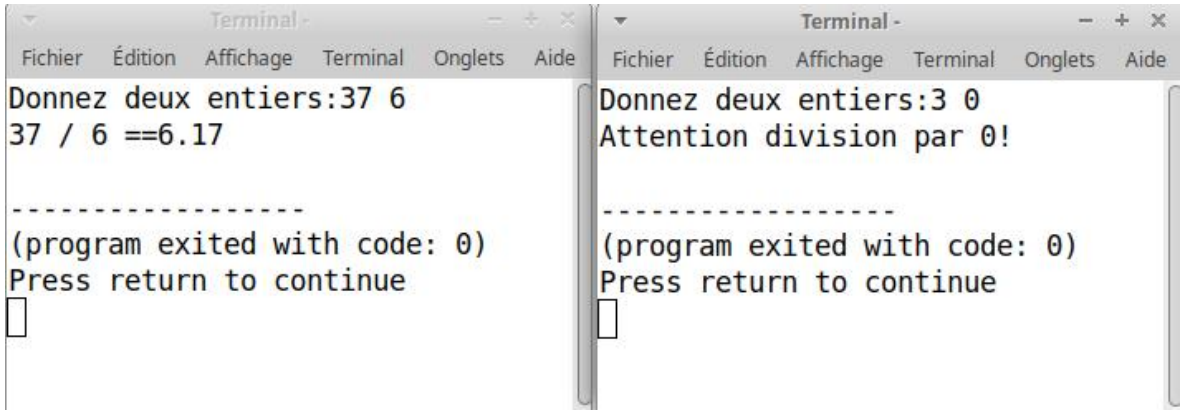
Les instructions conditionnelles en C

Jean-Gabriel Luque

Département d'informatique de l'université de Rouen - Normandie

Qu'est-ce que c'est et à quoi ça sert?

Supposons que nous voulons faire un programme avec le comportement suivant :



$\Rightarrow$ La trace du programme dépend des valeurs fournies

Exemple:

- 37 et 6 : 5 -> 6 -> 7 -> 8 -> 11 -> 16 -> 17.
- 3 et 0 : 5 -> 6 -> 7 -> 8 -> 13 -> 16 -> 17.

IMPOSSIBLE SANS INSTRUCTION DE TEST

Qu'est-ce que c'est et à quoi ça sert?

Programme :

```
 4
 5  int main(void) {
 6      printf("Donnez deux entiers:");
 7      int a, b;
 8      assert(scanf("%d%d", &a, &b) == 2);
 9
10      if ( b != 0) {
11          printf("%d / %d == %0.2lf", a, b, (a+0.0) / b);
12      } else {
13          printf("Attention division par 0!");
14      }
15
16      return EXIT_SUCCESS;
17  }
18
```

⇒ La trace du programme dépend des valeurs fournies
Exemple:

- 37 et 6 : 5 -> 6 -> 7 -> 8 -> 11 -> 16 -> 17.
- 3 et 0 : 5 -> 6 -> 7 -> 8 -> 13 -> 16 -> 17.

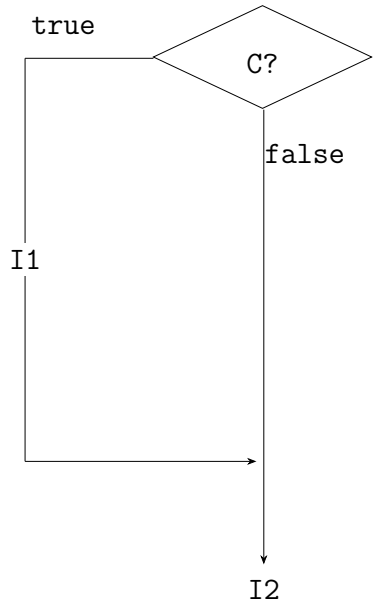
IMPOSSIBLE SANS INSTRUCTION DE TEST

L'instruction `if`

Syntaxe:

```
if (C) {
  I1
}
I2
```

`C` est une expression booléenne, `I1` et `I2` deux blocs d'instructions. Si l'évaluation de `C` donne `true` alors `I1` est executé, sinon rien ne se passe. Puis, dans les deux cas, `I2` est exécuté.



L'instruction `if`

Syntaxe:

```
if (C) {
  I1
}
I2
```

Exemple:

```
if (scanf("%d%d", &a, &b) != 2) {
  printf("Une erreur de saisie nous contraint à ");
  printf("interrompre le programme.\n");
  return EXIT_FAILURE;
}
```

