Aabu

Geste de savoir

Arithmétique des résistances

28 novembre 2020

Table des matières

En électronique, on peut associer deux résistances soit en série, soit en parallèle. Des résistances sont associées en série si elles sont traversées par le même courant. Elles sont associées en parallèle si elles ont la même tension à leurs bornes. Dans chaque cas, on peut calculer une résistance équivalente, qui est la valeur de la résistance permettant d'avoir le même comportement électrique que les deux résistances associées.

On peut reproduire ce processus d'association autant de fois qu'on veut et combiner des résistances en série et en parallèle dans tous les sens. Encore une fois, on peut calculer une résistance équivalente, mais les calculs sont pénibles à écrire, même avec une calculatrice évoluée!

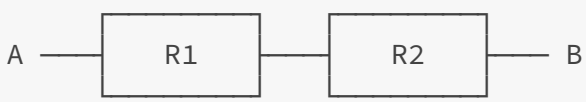
Dans ce billet, je m'amuse à réaliser une calculatrice de résistance équivalente, qui utilise une notation simple pour décrire des associations arbitrairement compliquées de résistances en série et en parallèle.

1. Notation

Habituellement, on décrit les circuits électriques avec des diagrammes. Cependant, j'ai envie de faire une simple interface en ligne de commande pour ma calculatrice, donc j'aimerais une notation sous forme de texte permettant de décrire les associations de résistance en série et en parallèle.

1.1. Association en série

Sur le diagramme ci-dessous, les deux résistances sont en série.

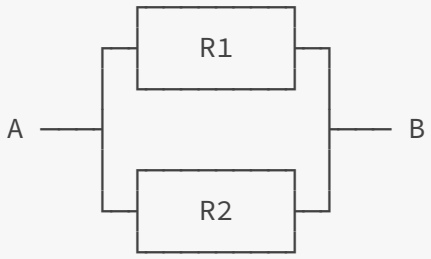


On notera cette association `R1 : R2`.

La valeur de la résistance équivalente est la somme des deux résistances en série: `R1 : R2 = R1 + R2`.

1.2. Association en parallèle

Sur le diagramme ci-dessous, les deux résistance sont en parallèle.



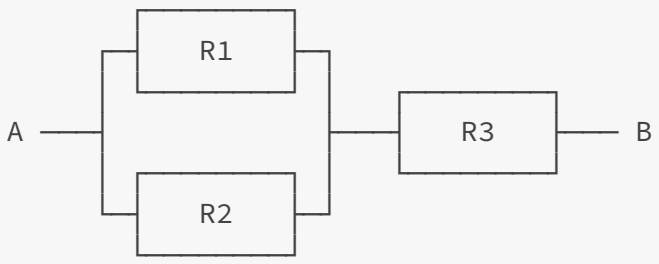
On notera cette association `R1 | R2`.

L'inverse de la résistance équivalente est la somme des inverses des deux résistances en parallèle. En mettant sous une forme présentable, cela revient à: `R1 | R2 = R1 * R2/(R1 + R2)`.

1.3. Multiples associations

On peut combiner les deux opérations précédentes à loisir, à condition de mettre des parenthèses pour indiquer l'ordre des associations.

Par exemple, `(R1 | R2) : R3` correspond au schéma ci-dessous.



On pourrait calculer la valeur de la résistance équivalente en combinant les règles de calcul vue ci-avant, mais c'est déjà trop pour mon degré de paresse.

2. Réalisation de la calculatrice

J'ai fait les choix suivants pour réaliser ma calculatrice:

— la faire en Rust;
— n'utiliser aucune dépendance en dehors de la bibliothèque standard.

Rust est assez expressif et permet d'exprimer certaines idées de manière relativement naturelle. Ça permet de se faire plaisir en ne galérant pas trop sur certains sujets, tout en se concentrant sur ce que je voulais expérimenter, à savoir la compilation *à la main* (d'où le choix de n'utiliser aucune dépendance hors bibliothèque standard).

