CS 480

Introduction to Artificial Intelligence

February 10, 2021

Announcements / Reminders

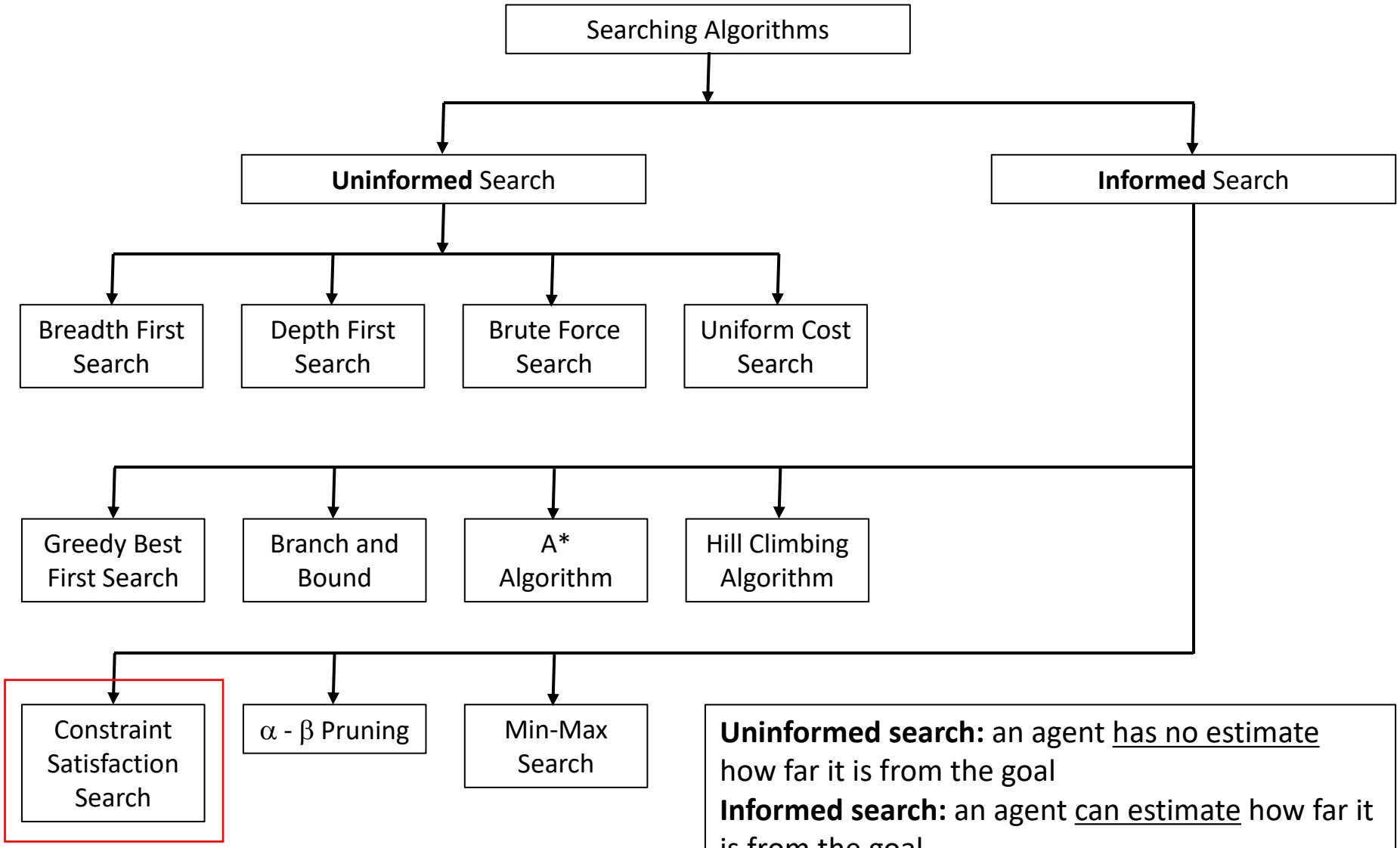
- Written Assignment #01 is posted
  - due on Thursday (02/17/22) at 11:00 PM CST
- Contribute to the discussion on Blackboard, please
- Please follow the Week 05 To Do List instructions

- Midterm course review will be available next week
  - Please respond. Thank you!

Plan for Today

- **Constraint Satisfaction Problems: Continued**

Selected Searching Algorithms



Uninformed search: an agent has no estimate how far it is from the goal
Informed search: an agent can estimate how far it is from the goal

Constraint Satisfaction Problem

A Constraint Satisfaction Problem (CSP) consists of three components:

- **a set of variables** $X = \{X_1, ..., X_n\}$
- **a set of domains** $D = \{D_1, ..., D_n\}$
- **a set of constraints** C **that specify allowable combinations of values**

- **A domain** D_i **is a set of allowable values** $\{v1, ..., vk\}$ **for variable** X_i
- **A constraint** C_j **is a** $\langle \text{scope, relation} \rangle$ **pair, for example** $\langle (X1, X2), X1 > X2 \rangle$

Constraint Satisfaction Problem

The goal is to find an assignment (variable = value):

$$\{X_1 = v_1, \ldots, X_n = v_n\}$$

- **If NO constraints violated: consistent assignment**
- **If ALL variables have a value: complete assignment**
- **If SOME variables have NO value: partial assignment**
- **SOLUTION: consistent and complete assignment**
- **PARTIAL SOLUTION: consistent and partial assignment**

CSP: Variable Types

- **Domains can be:**
  - **finite, for example:** $\{1, 2, 3, 5, 8, 20\}$ **(simpler)**
  - **infinite, for example: a set of all integers**
- **Variables can be:**
  - **discrete, for example:** $X = \{X_1, ..., X_n\}$ **(simpler)**
  - **continuous, for example:** R_+
- **Constraints can be:**
  - **unary (involve single variable), for example:** $X_1 = 5$
  - **binary (involve two variables), for example:** $X_1 = X_2$
  - **higher order (involve > 2 variables), for example:** $X_1 = X_2 * X_3$
- **Soft constraints (preferences: green over blue) possible**

CSP as a Search Problem

CSP is a variant of a search problem you already know. The problem can be restated / updated with:

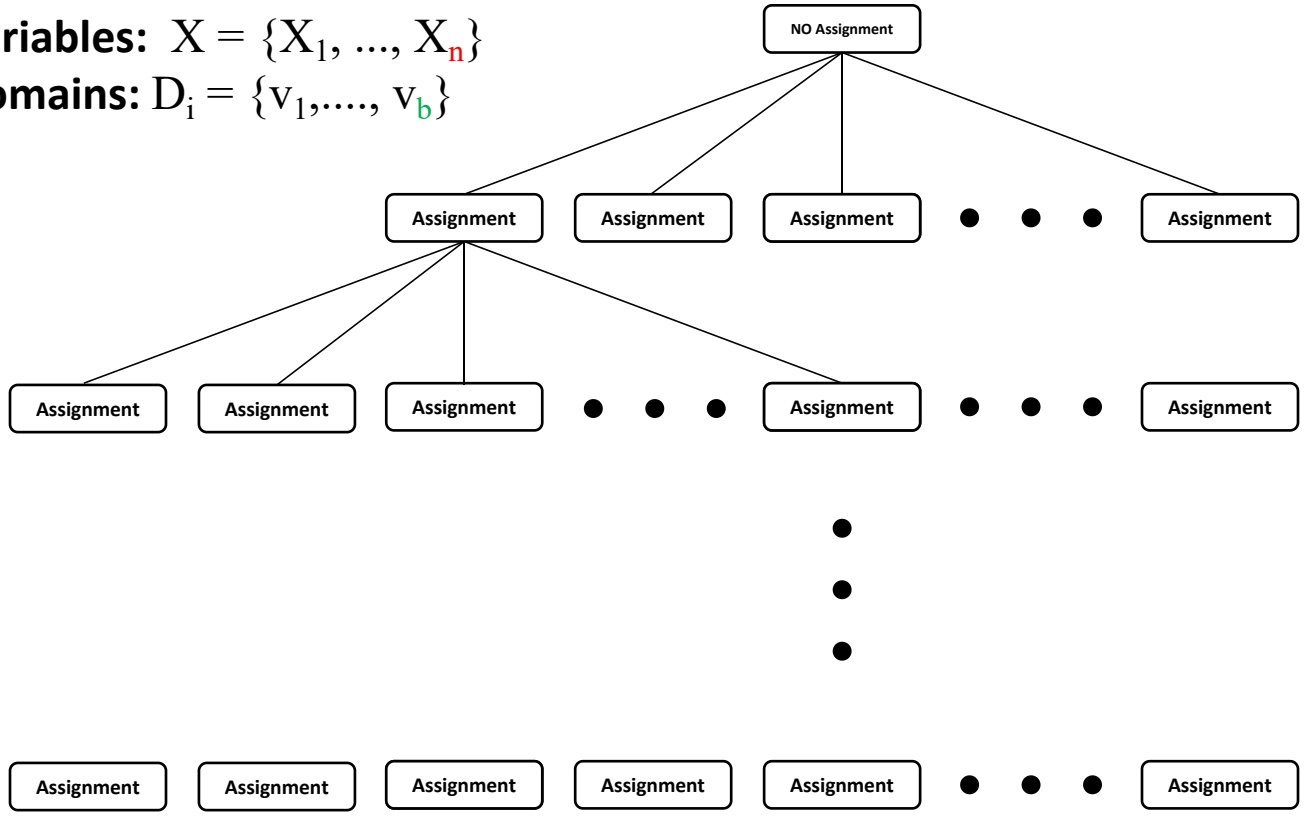
- **Initial state: the empty assignment { }, in which all variables are unassigned.**
- **Successor function: a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables.**
- **Goal test: the current assignment is complete.**
- **Path cost: a constant cost (e.g., 1) for every step.**

CSP Search Tree: Idea

CSP Problem:

Variables: $X = \{X_1, ..., X_n\}$

Domains: $D_i = \{v_1, ..., v_b\}$



0 variable assigned

1 variables assigned

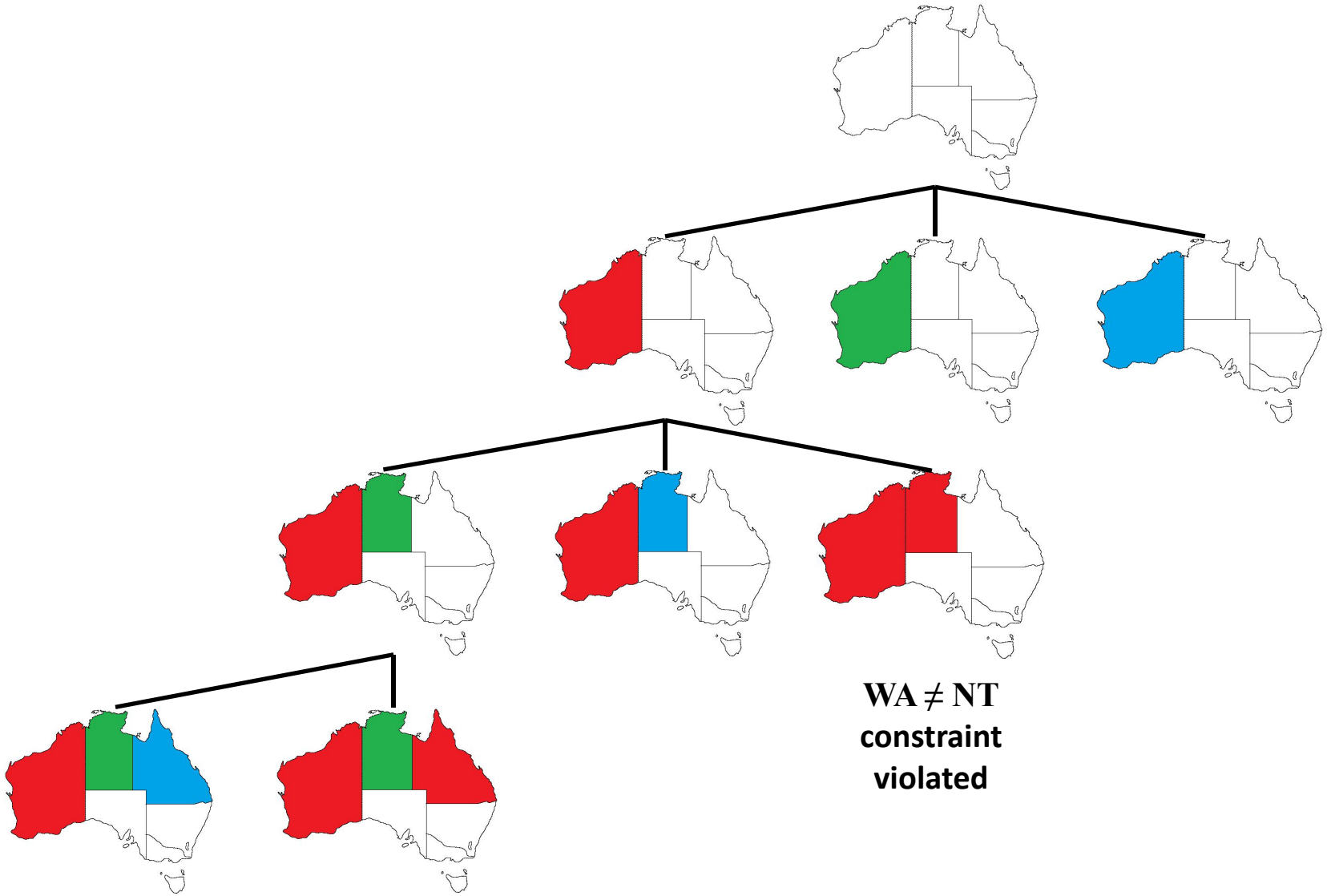
2 variables assigned

ALL (n) variables assigned

Tree leaves are COMPLETE assignments

The sequence of variable assignments does NOT matter*
***(when you disregard performance)**

CSP as a Tree Search Problem



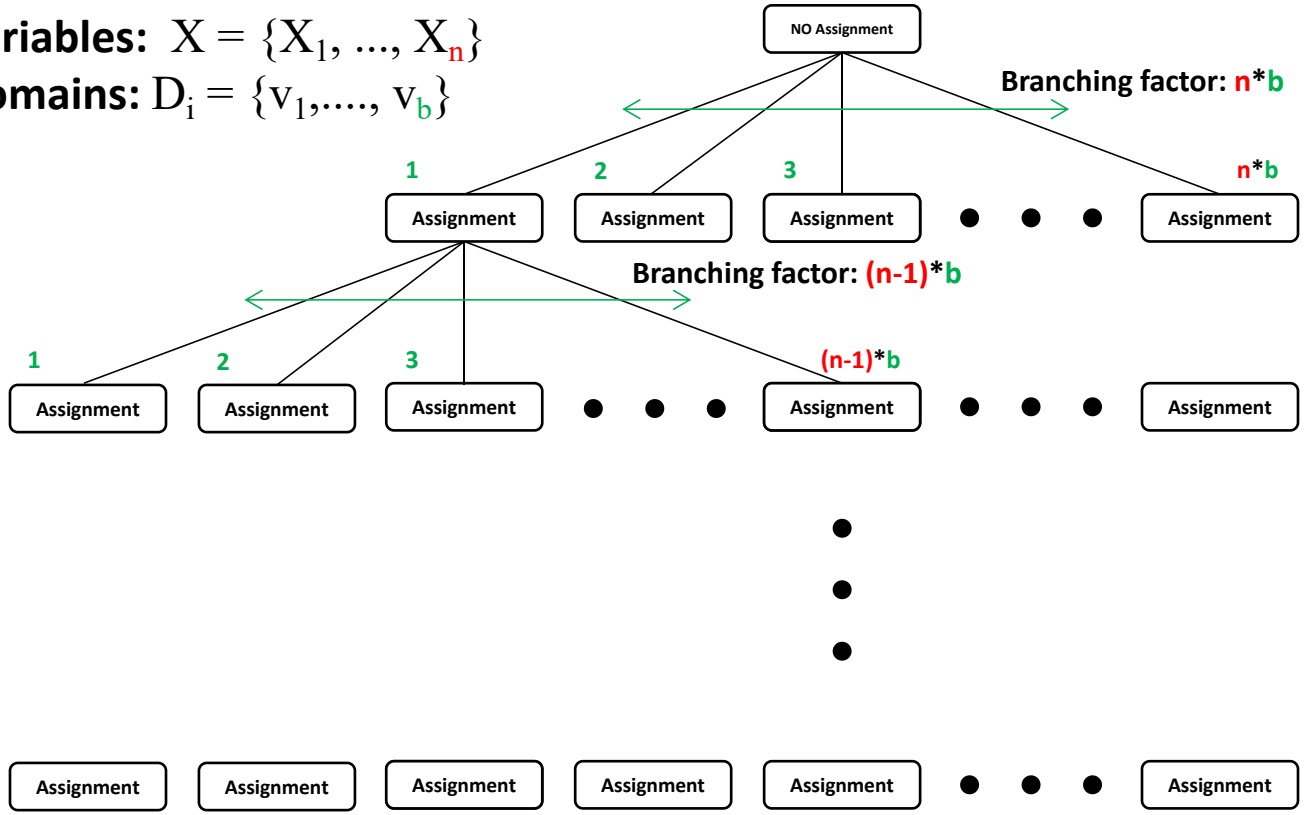
WA ≠ NT
constraint
violated

CSP Search Tree: Size

CSP Problem:

Variables: $X = \{X_1, ..., X_n\}$

Domains: $D_i = \{v_1, ..., v_b\}$



$N_0 = 0$

$N_1 = n*b$

$N_2 = n*b*(n-1)*b = n*(n-1)*b^2$

$N_n = n! * b^n$

Total number of leafnodes / states: $n! * b^n$

(ignores COMMUTATIVITY of CSP assignments:

assigning $X_1 = m$ and then $X_2 = n$ SAME as assigning $X_2 = n$ and then $X_1 = m$)

In reality: there is only b^n complete assignments

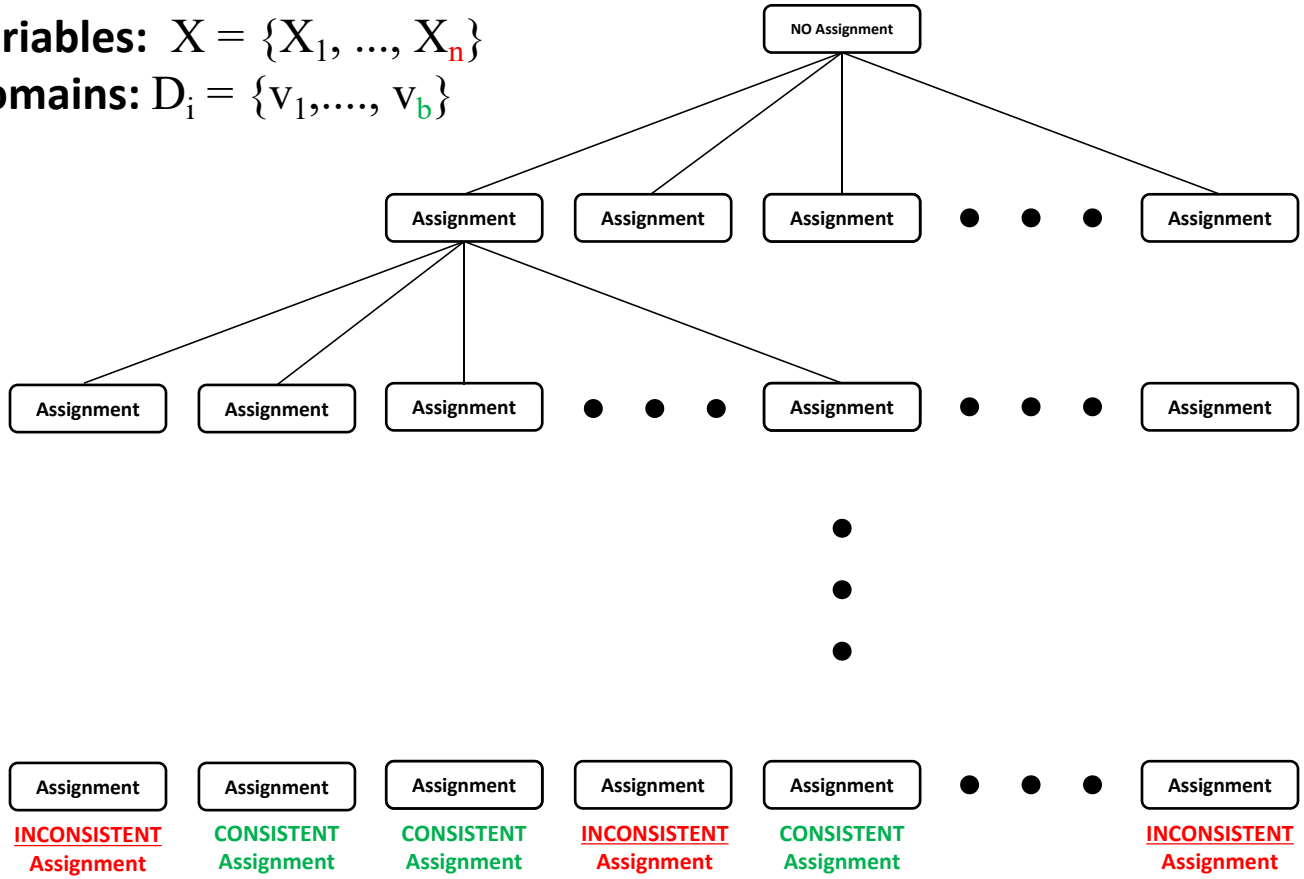
Can We Do Better?