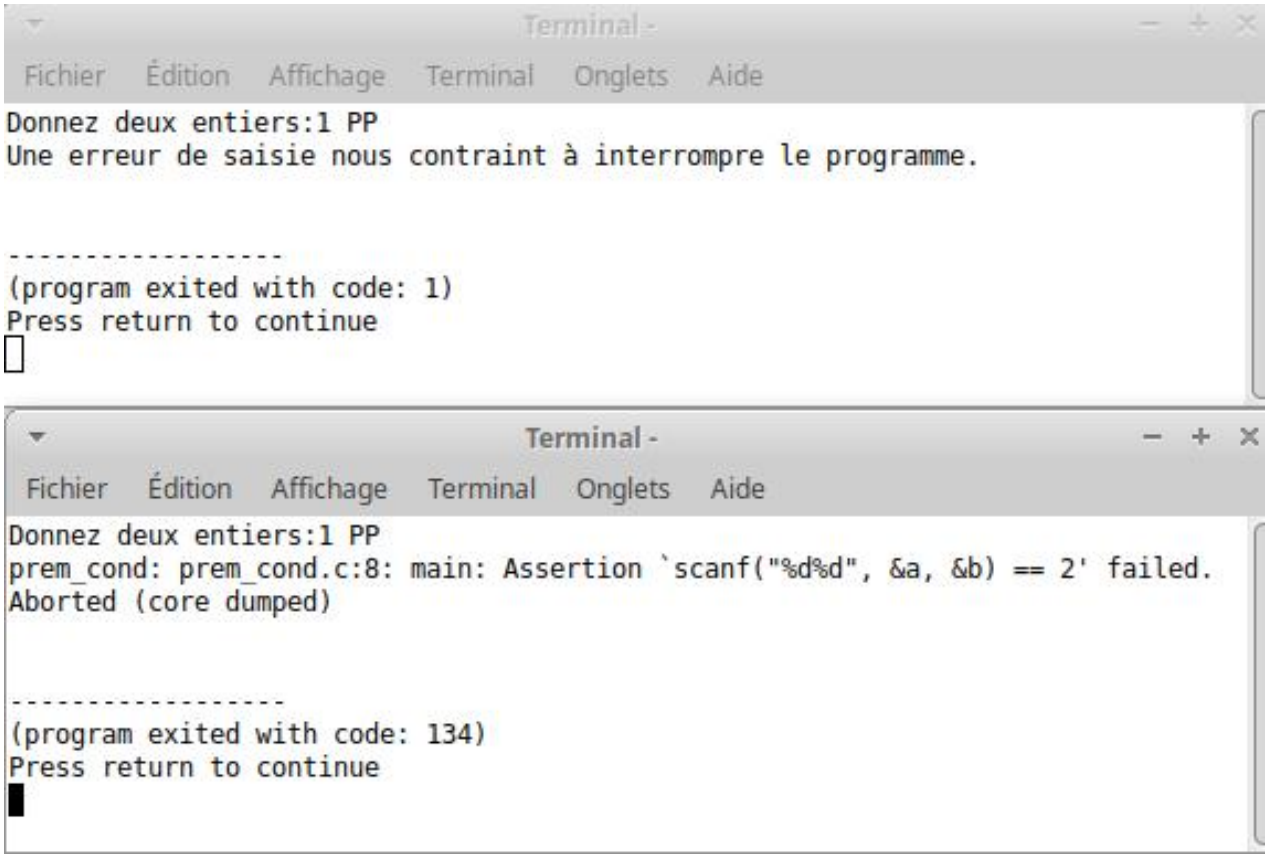
Affichage d'un message personnalisé. À comparer avec

```
assert(scanf("%d%d", &a, &b) == 2);
```

L'instruction `if`

Syntaxe:

```
if (C) {
  I1
}
I2
```

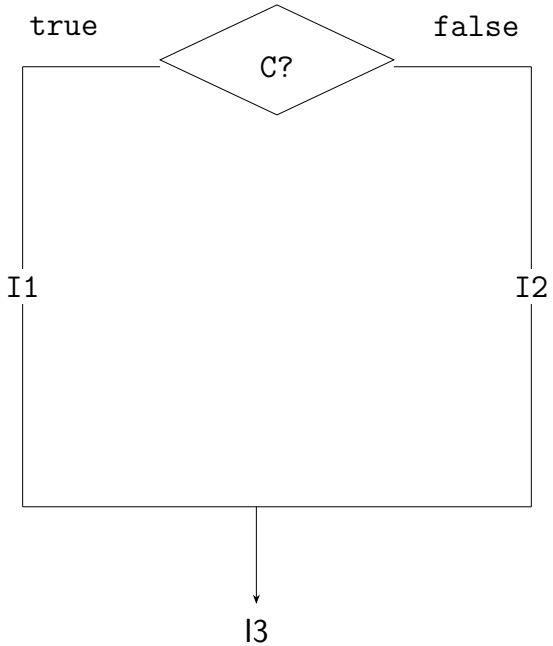


L'instruction `if ... else ...`

Syntaxe:

```
if (C) {
   I1
} else {
   I2
}
```

`C` est une expression booléenne, `I1`, `I2` et `I3` trois blocs d'instructions.
Si l'évaluation de `C` donne `true` alors `I1` est executé, sinon `I2` est exécuté.
Puis, dans les deux cas, `I3` est exécuté.



L'instruction `if ... else ...`

Syntaxe:

```
if (C) {
  I1
} else {
  I2
}
```

Exemple:

```
if ( b != 0) {
  printf("%d / %d == %0.2lf", a, b, (a+0.0) / b);
} else {
  printf("Attention division par 0!");
}
```

L'instruction `if ... else ...`

Les tests peuvent s'imbriquer:

```c
int res;
if (op == '+') {
  res = a + b;
} else {
  if (op == '-') {
    res = a - b;
  } else {
    if (op == '*' || op =='x') {
      res = a * b;
    } else {
        if (op == '/') {
          if ( b != 0) {
            res = a / b;
          } else {
            printf("Attention division par 0!");
            return EXIT_FAILURE;
          }
        } else {
            printf("Opération inconnue.\n");
            return EXIT_FAILURE;
        }
    }
  }
}

printf("%d %c %d == %d\n", a, op, b,res);
```

