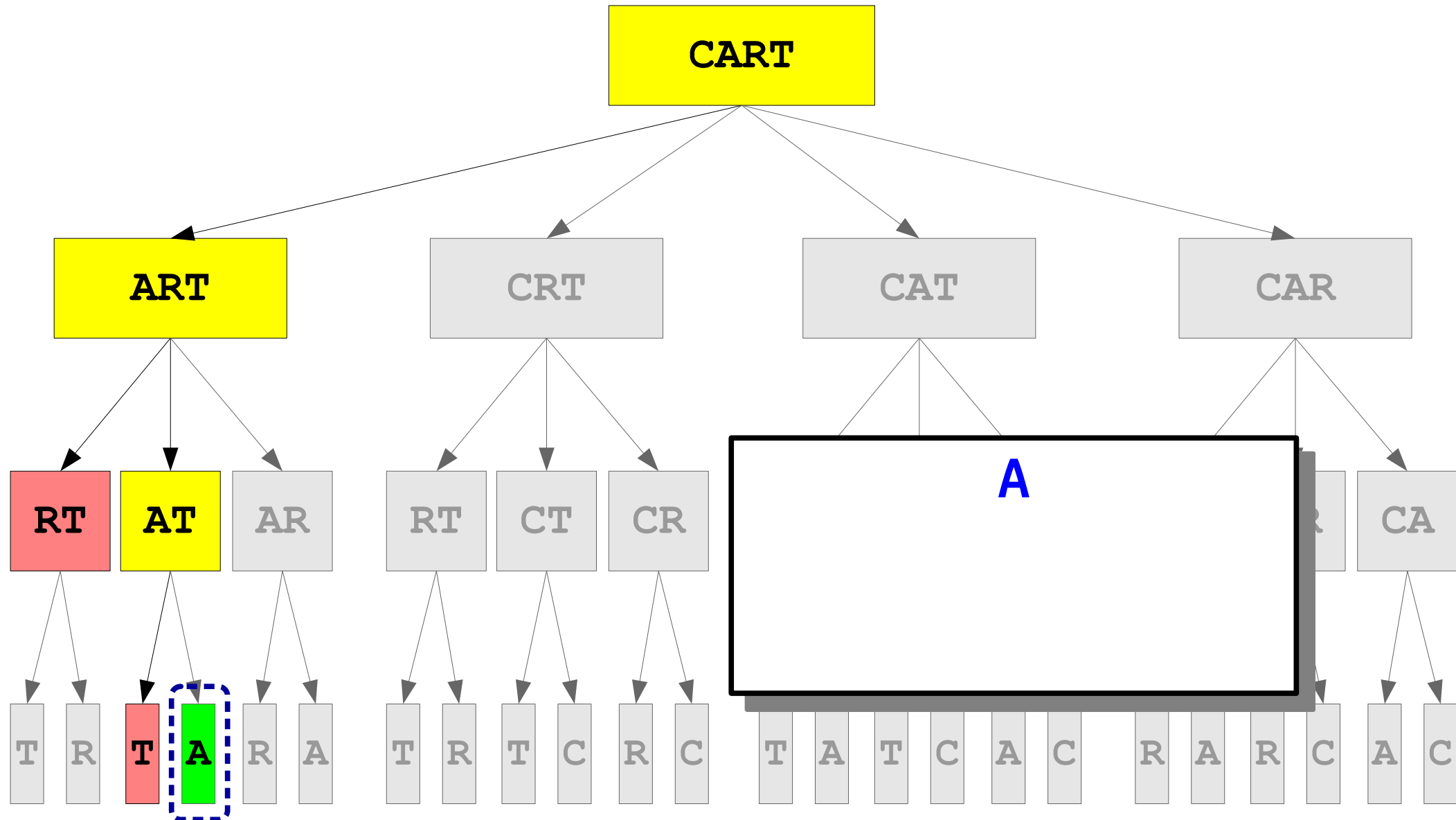
Generating the Answer



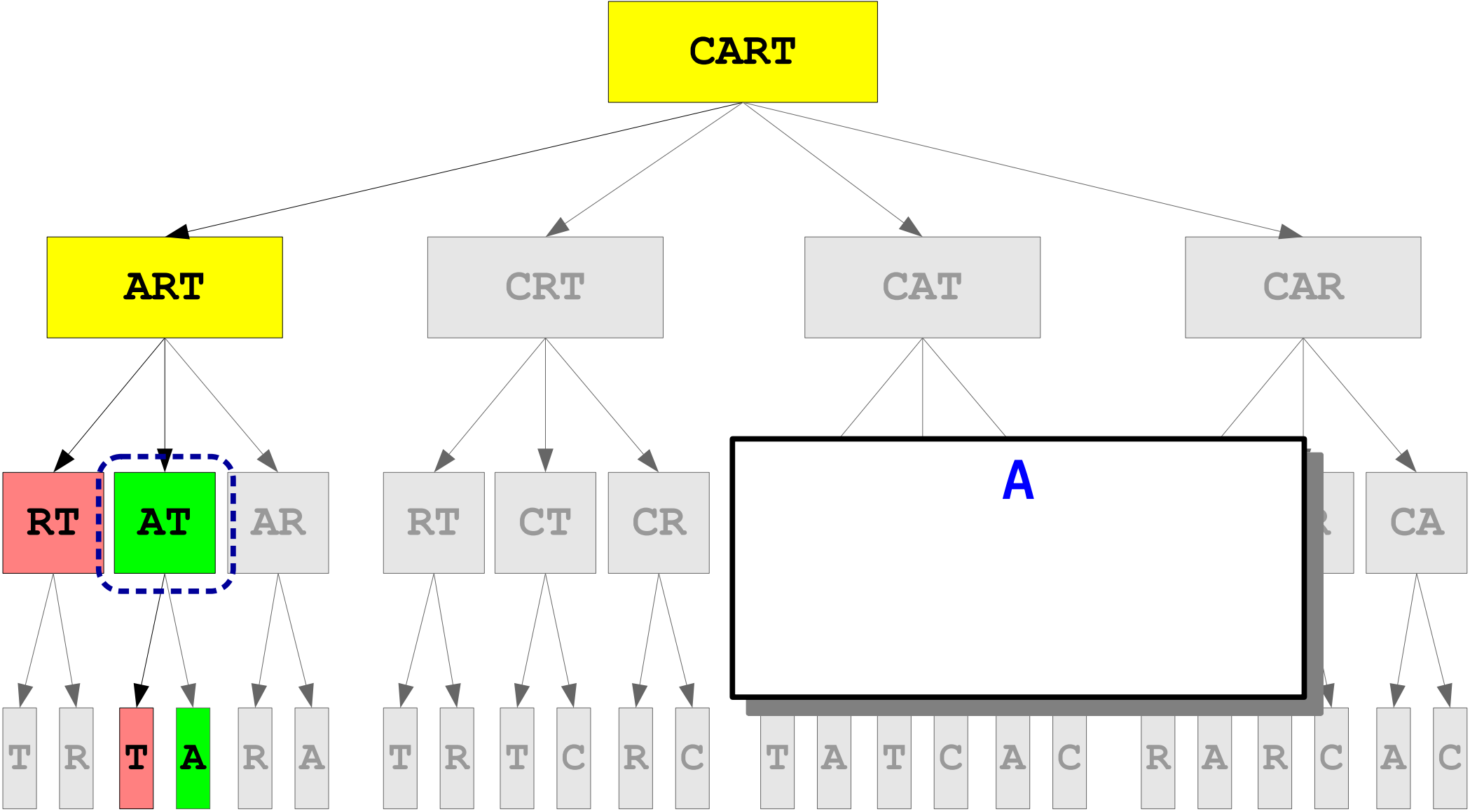
Generating the Answer



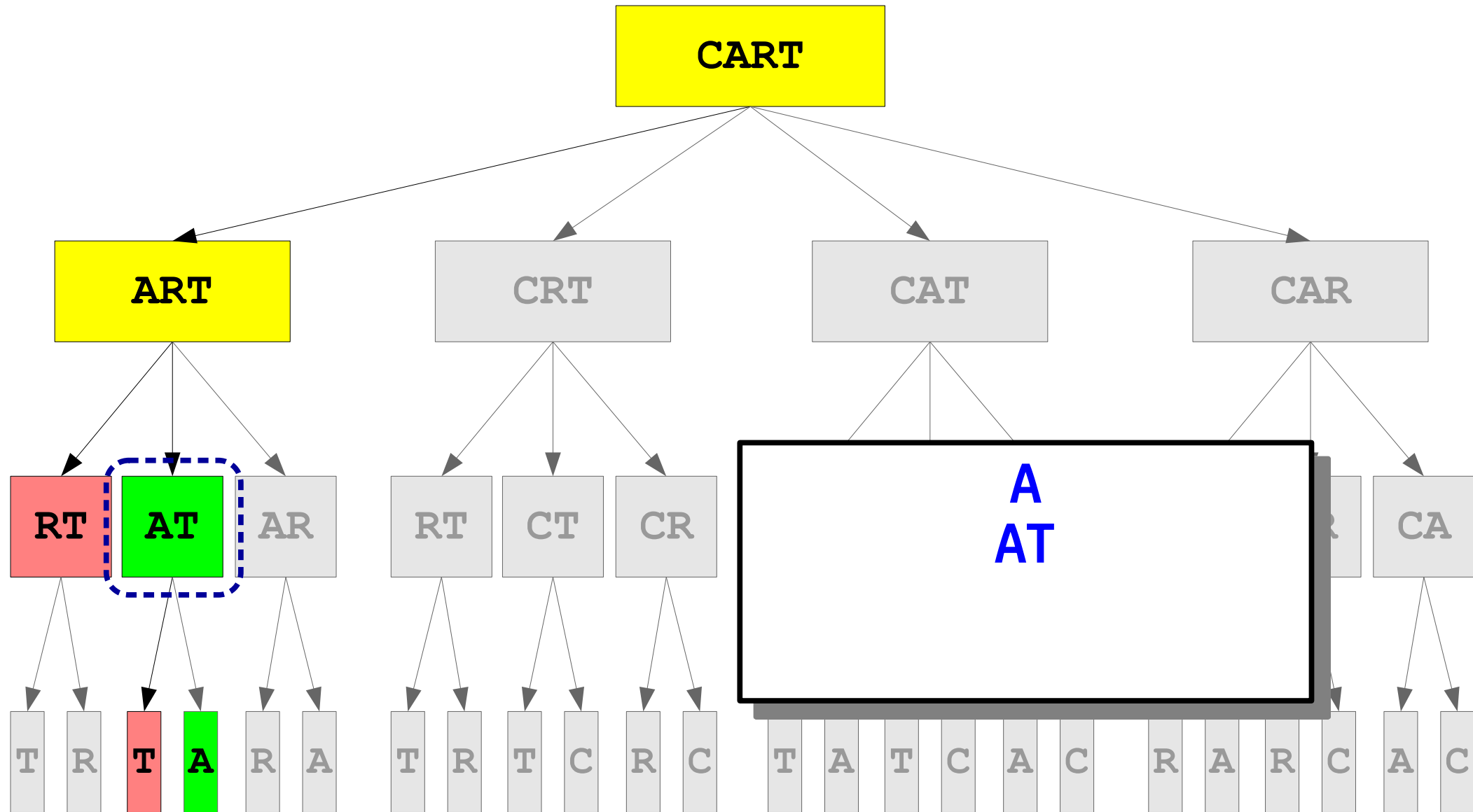
Generating the Answer



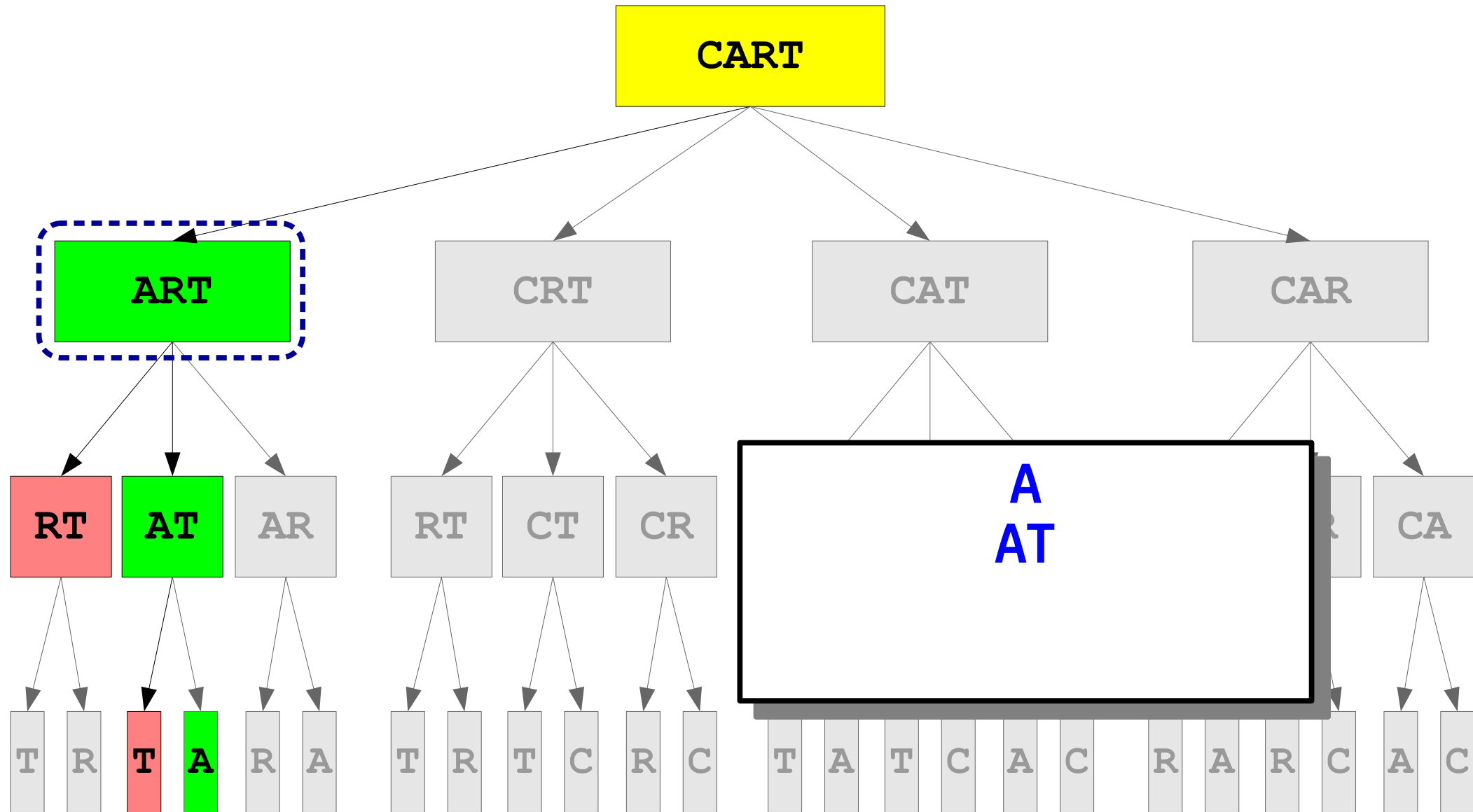
Generating the Answer



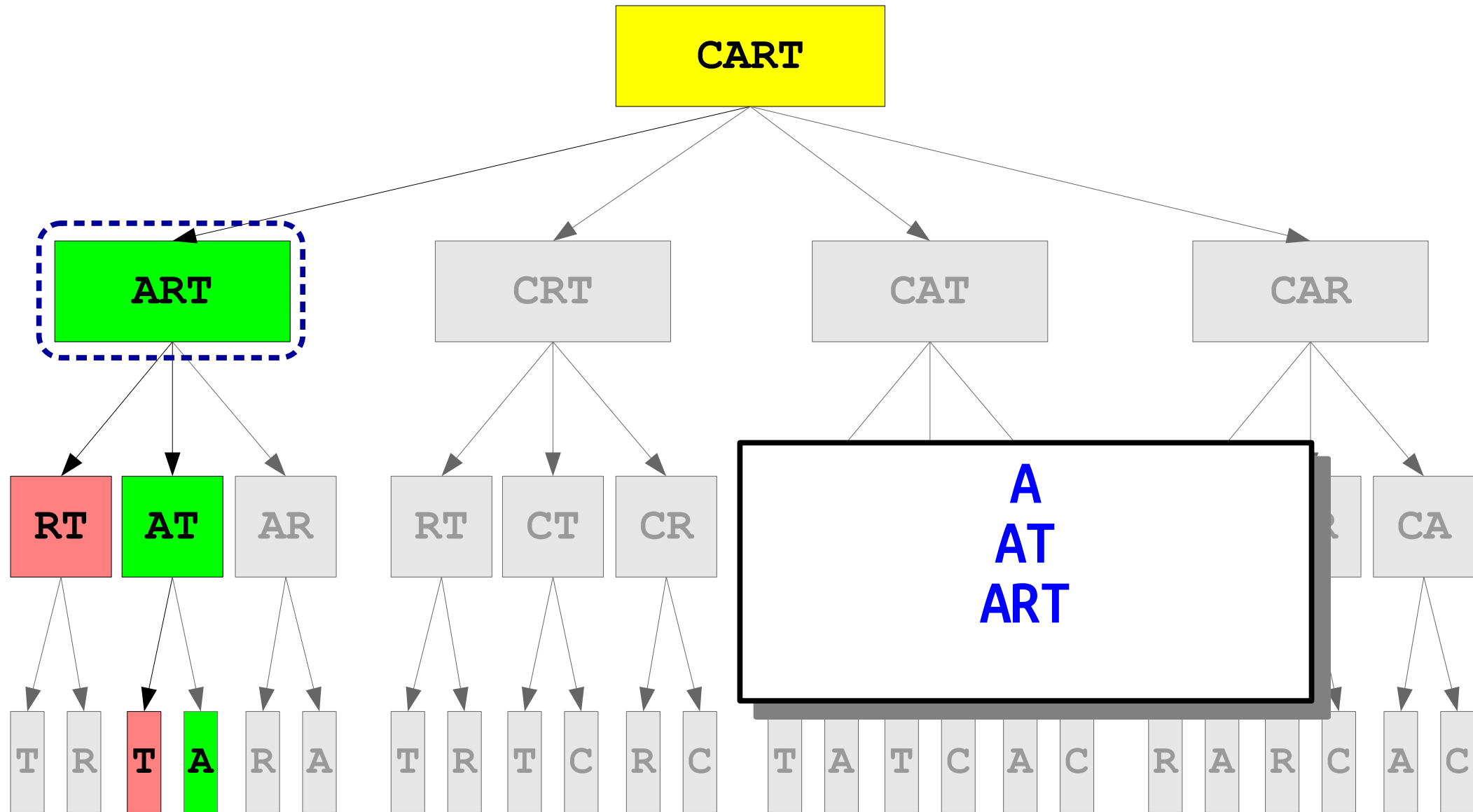
Generating the Answer



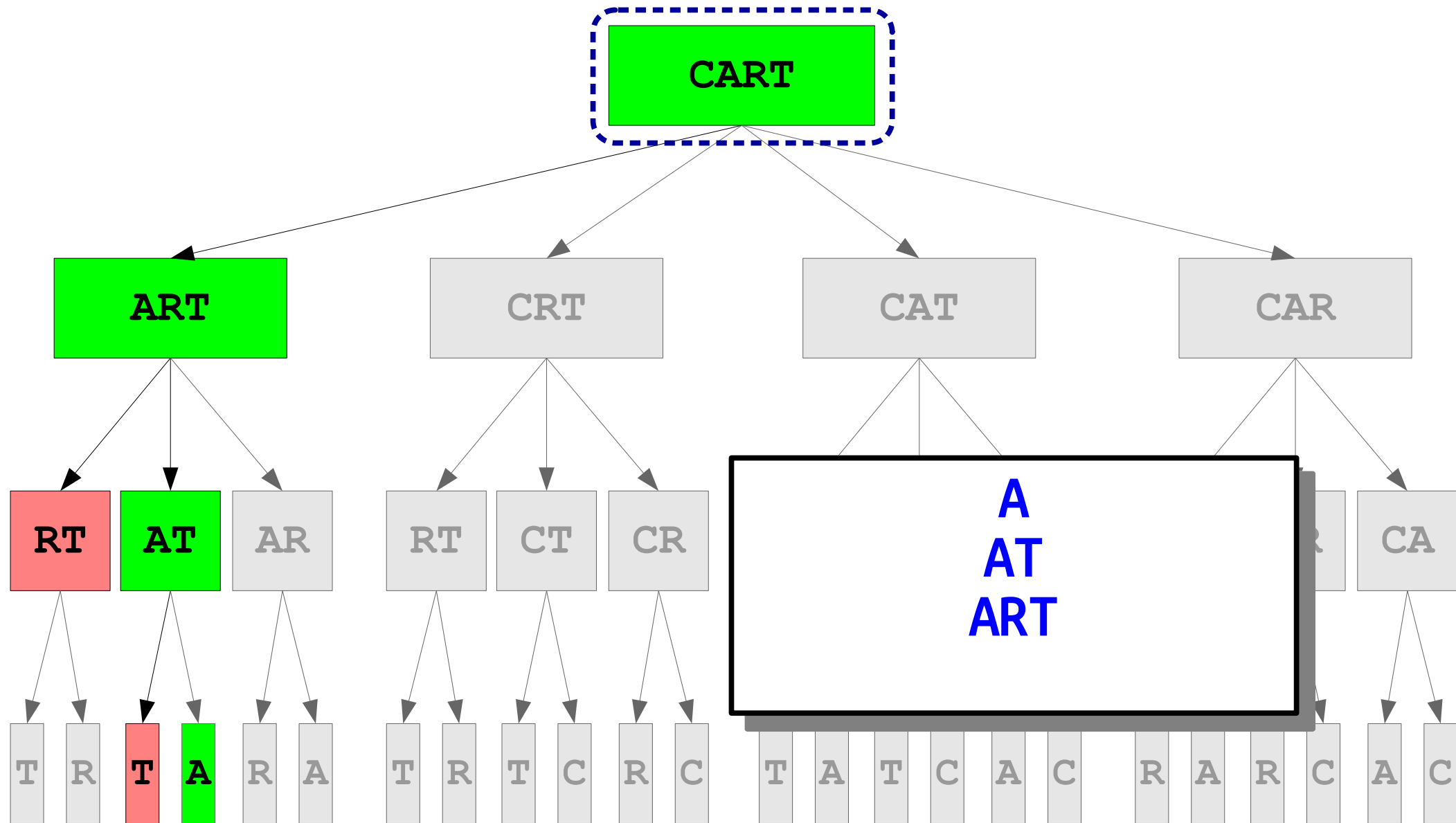
Generating the Answer



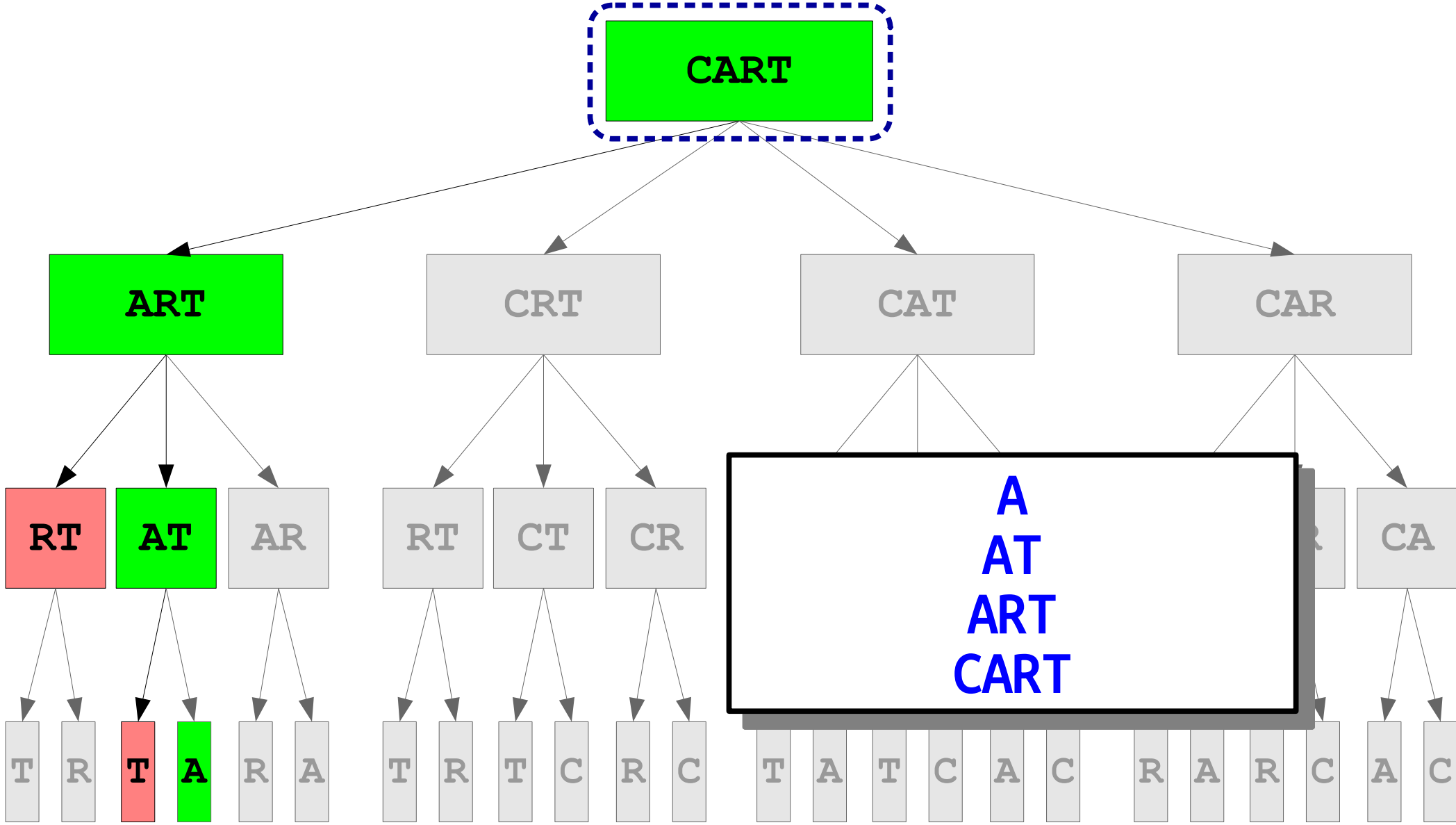
Generating the Answer



Generating the Answer



Generating the Answer



Another Backtracking Example

A Great Tool of Science

1 H Hydrogen																	2 He Helium	
3 Li Lithium	4 Be Beryllium											5 B Boron	6 C Carbon	7 N Nitrogen	8 O Oxygen	9 F Fluorine	10 Ne Neon	
11 Na Sodium	12 Mg Magnesi...											13 Al Aluminium	14 Si Silicon	15 P Phosph...	16 S Sulfur	17 Cl Chlorine	18 Ar Argon	