CSP Search Tree: Solutions

CSP Problem:

Variables: $X = \{X_1, ..., X_n\}$

Domains: $D_i = \{v_1, ..., v_b\}$

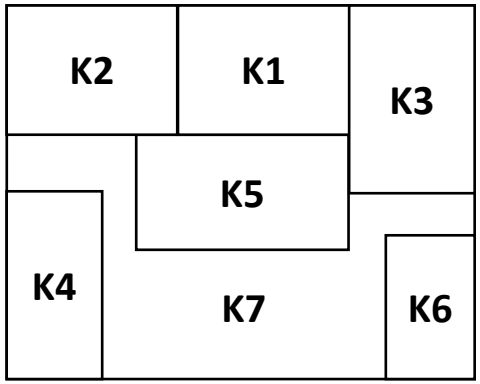


Some nodes / states will be CONSISTENT, while others will be INCONSISTENT.

Depth first search could possibly visit them all → WASTEFUL.

CSP Example: Map Coloring

Problem:



Variables:

$X = \{K1, K2, K3, K4, K5, K6, K7\}$

Variable Domains:

$D_{K1} = \{RED, BLUE, GREEN\}$
$D_{K2} = \{RED, BLUE, GREEN\}$
$D_{K3} = \{RED, BLUE, GREEN\}$
$D_{K4} = \{RED, BLUE, GREEN\}$
$D_{K5} = \{RED, BLUE, GREEN\}$
$D_{K6} = \{RED, BLUE, GREEN\}$
$D_{K7} = \{RED, BLUE, GREEN\}$

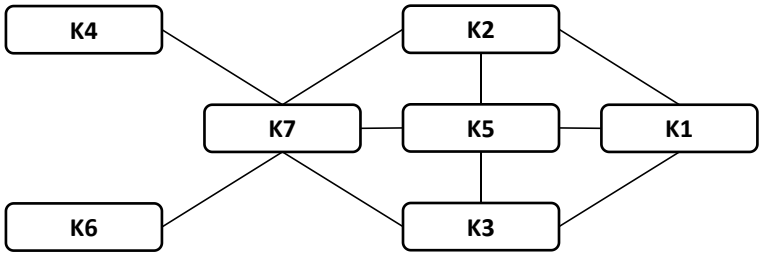
Color this map in a way that no two neighbors have same color

Constraints (Rules):

- **Neighboring regions have to have DISTINCT colors:**

$CONSTRAINTS = C = \{K1 \neq K2, K1 \neq K3, K1 \neq K5, K2 \neq K5, K2 \neq K7, K3 \neq K5, K3 \neq K7, K4 \neq K7, K5 \neq K7, K6 \neq K7\}$

Constraint Graph:

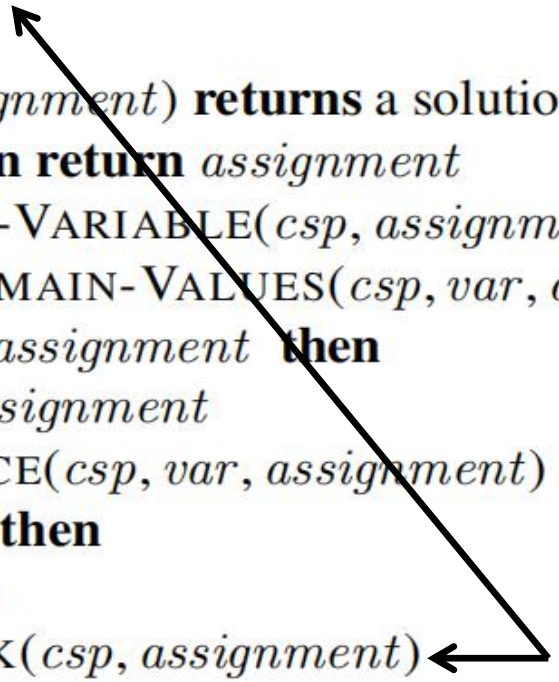


CSP Backtracking: Pseudocode

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
    return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
            remove {var = value} from assignment
    return failure
```

CSP Backtracking: Pseudocode

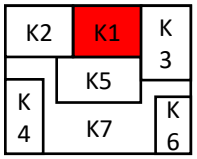
```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
    return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
            remove {var = value} from assignment
    return failure
```



RECURSION

Assignment:
K1: RED
K2: ???
K3: ???
K4: ???
K5: ???
K6: ???
K7: ???

Initial (NO assignment) state not shown



Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order: RED, BLUE, GREEN**

Assignment:

K1: RED

K2: RED

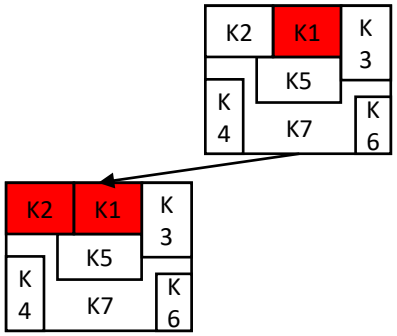
K3: ???

K4: ???

K5: ???

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: RED

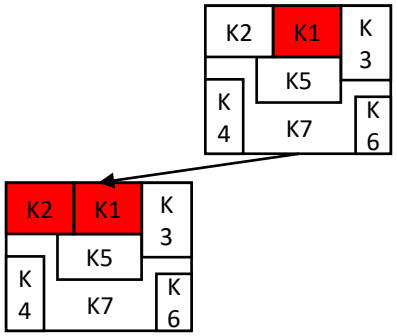
K3: ???

K4: ???

K5: ???

K6: ???

K7: ???



Violates: Rule 1