L'instruction `switch`

Syntaxe:

```
switch (E) {
  case V1:
          I1
          [break;]
  case V2:
          I2
          [break;]
  ...
  case Vk:
          Ik
          [break;]
  [default:
          I']
}
```

x

- `E`: expression produisant un type apparenté aux entiers (`int`, `char` etc.)
- `V1`$\cdots$`Vk`: Constantes du même type que `E`
- `I1`$\cdots$`Ik` et `I`: blocs d'instructions
- `[...]`: parties optionnelles

L'instruction `switch`

Syntaxe:

```
switch (E) {
  case V1:
          I1
          [break;]
  case V2:
          I2
          [break;]
  ...
  case Vk:
          Ik
          [break;]
  [default:
          I']
}
```

+ Si l'évaluation de `E` produit la même valeur que `Vn` alors les blocs suivant `case Vn` sont exécutés jusqu'à ce que l'on rencontre un `break`.

+ Si l'évaluation de `E` ne produit aucune des valeurs `V1`, $\cdots$, `Vk` alors le bloc `I'` est exécuté (si il existe).

L'instruction `switch`

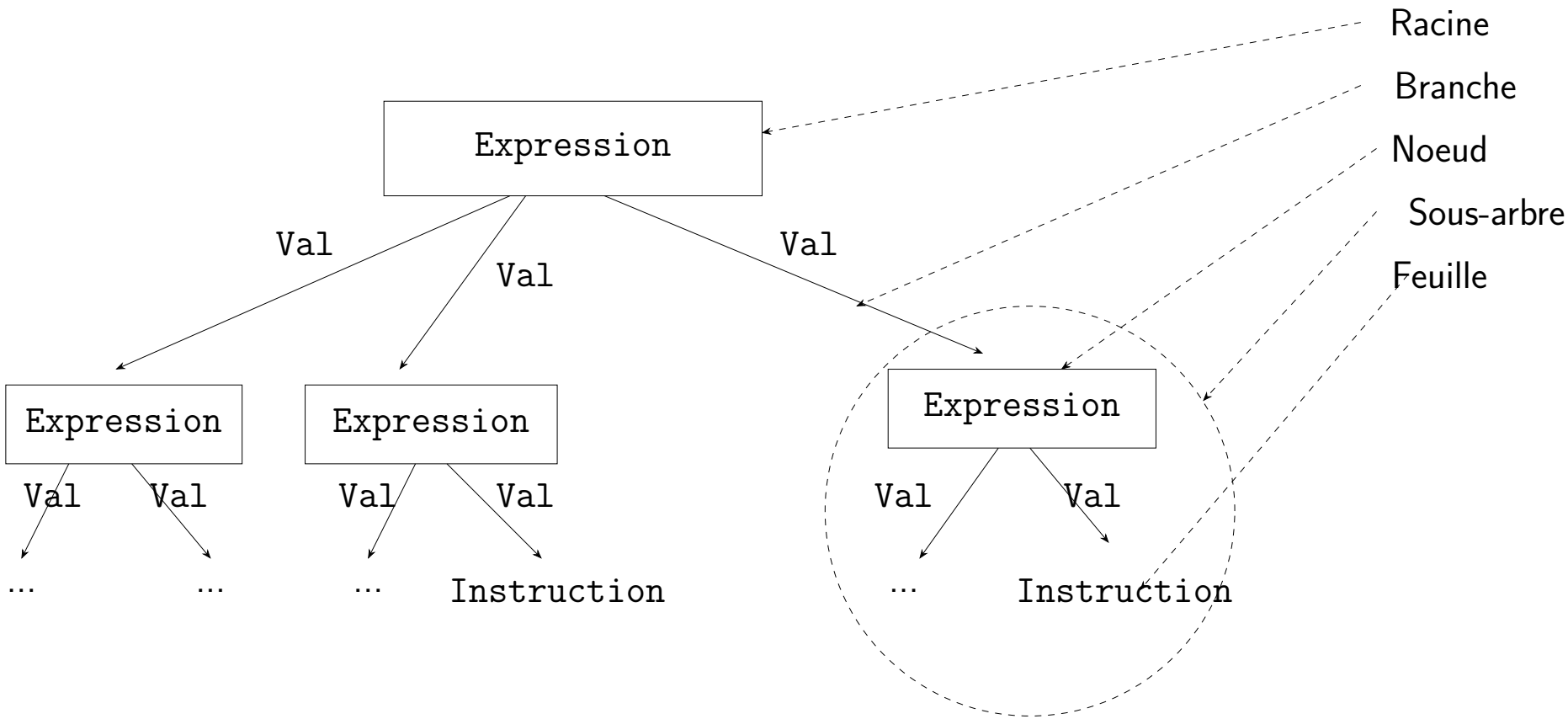
Exemple:

```c
int res;
switch (op) {
  case '+':
      res = a + b;
      break;
  case '-':
      res = a - b;
      break;
  case '*':
  case 'x':
      res = a * b;
      break;
  case '/':
      if (b != 0) {
        res = a / b;
      } else {
        printf("Attention division par 0!");
        return EXIT_FAILURE;
      }
      break;
  default:
      printf("Opération inconnue.\n");
      return EXIT_FAILURE;
}

printf("%d %c %d == %d\n", a, op, b,res);
```