19 K Potassium	20 Ca Calcium	21 Sc Scandium	22 Ti Titanium	23 V Vanadium	24 Cr Chromium	25 Mn Mangan...	26 Fe Iron	27 Co Cobalt	28 Ni Nickel	29 Cu Copper	30 Zn Zinc	31 Ga Gallium	32 Ge Germani...	33 As Arsenic	34 Se Selenium	35 Br Bromine	36 Kr Krypton	
37 Rb Rubidium	38 Sr Strontium	39 Y Yttrium	40 Zr Zirconium	41 Nb Niobium	42 Mo Molybde...	43 Tc Techneti...	44 Ru Ruthenium	45 Rh Rhodium	46 Pd Palladium	47 Ag Silver	48 Cd Cadmium	49 In Indium	50 Sn Tin	51 Sb Antimony	52 Te Tellurium	53 I Iodine	54 Xe Xenon	
55 Cs Caesium	56 Ba Barium	57 La Lanthan...	72 Hf Hafnium	73 Ta Tantalum	74 W Tungsten	75 Re Rhenium	76 Os Osmium	77 Ir Iridium	78 Pt Platinum	79 Au Gold	80 Hg Mercury	81 Tl Thallium	82 Pb Lead	83 Bi Bismuth	84 Po Polonium	85 At Astatine	86 Rn Radon	