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order: RED, BLUE, GREEN**

Assignment:

K1: RED

K2: ???

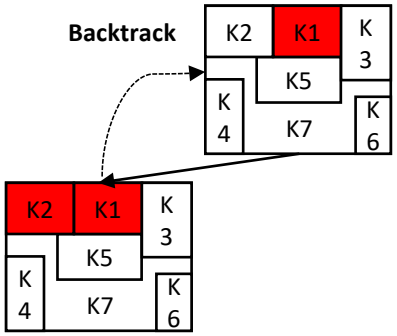
K3: ???

K4: ???

K5: ???

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

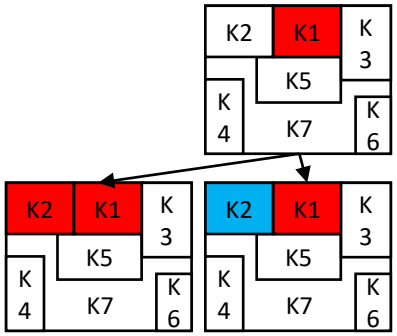
K3: ???

K4: ???

K5: ???

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order: RED, BLUE, GREEN**

Assignment:

K1: RED

K2: BLUE

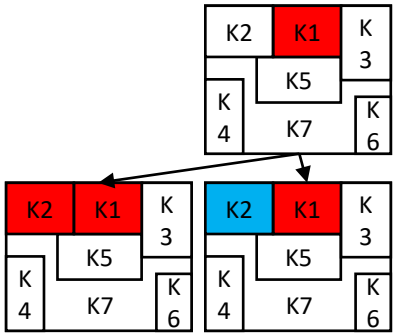
K3: ???

K4: ???

K5: ???

K6: ???

K7: ???



CONSISTENT PARTIAL Assignment

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order: RED, BLUE, GREEN**

Assignment:

K1: RED

K2: BLUE

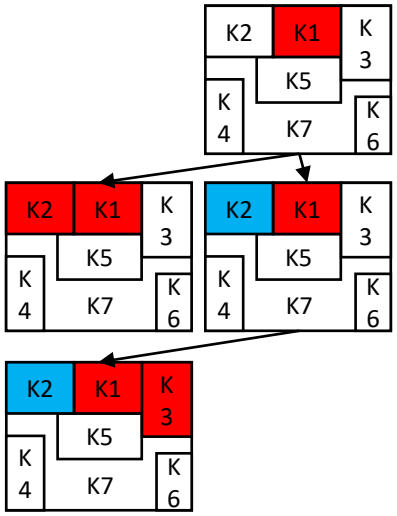
K3: RED

K4: ???

K5: ???

K6: ???

K7: ???



Violates: Rule 2

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

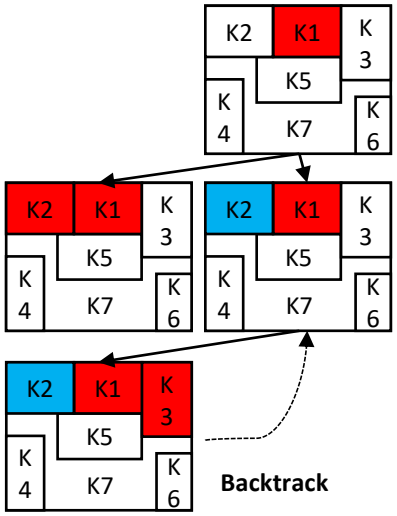
Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order: RED, BLUE, GREEN**

Assignment:
K1: RED
K2: BLUE
K3: ???
K4: ???
K5: ???
K6: ???
K7: ???



Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

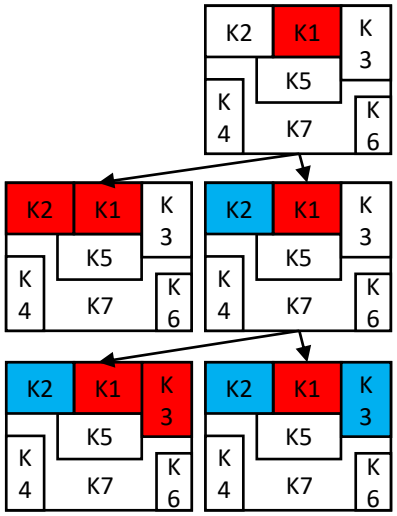
K3: BLUE

K4: ???

K5: ???

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order: RED, BLUE, GREEN**

Assignment:

K1: RED

K2: BLUE

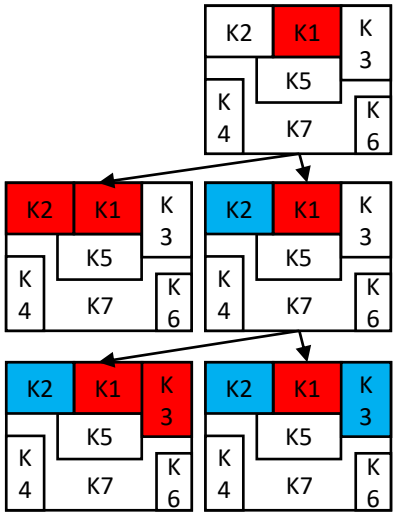
K3: BLUE

K4: ???

K5: ???

K6: ???

K7: ???



CONSISTENT PARTIAL Assignment

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order: RED, BLUE, GREEN**

Assignment:

K1: RED

K2: BLUE

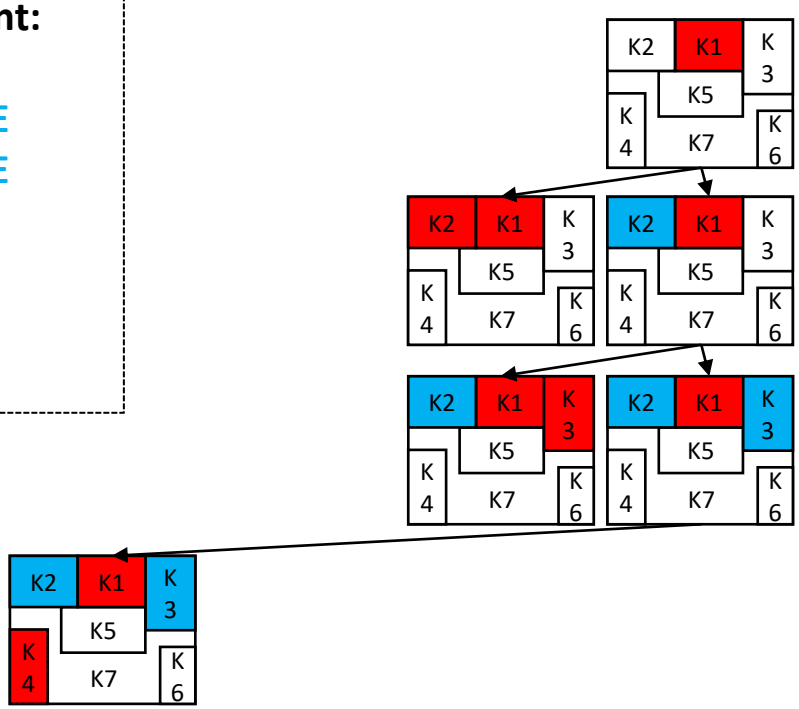
K3: BLUE

K4: RED

K5: ???

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

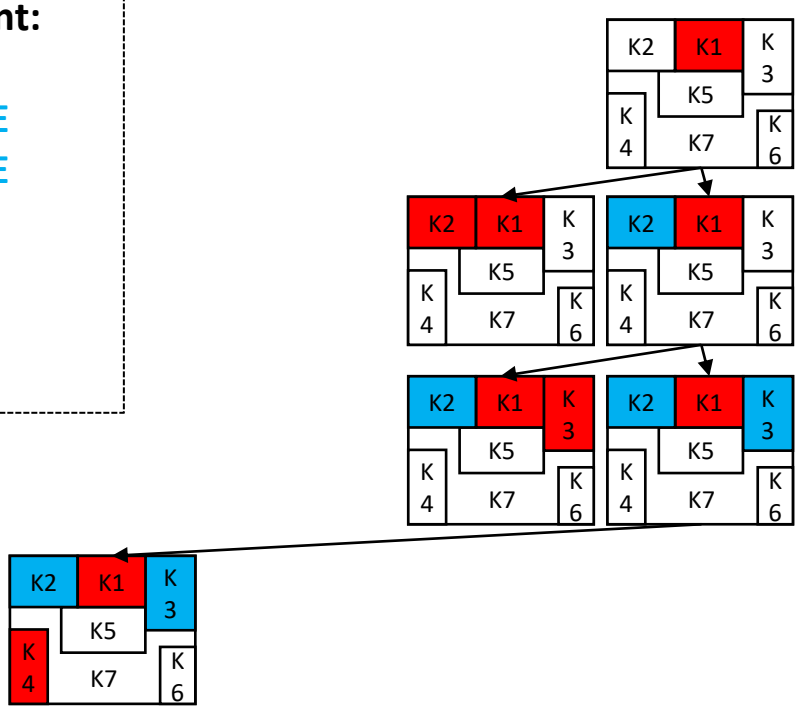
K3: BLUE

K4: RED

K5: ???

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

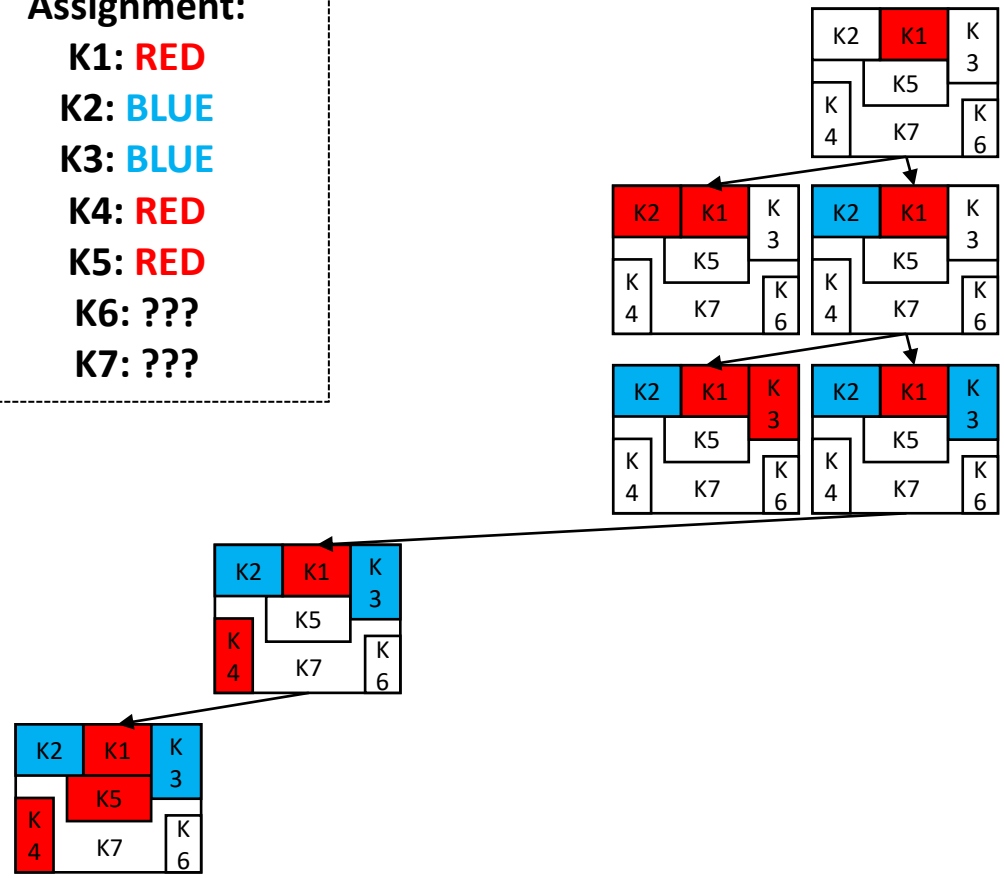
K3: BLUE

K4: RED

K5: RED

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

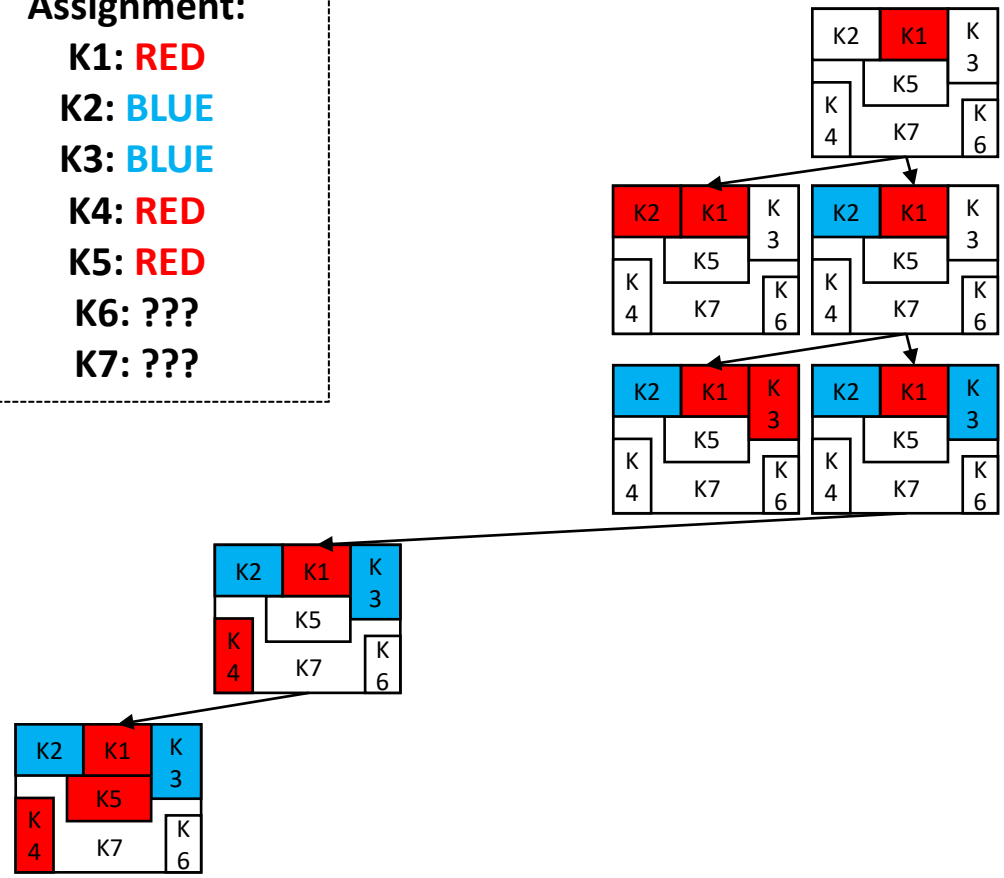
K3: BLUE

K4: RED

K5: RED

K6: ???

K7: ???



Violates: Rule 3

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