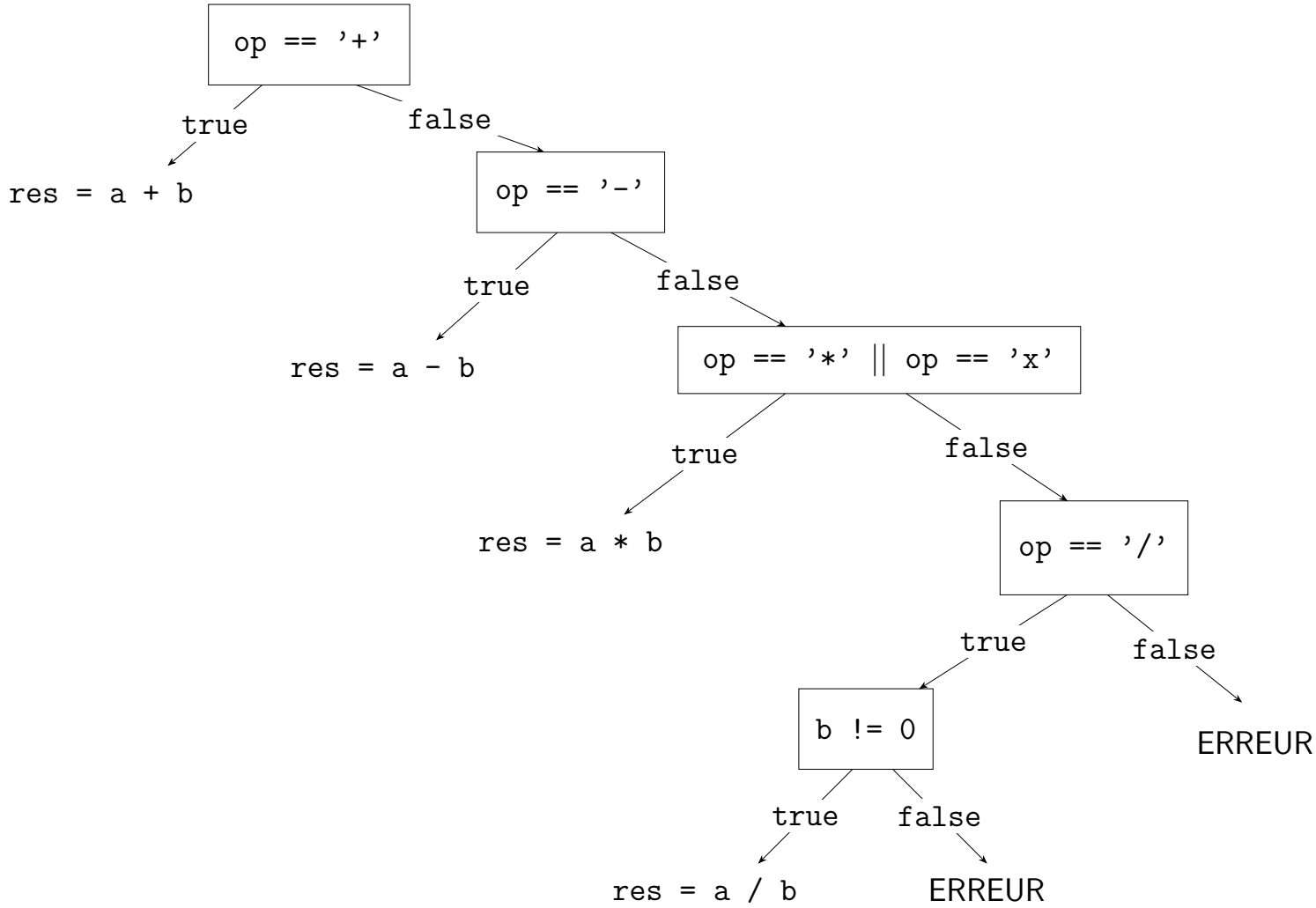
Arbre de décision

Représentation graphique de la structure des tests d'un programme.