87 Fr Francium	88 Ra Radium	89 Ac Actinium	104 Rf Rutherfo...	105 Db Dubnium	106 Sg Seaborg...	107 Bh Bohrium	108 Hs Hassium	109 Mt Meitneri...	110 Ds Darmsta...	111 Rg Roentge...	112 Cn Coperni...	113 Nh Nihonium	114 Fl Flerovium	115 Mc Moscovi...	116 Lv Livermor...	117 Ts Tenness...	118 Og Oganes...	
			58 Ce Cerium	59 Pr Praseod...	60 Nd Neodym...	61 Pm Prometh...	62 Sm Samarium	63 Eu Europium	64 Gd Gadolini...	65 Tb Terbium	66 Dy Dysprosi...	67 Ho Holmium	68 Er Erbium	69 Tm Thulium	70 Yb Ytterbium	71 Lu Lutetium		
			90 Th Thorium	91 Pa Protacti...	92 U Uranium	93 Np Neptunium	94 Pu Plutonium	95 Am Americium	96 Cm Curium	97 Bk Berkelium	98 Cf Californi...	99 Es Einsteini...	100 Fm Fermium	101 Md Mendele...	102 No Nobelium	103 Lr Lawrenc...		

Oooh! Letters!

1 H Hydrogen																	2 He Helium				
3 Li Lithium	4 Be Beryllium															5 B Boron	6 C Carbon	7 N Nitrogen	8 O Oxygen	9 F Fluorine	10 Ne Neon