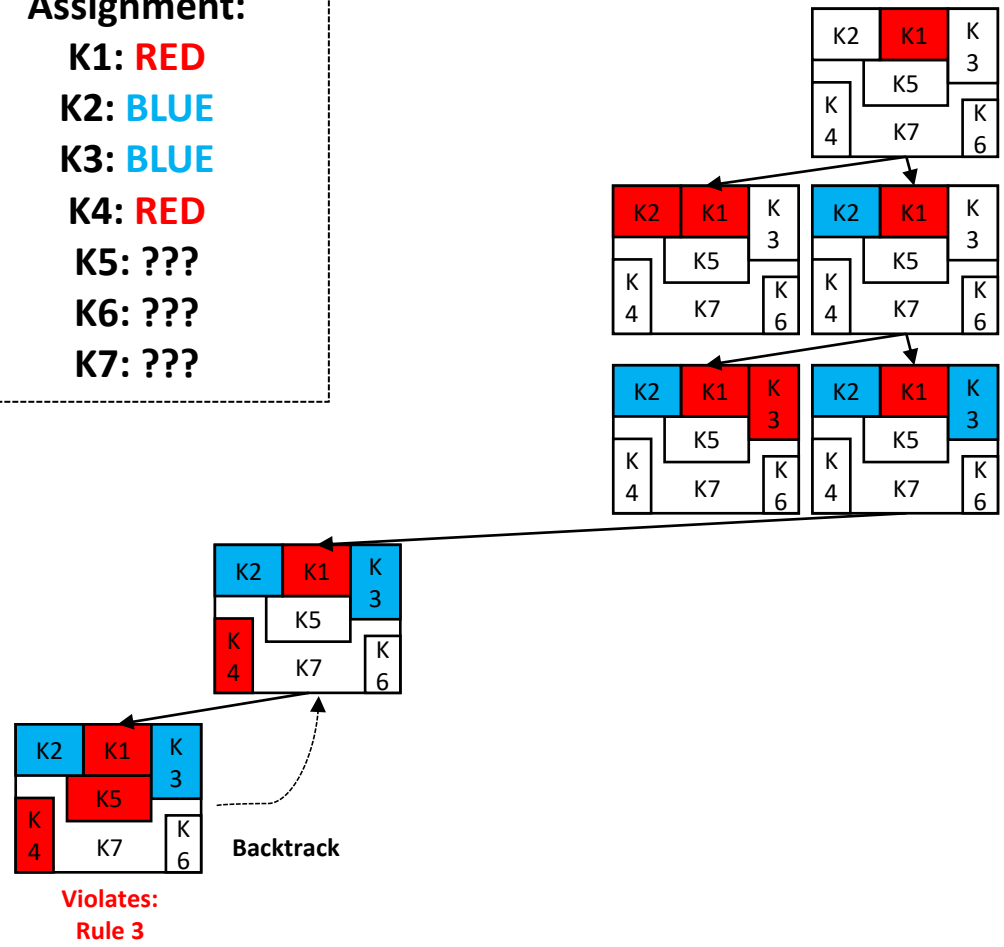
K3: BLUE

K4: RED

K5: ???

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Assignment:

K1: RED

K2: BLUE

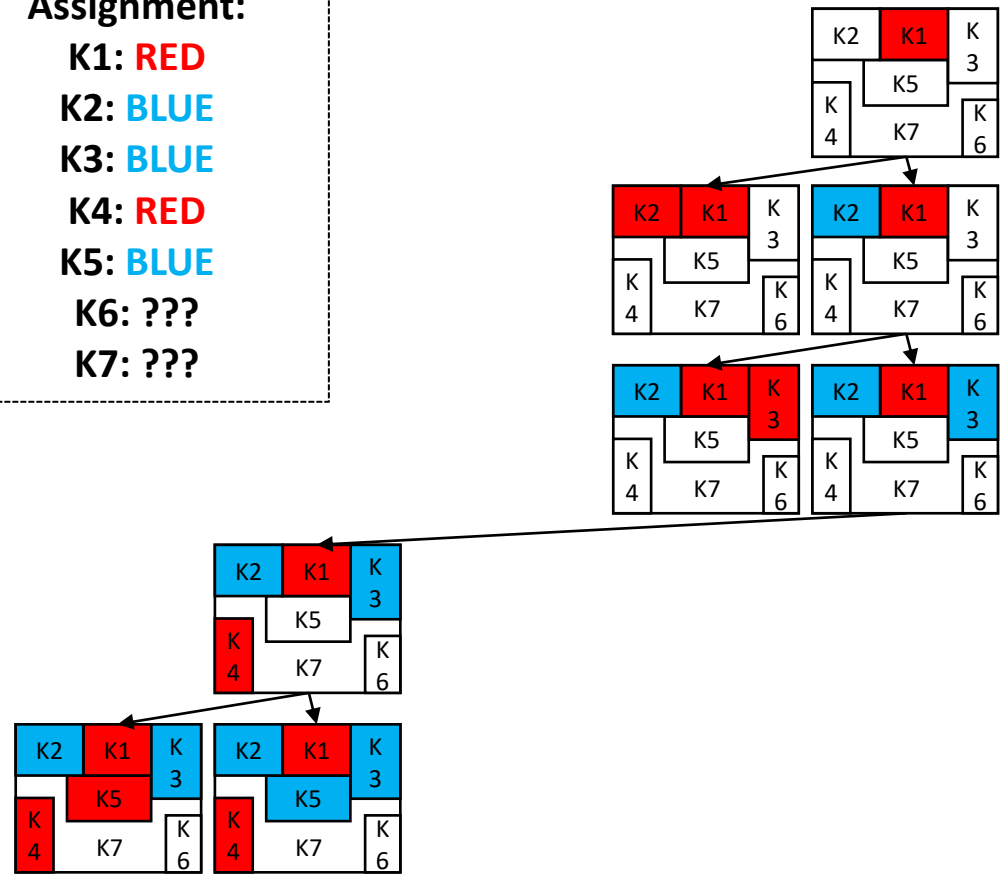
K3: BLUE

K4: RED

K5: BLUE

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

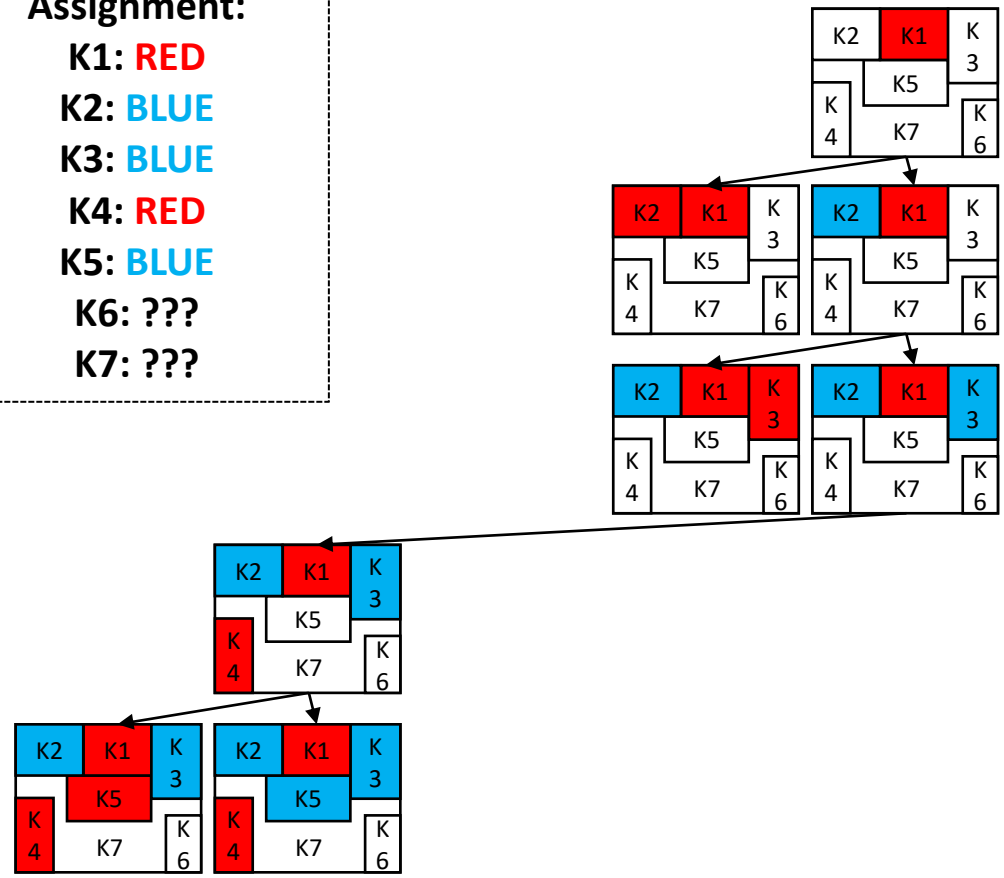
K3: BLUE

K4: RED

K5: BLUE

K6: ???

K7: ???



Violates:
Rule 4
Rule 6

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

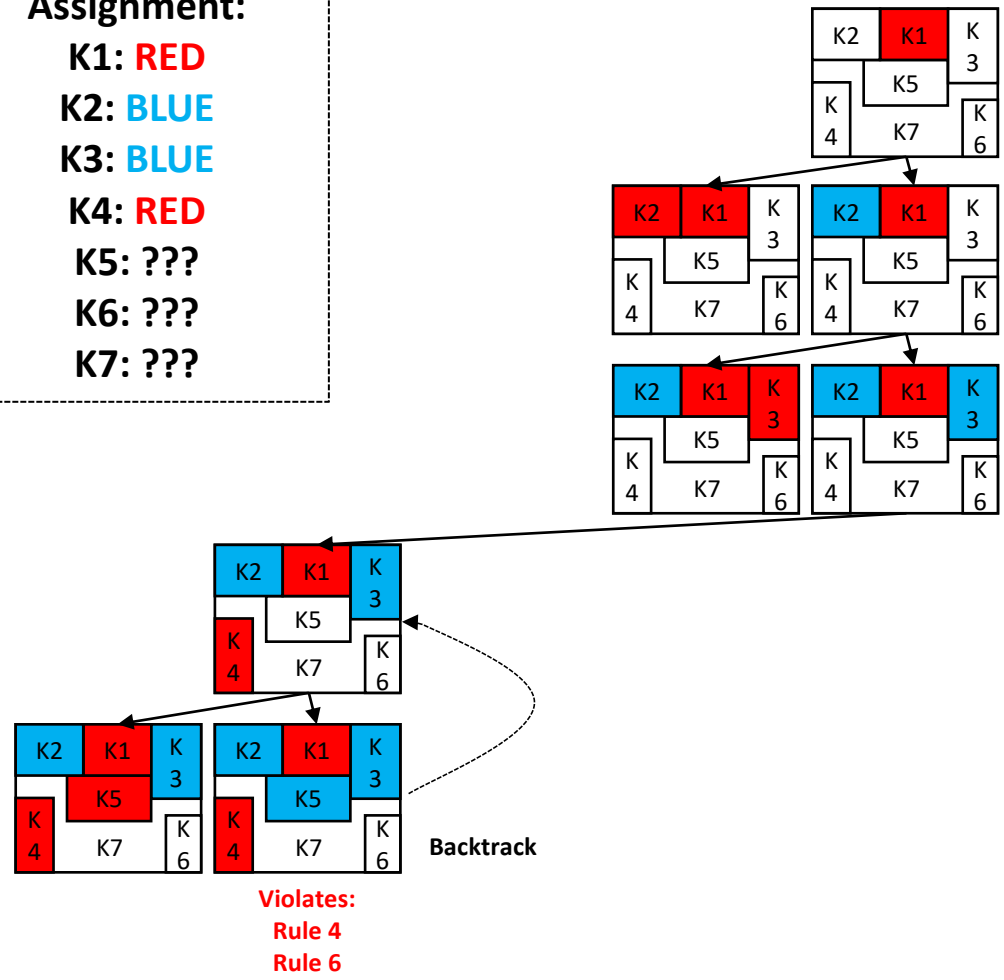
K3: BLUE

K4: RED

K5: ???

K6: ???

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

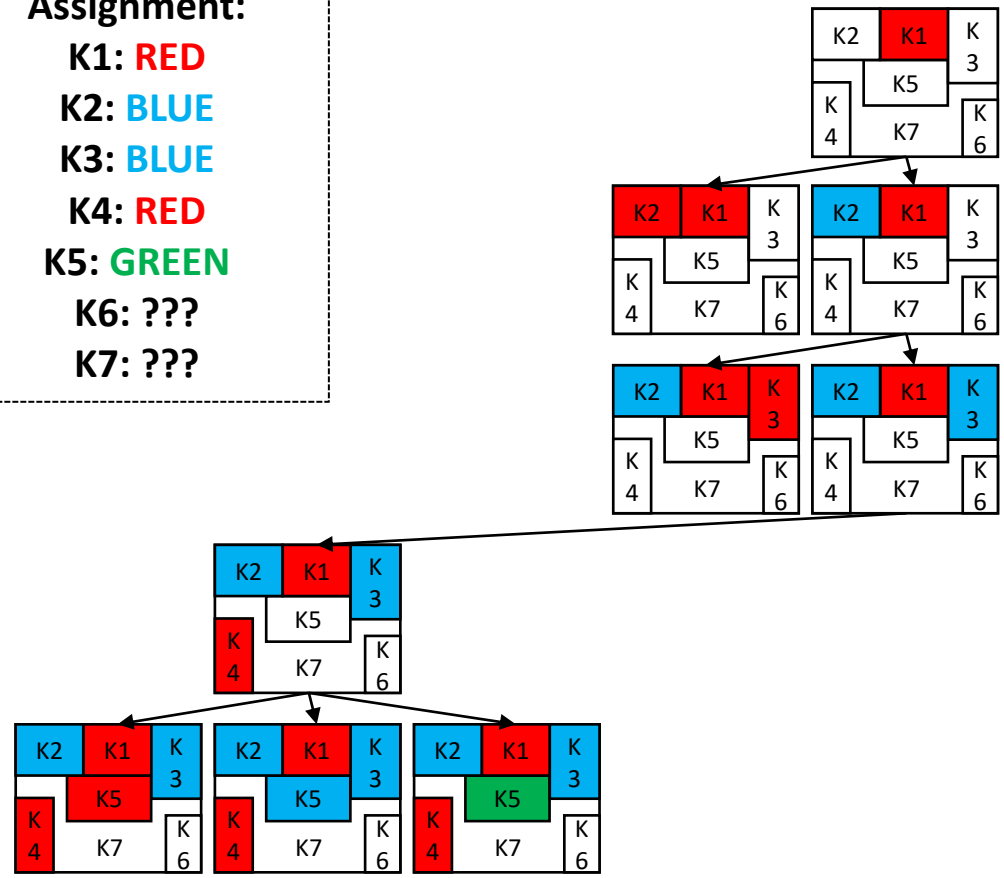
Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

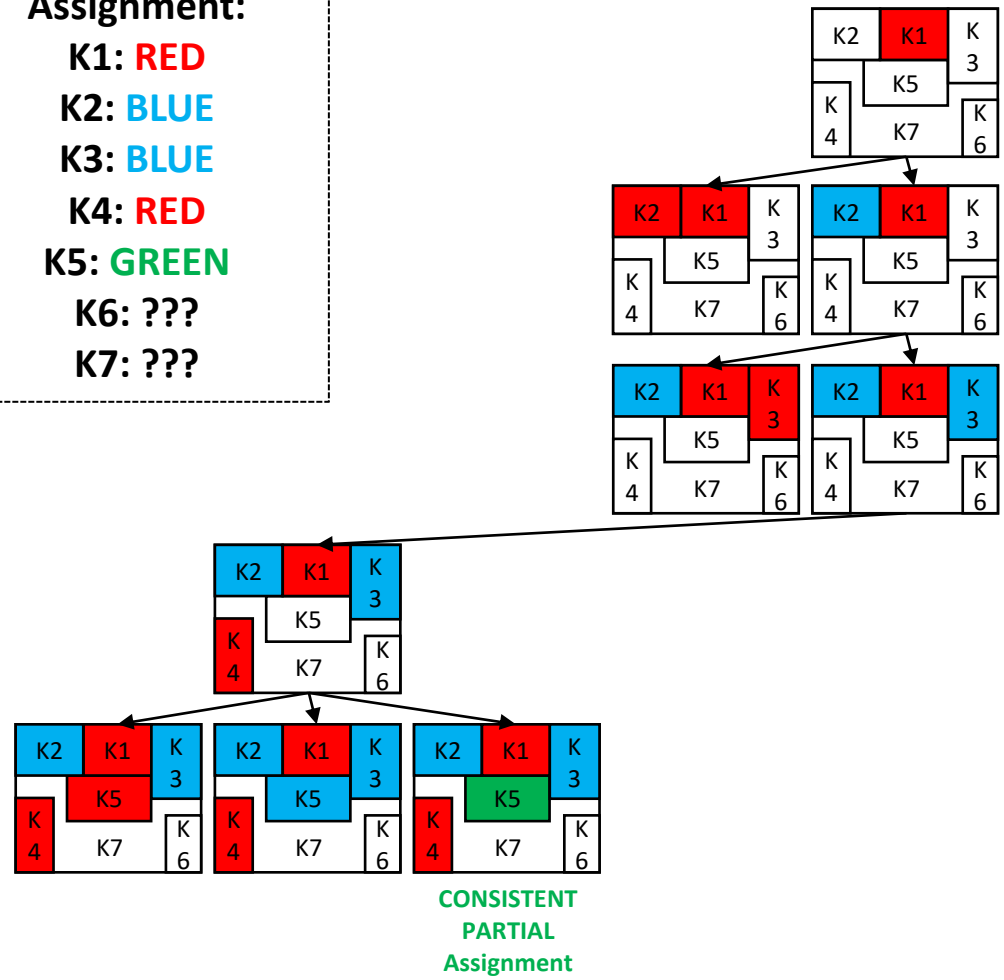
Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: ???
K7: ???



Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: ???
K7: ???



Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

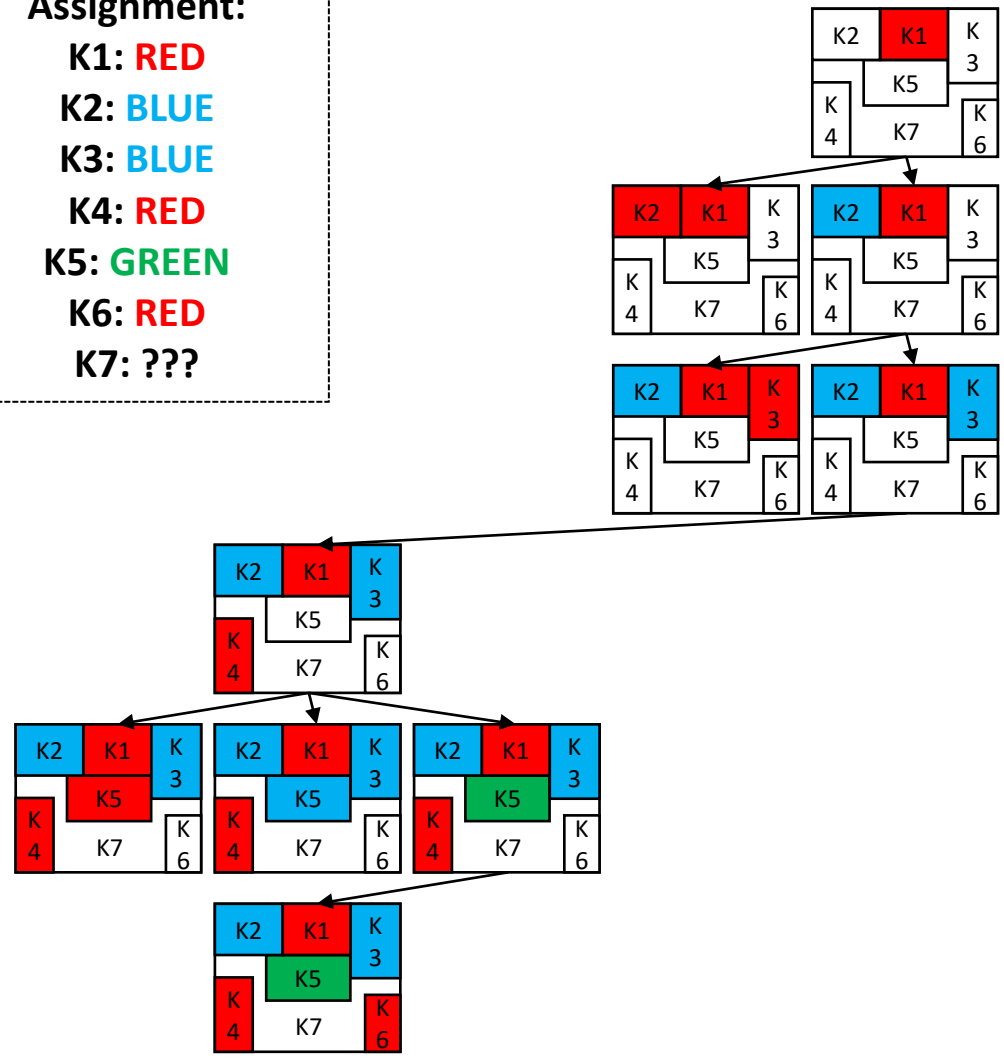
K3: BLUE

K4: RED

K5: GREEN