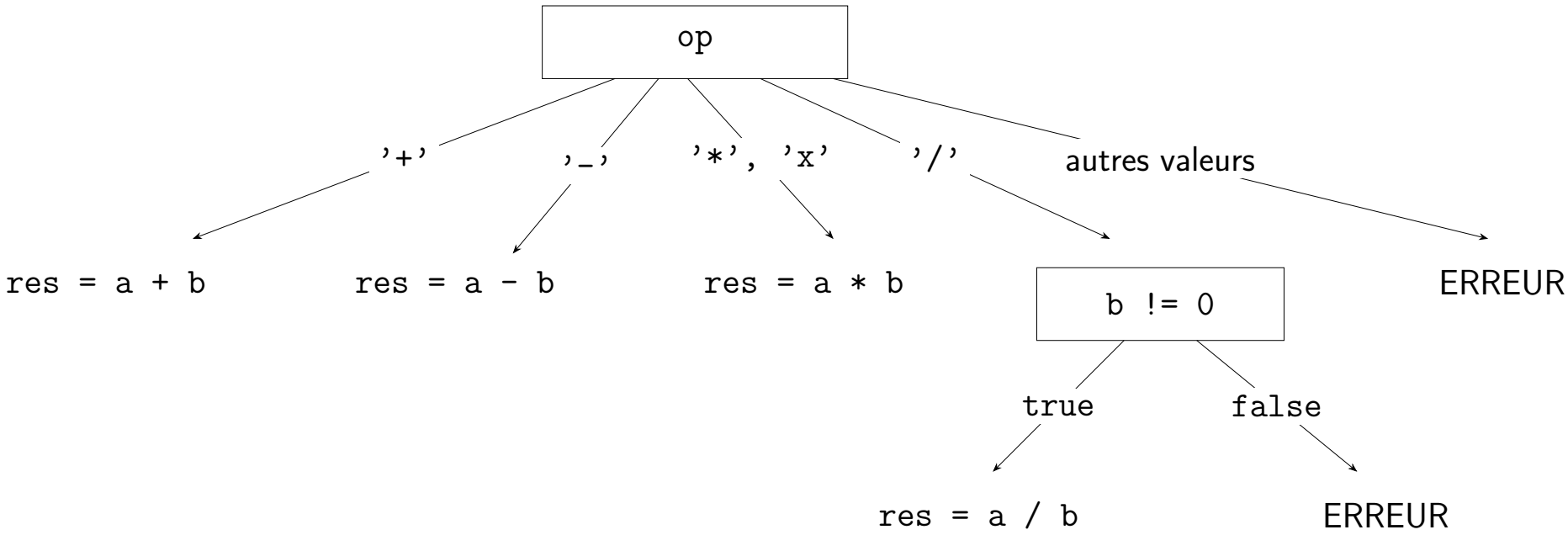
Arbre de décision

Exemple



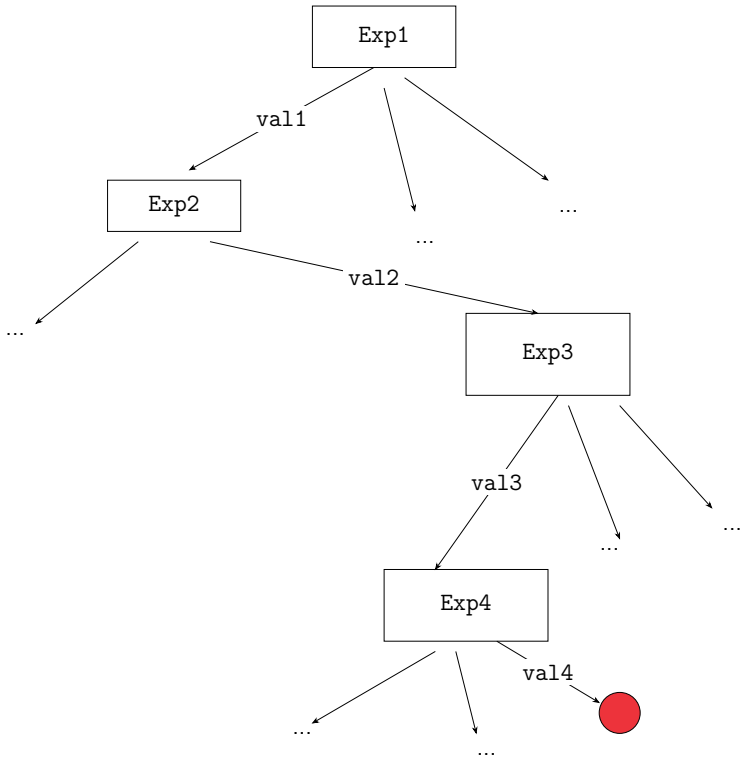
Arbre de décision

Exemple



Équivalence d'arbres

Conjonction

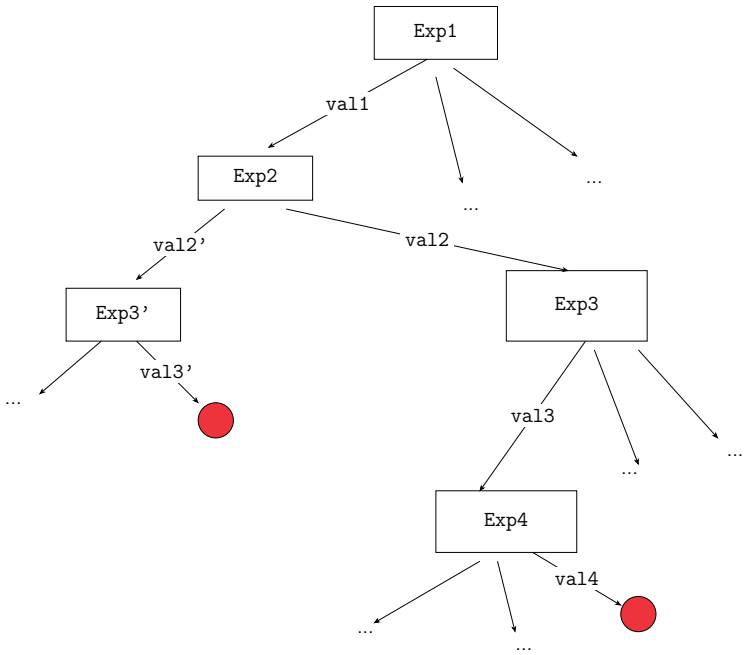


- est exécutée si