11 Na Sodium	12 Mg Magnesi...															13 Al Aluminium	14 Si Silicon	15 P Phosph...	16 S Sulfur	17 Cl Chlorine	18 Ar Argon
19 K Potassium	20 Ca Calcium	21 Sc Scandium	22 Ti Titanium	23 V Vanadium	24 Cr Chromium	25 Mn Mangan...	26 Fe Iron	27 Co Cobalt	28 Ni Nickel	29 Cu Copper	30 Zn Zinc	31 Ga Gallium	32 Ge Germani...	33 As Arsenic	34 Se Selenium	35 Br Bromine	36 Kr Krypton				
37 Rb Rubidium	38 Sr Strontium	39 Y Yttrium	40 Zr Zirconium	41 Nb Niobium	42 Mo Molybde...	43 Tc Techneti...	44 Ru Ruthenium	45 Rh Rhodium	46 Pd Palladium	47 Ag Silver	48 Cd Cadmium	49 In Indium	50 Sn Tin	51 Sb Antimony	52 Te Tellurium	53 I Iodine	54 Xe Xenon				
55 Cs Caesium	56 Ba Barium	57 La Lanthan...	72 Hf Hafnium	73 Ta Tantalum	74 W Tungsten	75 Re Rhenium	76 Os Osmium	77 Ir Iridium	78 Pt Platinum	79 Au Gold	80 Hg Mercury	81 Tl Thallium	82 Pb Lead	83 Bi Bismuth	84 Po Polonium	85 At Astatine	86 Rn Radon				