Le résultat s'appelle `resarith` et est disponible sur GitHub .

2.1. Exemple d'utilisation

Il s'agit d'un programme en ligne de commande. On lui fournit une expression et il renvoie la résistance équivalente.

Par exemple, pour calculer la résistance équivalente à une résistance de 1.5 Ω en série avec une association de deux résistances en parallèle (de respectivement 2 Ω et 3 Ω), on exécute la commande:

```
$ resarith "1.5 : (2 | 3)"
2.7
```

2.2. Architecture

L'architecture du programme est très traditionnelle et s'appuie sur:

— un analyseur lexical,
— un analyseur syntaxique (ou *parser*),
— une fonction d'évaluation.

L'analyseur lexical interprète l'entrée du programme en termes de *tokens*, c'est-à-dire d'éléments abstraits facile à traiter. Il se limite à la reconnaissance de motifs simples tels que des parenthèses ouvrantes ou fermantes, des opérateurs, des espaces ou des nombres. Il n'a aucune connaissance sur la structure que devrait avoir une expression.

Les *tokens* sont transmis à l'analyseur syntaxique qui s'en sert pour construire un arbre décrivant de manière abstraite l'assemblage de résistances. Cette partie du programme est celle qui contient tout ce qu'il faut pour décider du respect des règles. L'analyseur syntaxique sait par

exemple que les parenthèses peuvent être imbriquées ou qu'un opérateur est entouré de deux sous-expressions.

La fonction d'évaluation réduit l'arbre en calculant de proche en proche les résistances équivalentes, puis renvoie le résultat final une fois qu'il n'y a plus aucune réduction possible. C'est la seule partie du programme qui connaît les règles de calcul de résistances. Dans un sens, c'est cette partie qui contient le cœur du programme, tout le reste n'étant que des préparatifs pour cette étape cruciale.

3. Épilogue mathématique

Dans la première section, nous avons définit deux opérations: la mise en série, et la mise en parallèle. Ces opérations ont quelques propriétés mathématiques familières.

3.1. Commutativité

Quand on associe deux résistances en série, l'ordre n'a aucune importance, la résistance équivalente sera la même! Autrement dit `R1 : R2 = R2 : R1`.

Pour la mise en parallèle, on a la même propriété de commutativité: `R1 | R2 = R2 | R1`.

3.2. Associativité

Quand on associe deux résistances en série, et qu'on en associe encore une autre, tout se passe comme si on associait une résistance en série avec un groupe de deux résistances mises en série préalablement. Autrement dit, `(R1 : R2) : R3 = R1 : (R2 : R3)`. Cette propriété justifie que l'on puisse omettre les parenthèses sans ambiguïté dans ce cas-là et écrire `R1 : R2 : R3`.

De même pour les résistances en parallèle, on a `(R1 | R2) | R3 = R1 | (R2 | R3)` et on peut écrire de manière équivalente: `R1 | R2 | R3`.

Cette notation est pratique pour économiser des parenthèses, mais par soucis de simplicité, je ne l'ai pas implémenté pas dans ma calculatrice.

3.3. Existence d'un élément neutre

Quand on associe une résistance en série avec une résistance nulle (c'est-à-dire un court-circuit), c'est comme ne rien faire: on retrouve la résistance d'origine. Autrement dit: `R : 0 = R` et `0 : R = R`. La résistance nulle est un élément neutre pour l'opération de mise en série.

Pour la mise en parallèle, l'élément neutre est la résistance infinie (c'est-à-dire un circuit ouvert). Autrement dit: `R | inf = R` et `inf | R = R`.

Ces propriétés sont celles de *monoïdes commutatifs*. C'est aussi le cas des entiers naturels avec l'addition ou la multiplication, par exemple.

Bref, tout ça est fort amusant, mais le problème inverse est aussi potentiellement amusant (et plus dur): quelle est l'association la plus simple de résistances standardisées pour atteindre une résistance cible avec une précision donnée?