`Exp1 == val1 && Exp2 == val2 && Exp3 == val3 && Exp4 == val4.`

Équivalence d'arbres

Disjonction

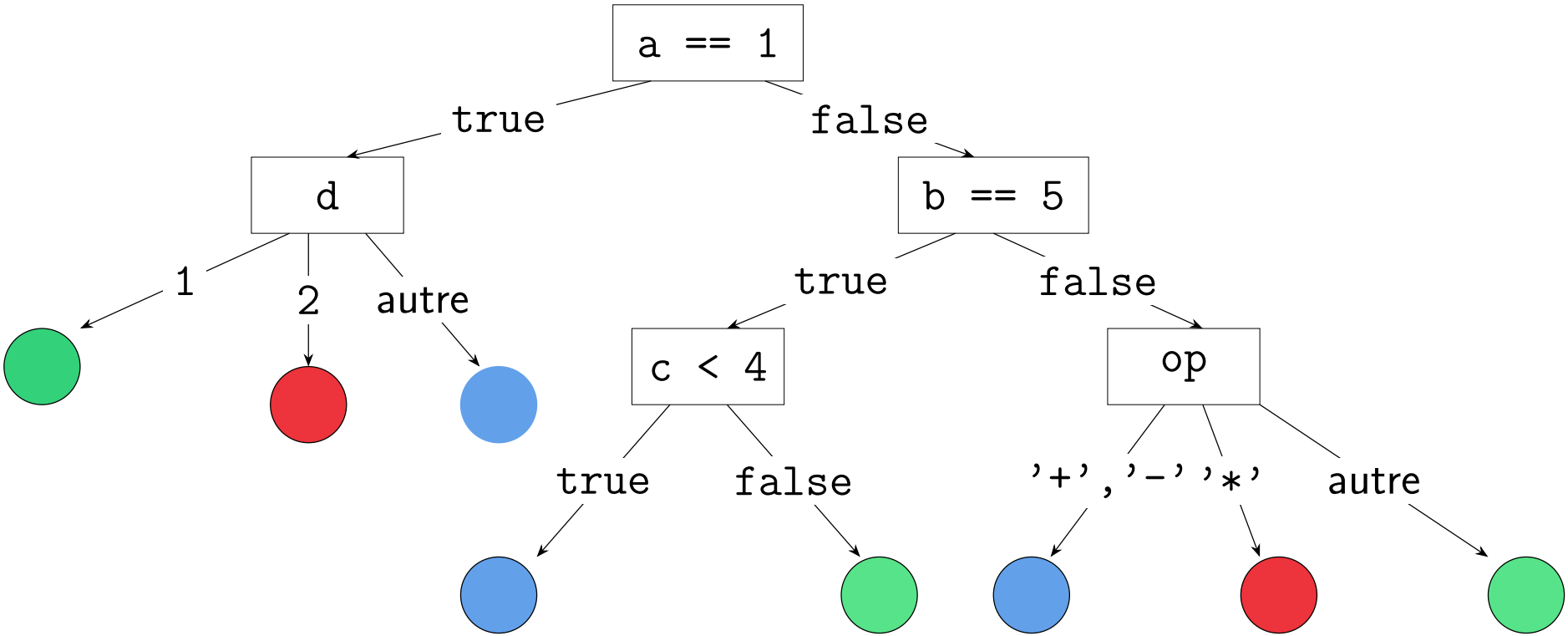


- est exécutée si

```
(Exp1 == val1 && Exp2 == val2' && Exp3' == val3') ||
(Exp1 == val1 && Exp2 == val2 && Exp3 == val3 && Exp4 == val4)
```