87 Fr Francium	88 Ra Radium	89 Ac Actinium	104 Rf Rutherfo...	105 Db Dubnium	106 Sg Seaborg...	107 Bh Bohrium	108 Hs Hassium	109 Mt Meitneri...	110 Ds Darmsta...	111 Rg Roentge...	112 Cn Coperni...	113 Nh Nihonium	114 Fl Flerovium	115 Mc Moscovi...	116 Lv Livermor...	117 Ts Tenness...	118 Og Oganes...				
			58 Ce Cerium	59 Pr Praseod...	60 Nd Neodym...	61 Pm Prometh...	62 Sm Samarium	63 Eu Europium	64 Gd Gadolini...	65 Tb Terbium	66 Dy Dysprosi...	67 Ho Holmium	68 Er Erbium	69 Tm Thulium	70 Yb Ytterbium	71 Lu Lutetium					
			90 Th Thorium	91 Pa Protacti...	92 U Uranium	93 Np Neptunium	94 Pu Plutonium	95 Am Americium	96 Cm Curium	97 Bk Berkelium	98 Cf Californi...	99 Es Einsteini...	100 Fm Fermium	101 Md Mendele...	102 No Nobelium	103 Lr Lawrenc...					

Oooh! Letters!

³⁵Br eaking
⁵⁶Ba d

Can We Do Better?