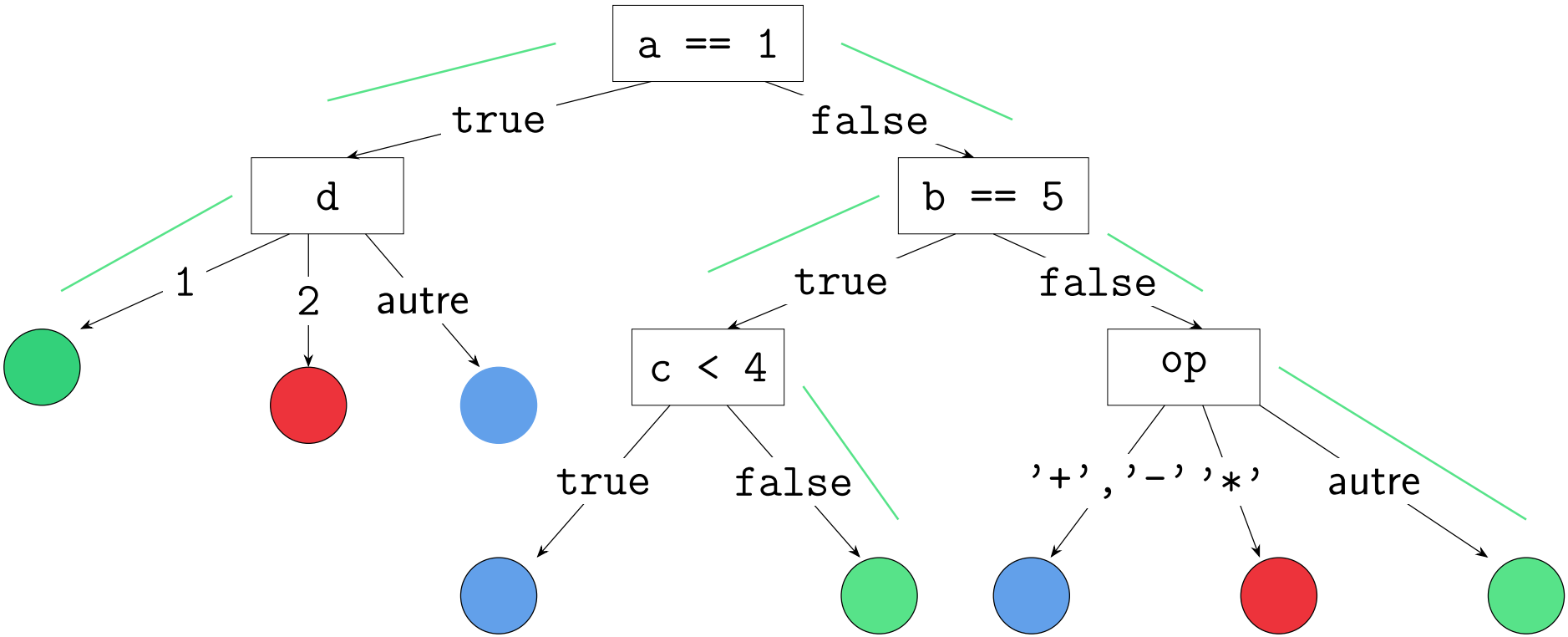
Équivalence d'arbres

Exemple



Équivalence d'arbres

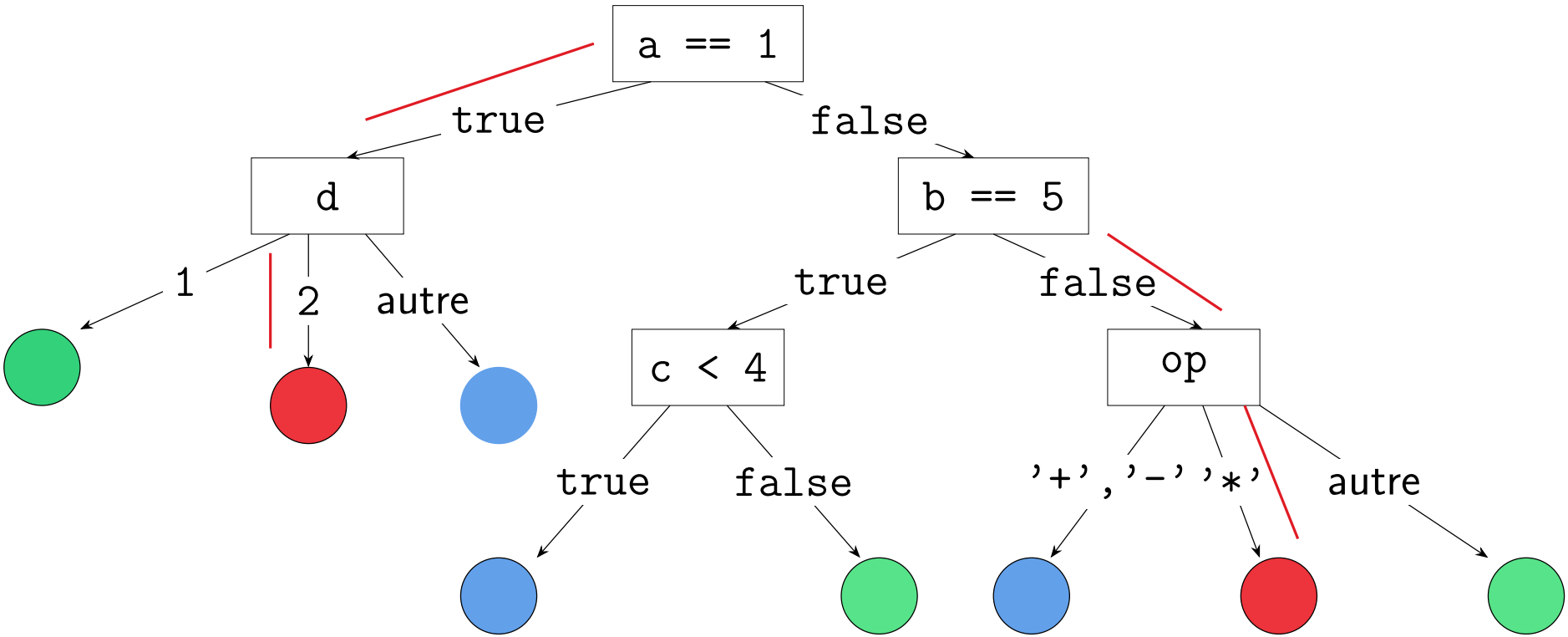
Exemple




```
VERT = (a == 1 && d == 1) ||
                (a != 1 && ((b == 5 && c >= 4) ||
                            (b != 5 && op != '+' &&
                              op != '-' && op != '*')))
```