³⁵Br eaking
⁵⁶Ba d

CHeMoWIZrDy

- Some words can be spelled using just element symbols from the periodic table.
- For example:

CaNiNe

FeLiNe

PHYSiCs

UNIVErSITiEs

HAllLuCINoGeNiCs

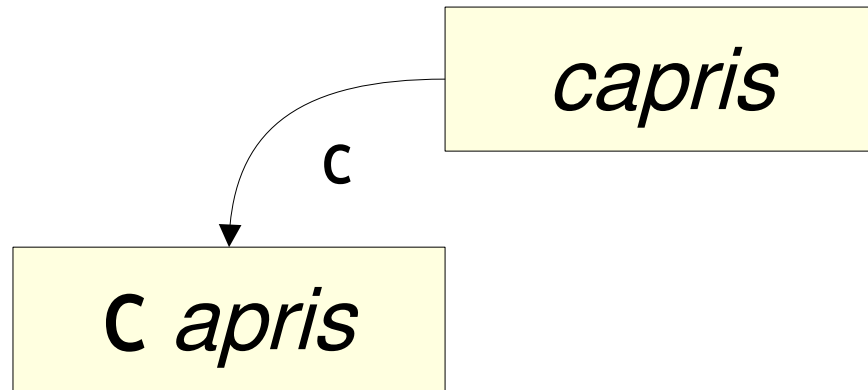
- Given a word, can we spell it using only symbols from the periodic table?
- And, if so, how?

NoTiCe ThAt

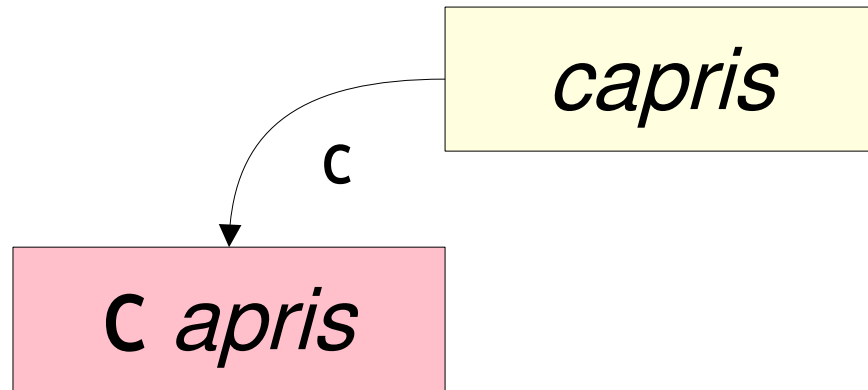
NoTiCe ThAt

capris

NoTiCe ThAt



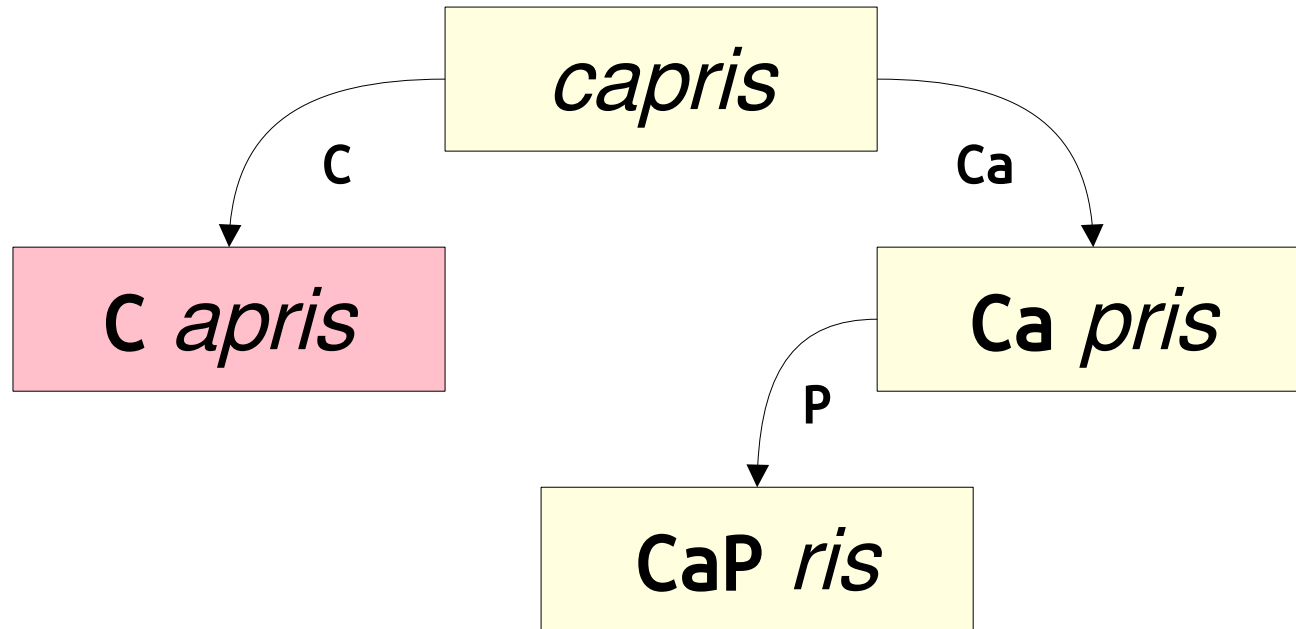
NoTiCe ThAt



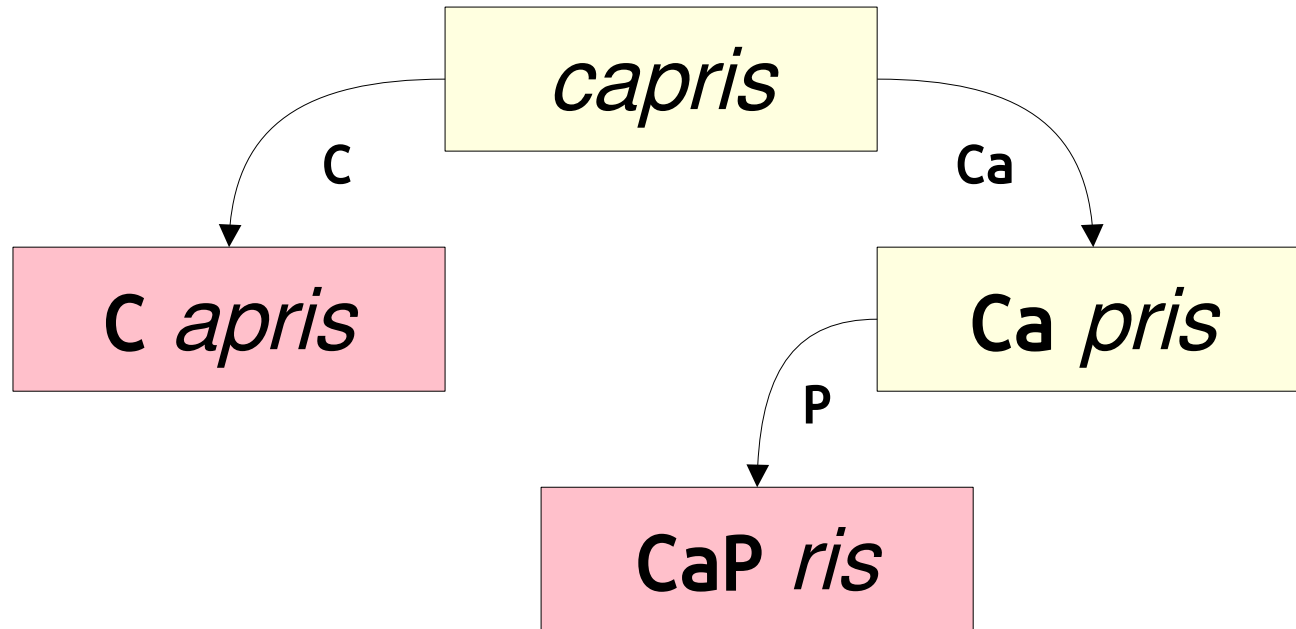
NoTiCe ThAt



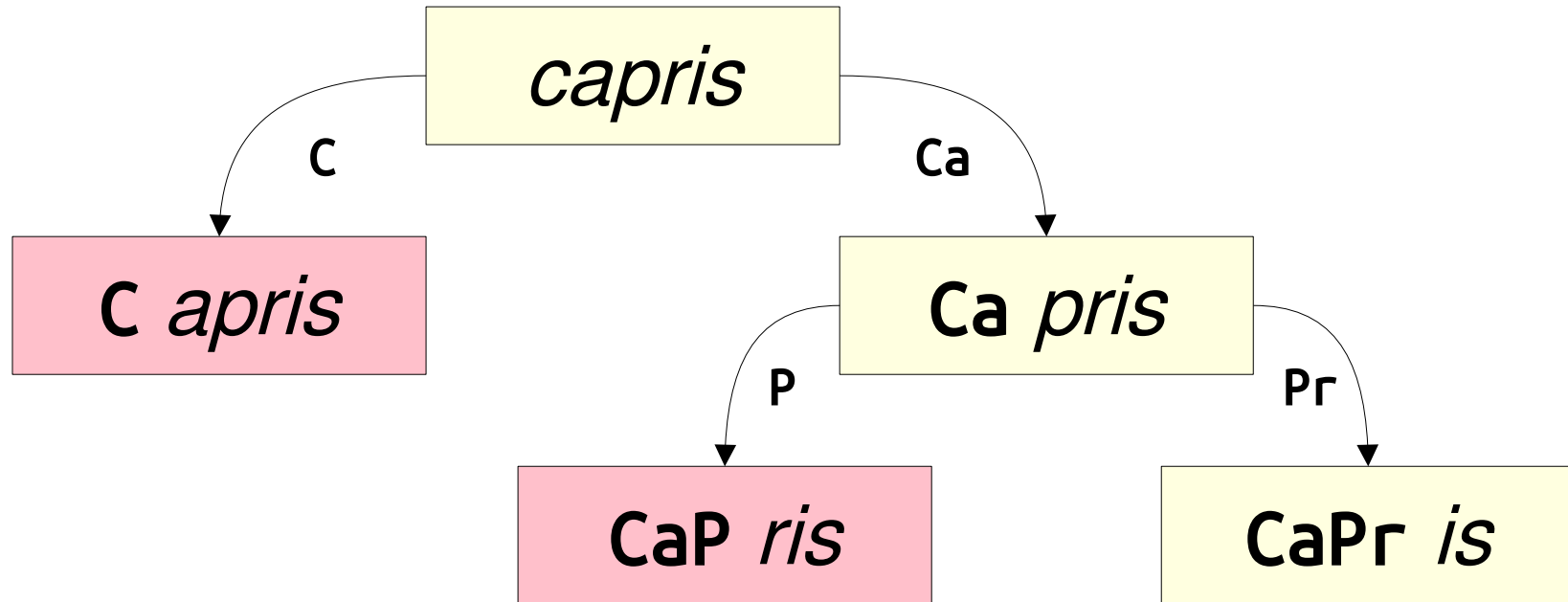
NoTiCe ThAt



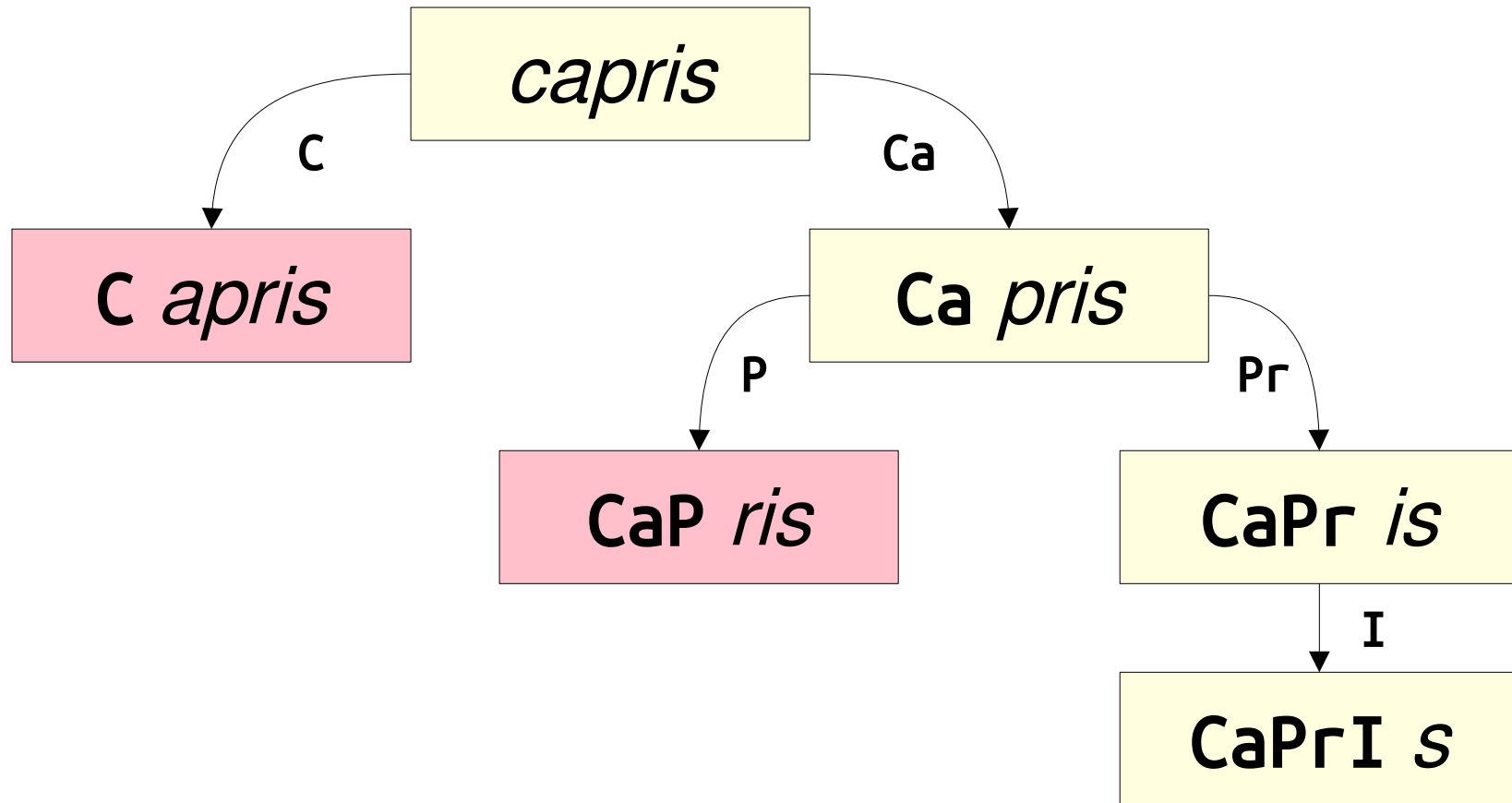
NoTiCe ThAt



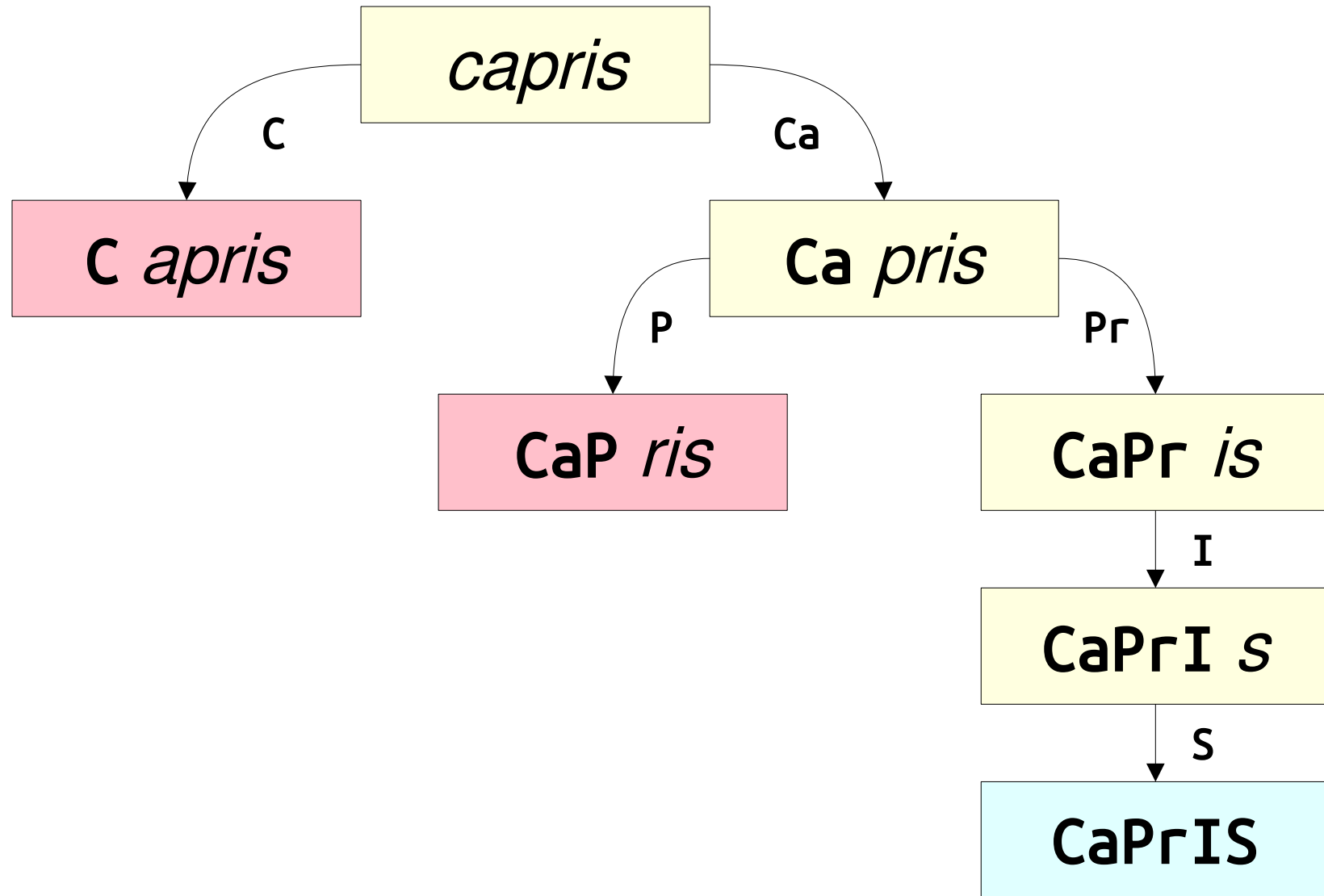
NoTiCe ThAt



NoTiCe ThAt



NoTiCe ThAt



RhHeCuRhSiON

- **BaSe CaSe:**
 - The empty string can be spelled using just element symbols.
- **RhHeCuRhSiV STeP:**
 - For each element symbol:
 - If the string starts with that symbol, check if the rest of the word is spellable.
 - If so, then the original word is spellable too.
 - Otherwise, no option works, so the word isn't spellable.

Closing Thoughts on Recursion

You now know how to use recursion to
***view problems from a different
perspective*** that can lead to ***short and
elegant solutions.***

You've seen how to use recursion to
enumerate all objects of some type,
which you can use to find the
optimal solution to a problem.

You've seen how to use recursive backtracking to ***determine whether something is possible*** and, if so to ***find some way to do it.***

Congratulations on making it this far!

Your Action Items

- ***Finish Chapter 9.***
 - It's all about backtracking, and there are some great examples in there!
- ***Finish Assignment 3.***
 - As always, get in touch with us if we can help out!

Next Time

- ***Algorithmic Analysis***
 - How do we formally analyze the complexity of a piece of code?
- ***Big-O Notation***
 - Quantifying efficiency!