Équivalence d'arbres

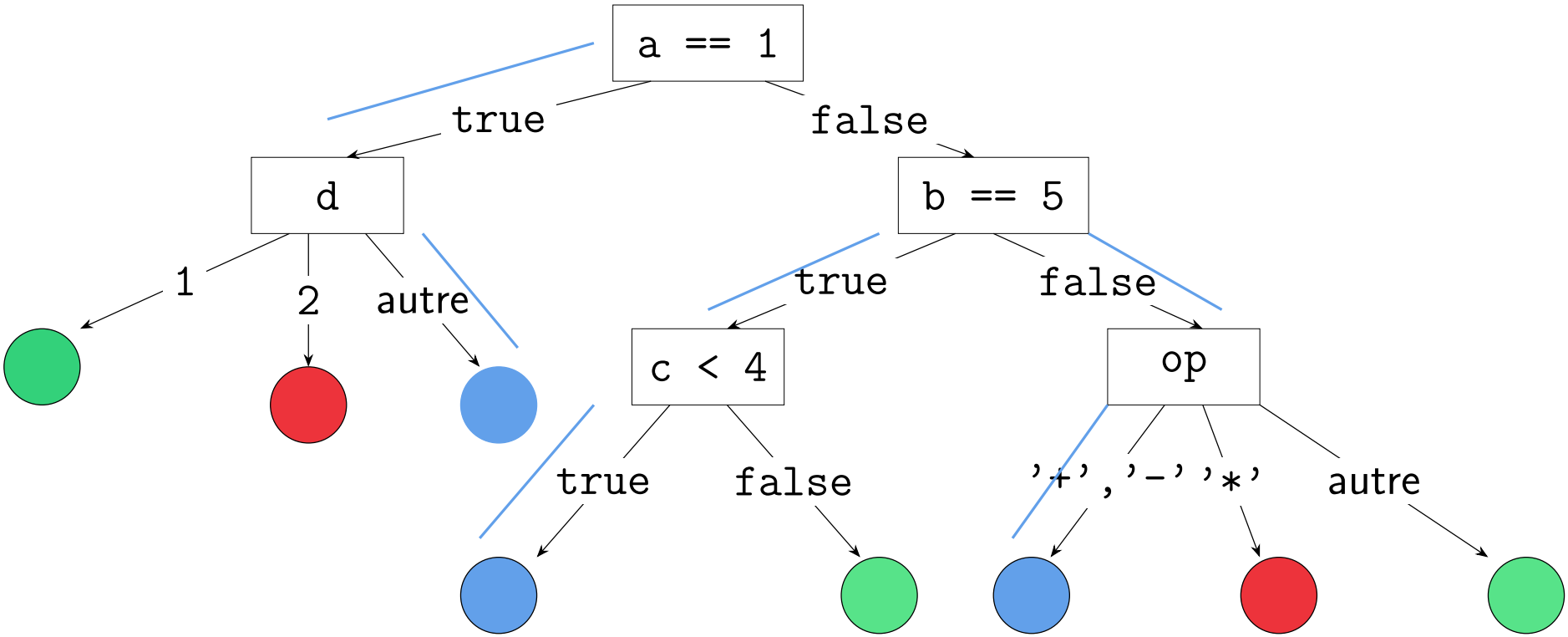
Exemple




```
ROUGE = (a == 1 && d == 2) ||
                (a != 1 && (b != 5 && op == '*"))
```

Équivalence d'arbres

Exemple

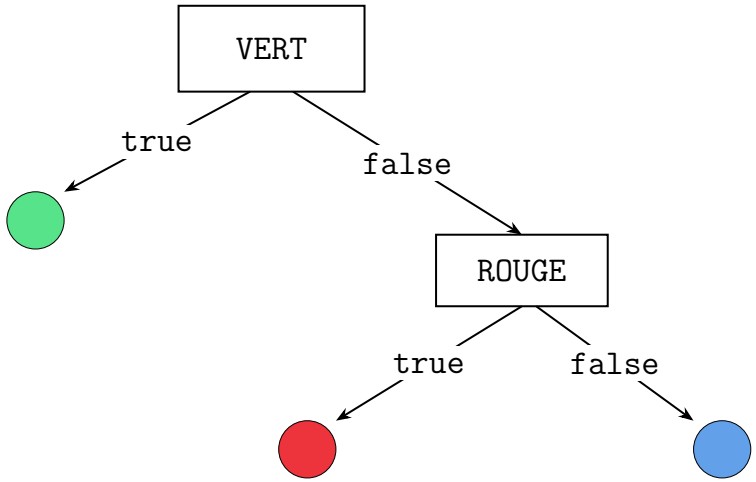


BLEU = ...

INUTILE ! Ce sont les autres cas.

Équivalence d'arbres

Exemple




```
if (VERT) {
  printf("VERT\n");
} else {
    if (ROUGE) {
      printf("ROUGE\n");
    } else {
        printf("BLEU\n");
    }
}
```

```
/*
  if (a == 1) {
    switch (d) {
      case 1:
          printf("Vert");
          break;
      case 2:
          printf("Rouge");
          break;
      default:
          printf("Bleu");
    }
  } else {
      if (b == 5) {
        if (c < 4) {
          printf("Bleu");
        } else {
            printf("Vert");
        }
      } else {
          switch (op) {
            case '+':
            case '-':
              printf("Bleu");
              break;
            case '*':
              printf("Rouge");
              break;
            default:
              printf("Vert");
          }
      }
  }*/
  bool VERT = (a == 1 && d ==1) || (a !=1 && ((b == 5 && c >= 4) || (b!=5 && op != '+' && op != '-' && op != '*')));
  bool ROUGE = (a == 1 && d == 2) || ( a!=1&& b != 5 && op == '*');
  if (VERT) {
    printf("VERT\n");
  } else {
      if (ROUGE) {
        printf("Rouge \n");
      } else {
        printf("Bleu\n");
      }
    }
```

Équivalence d'arbres

Remarques

- Cette méthode n'est valable que si aucune instruction ne vient modifier les variables entre les tests.
- Parfois, on peut ne pas utiliser de `else` en jouant astucieusement avec des `return`.
  Par exemple dans la fonction `main`:

```
if (VERT) {
  printf("VERT\n");
  return EXIT_SUCCESS;
}
if (ROUGE) {
  printf("ROUGE\n");
  return EXIT_SUCCESS;
}
printf("BLEU\n");
```

  Attention, cela implique que le programme termine après avoir affiché `VERT` ou `ROUGE`.