K6: RED

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

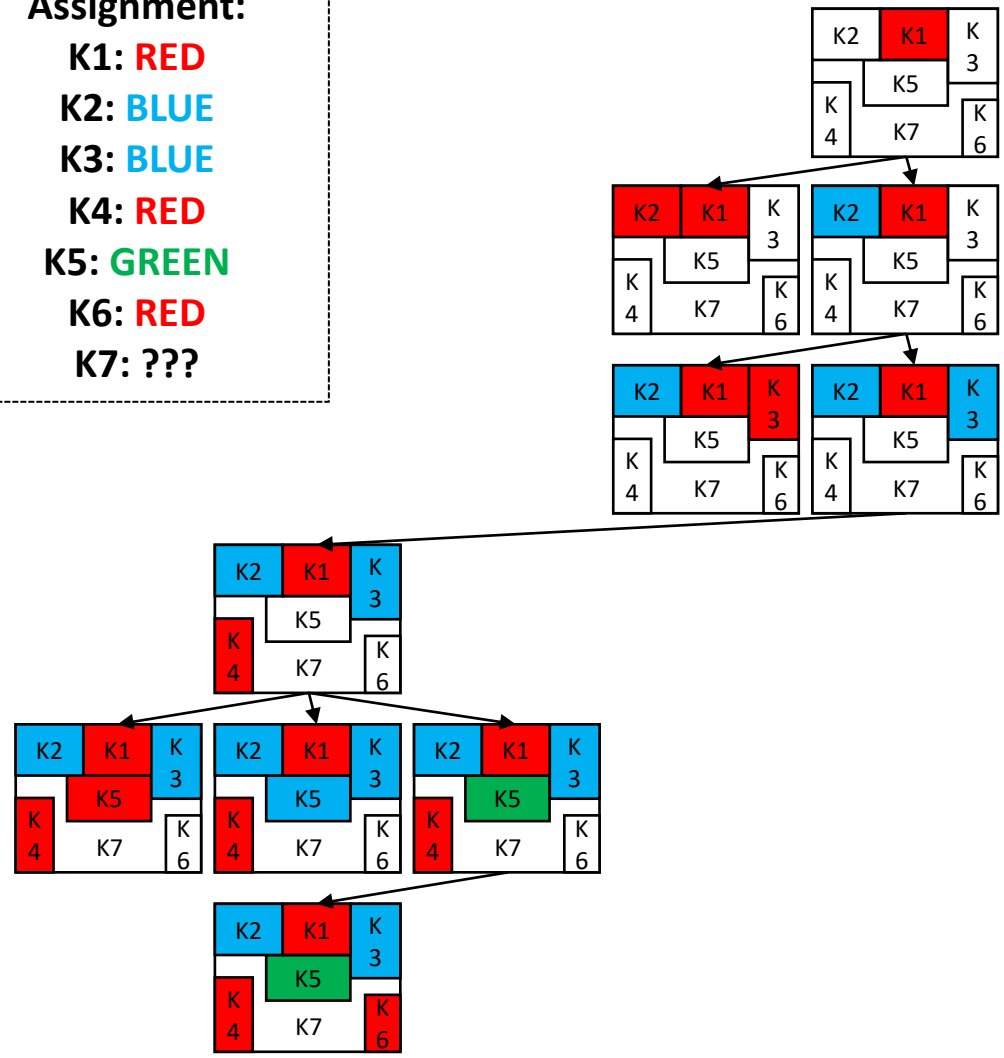
K3: BLUE

K4: RED

K5: GREEN

K6: RED

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

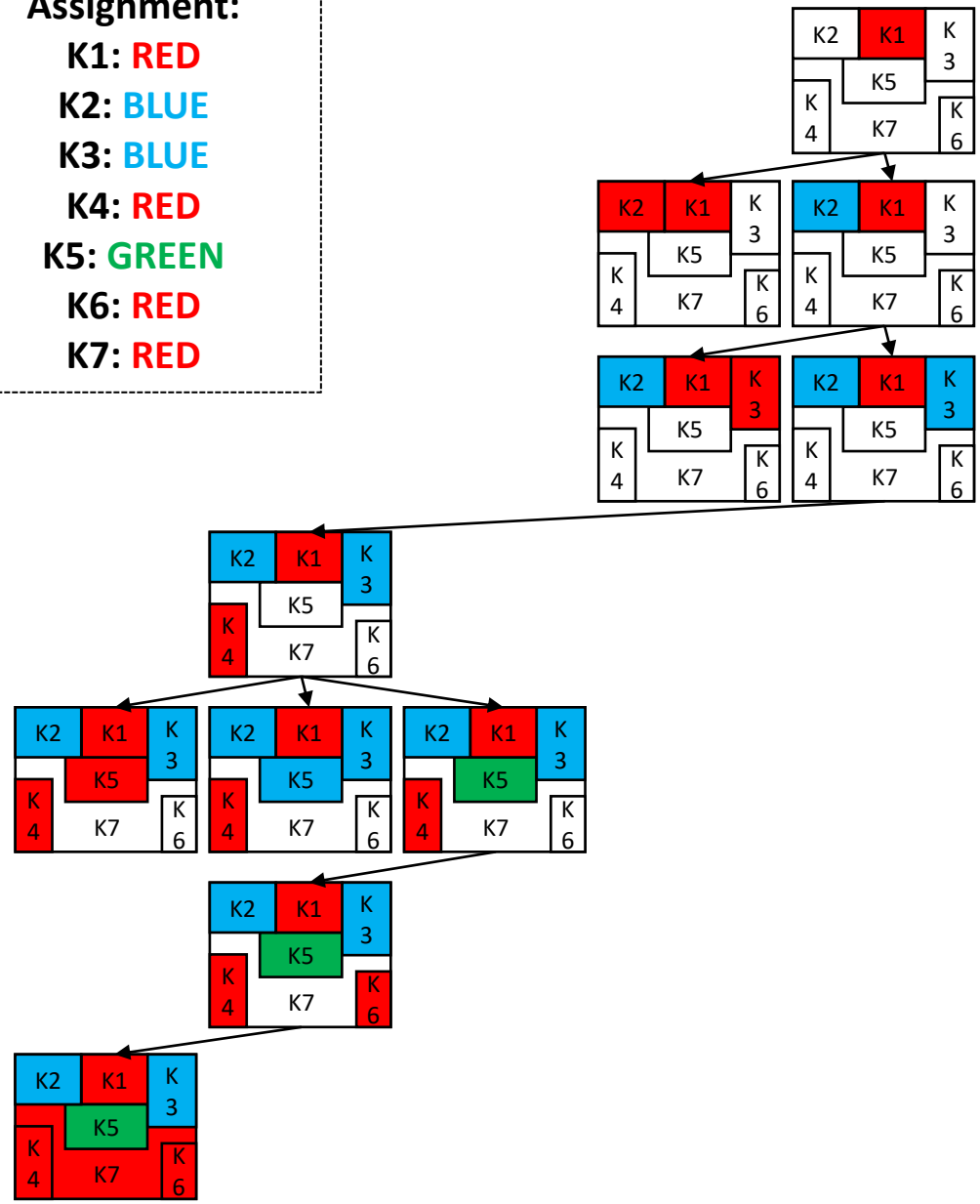
K3: BLUE

K4: RED

K5: GREEN

K6: RED

K7: RED



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

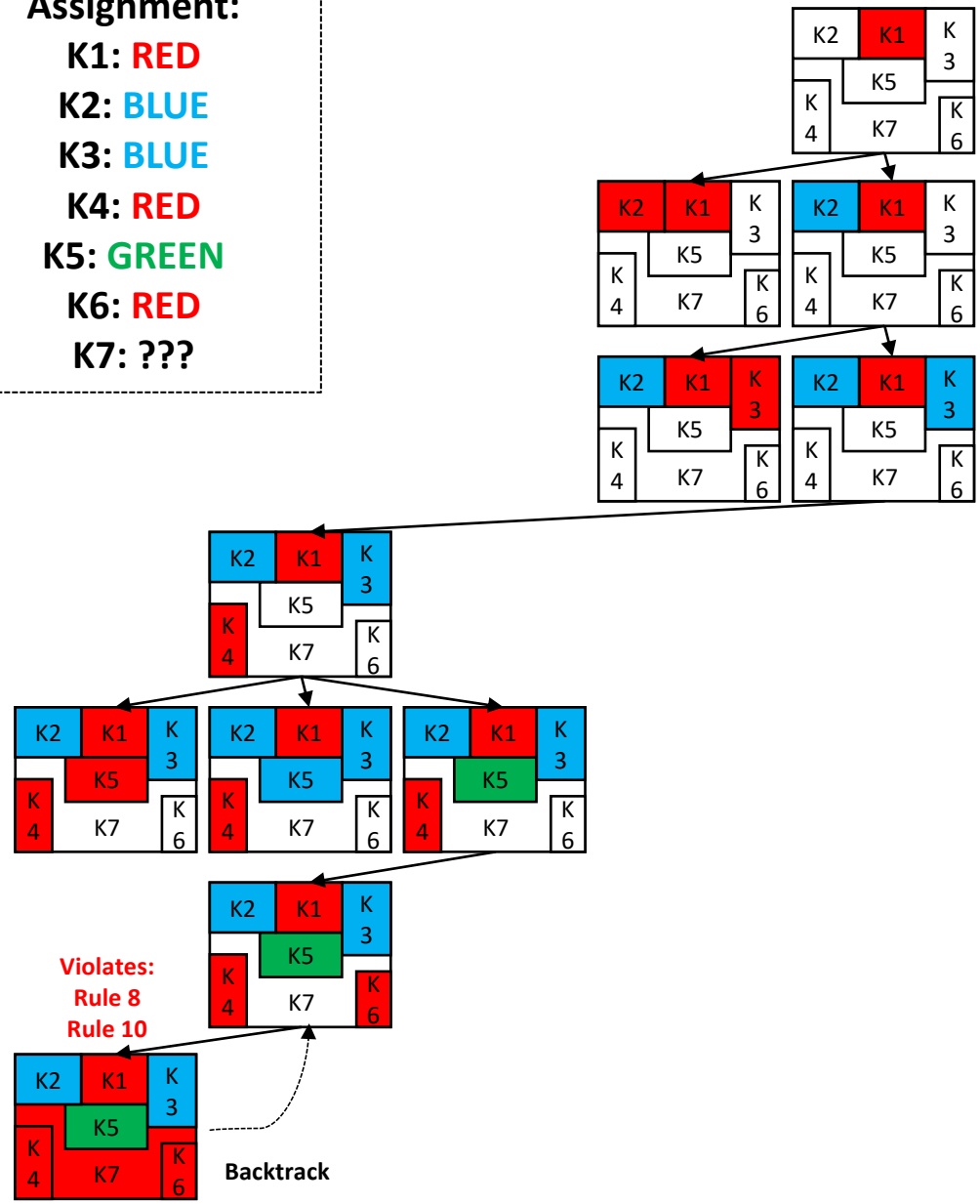
K3: BLUE

K4: RED

K5: GREEN

K6: RED

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Assignment:

K1: RED

K2: BLUE

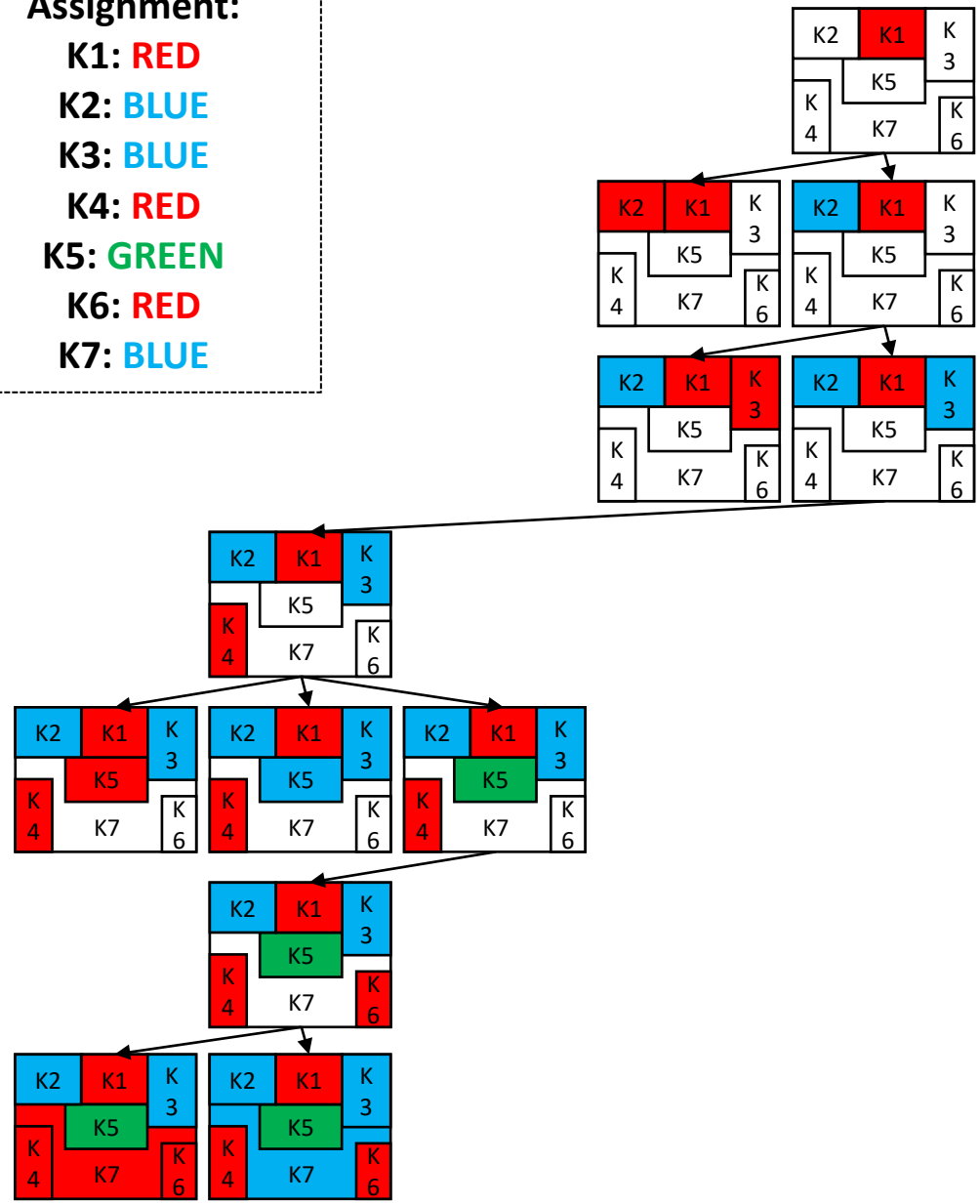
K3: BLUE

K4: RED

K5: GREEN

K6: RED

K7: BLUE



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

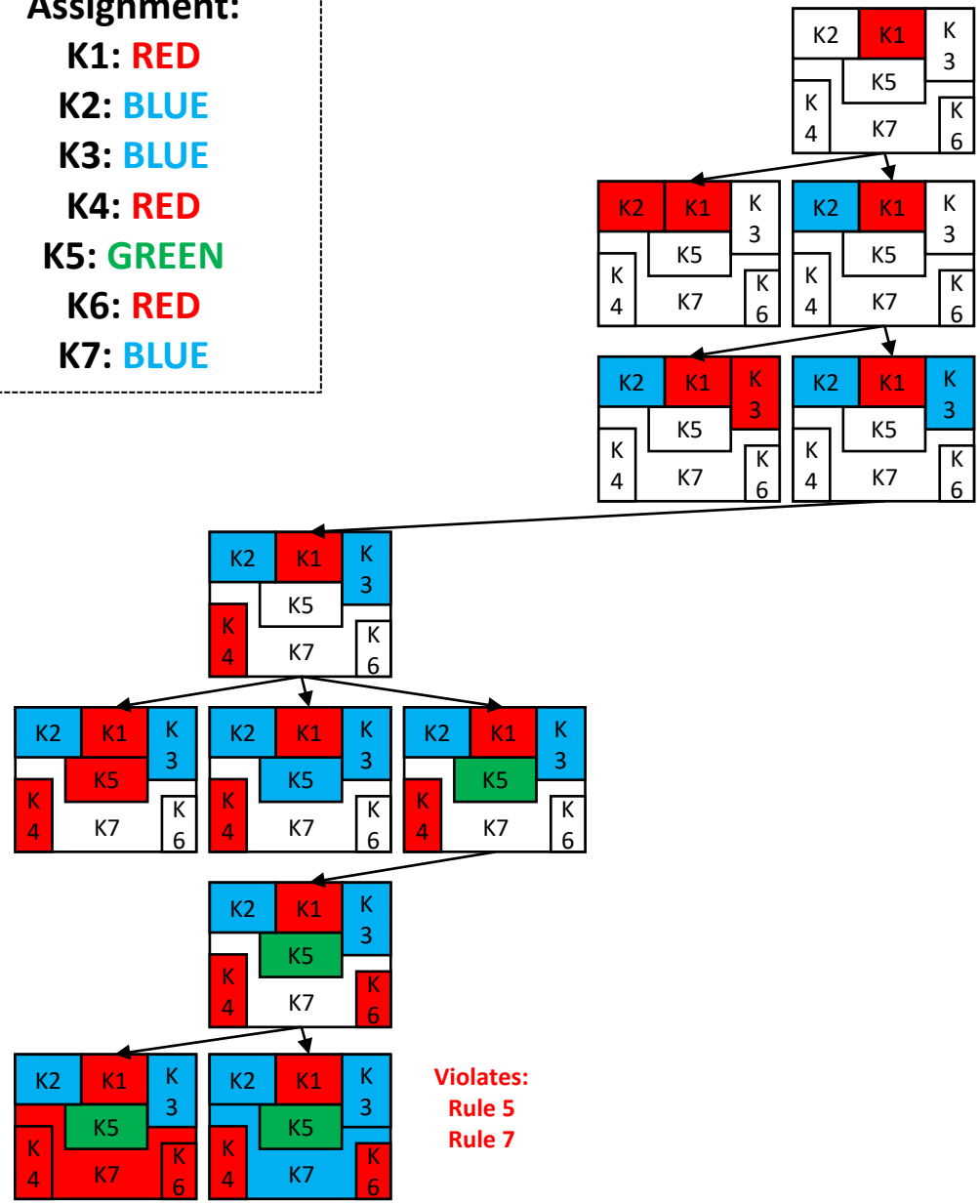
Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: RED
K7: BLUE

Constraints:

Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

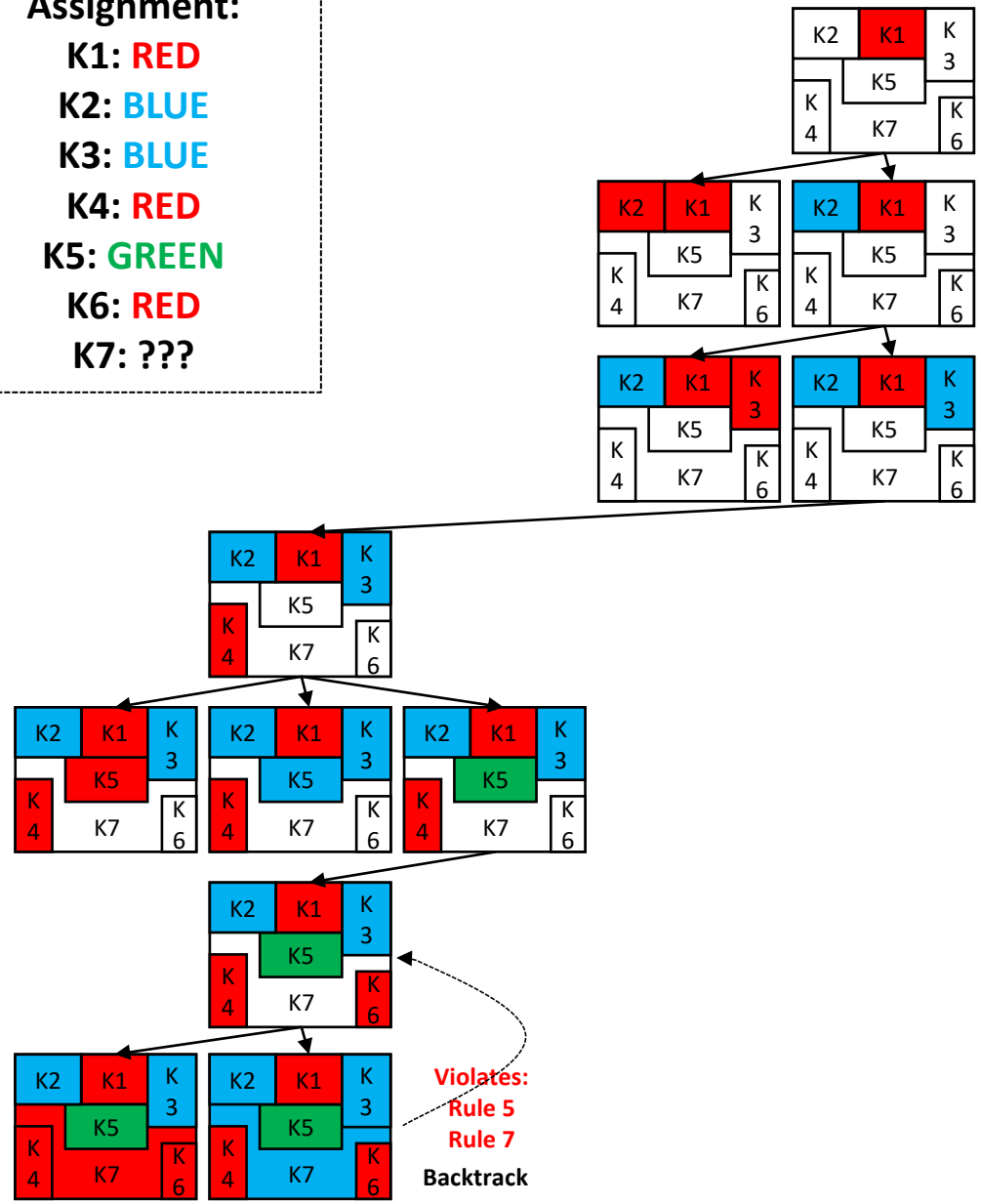
K3: BLUE

K4: RED

K5: GREEN

K6: RED

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Assignment:

K1: RED

K2: BLUE

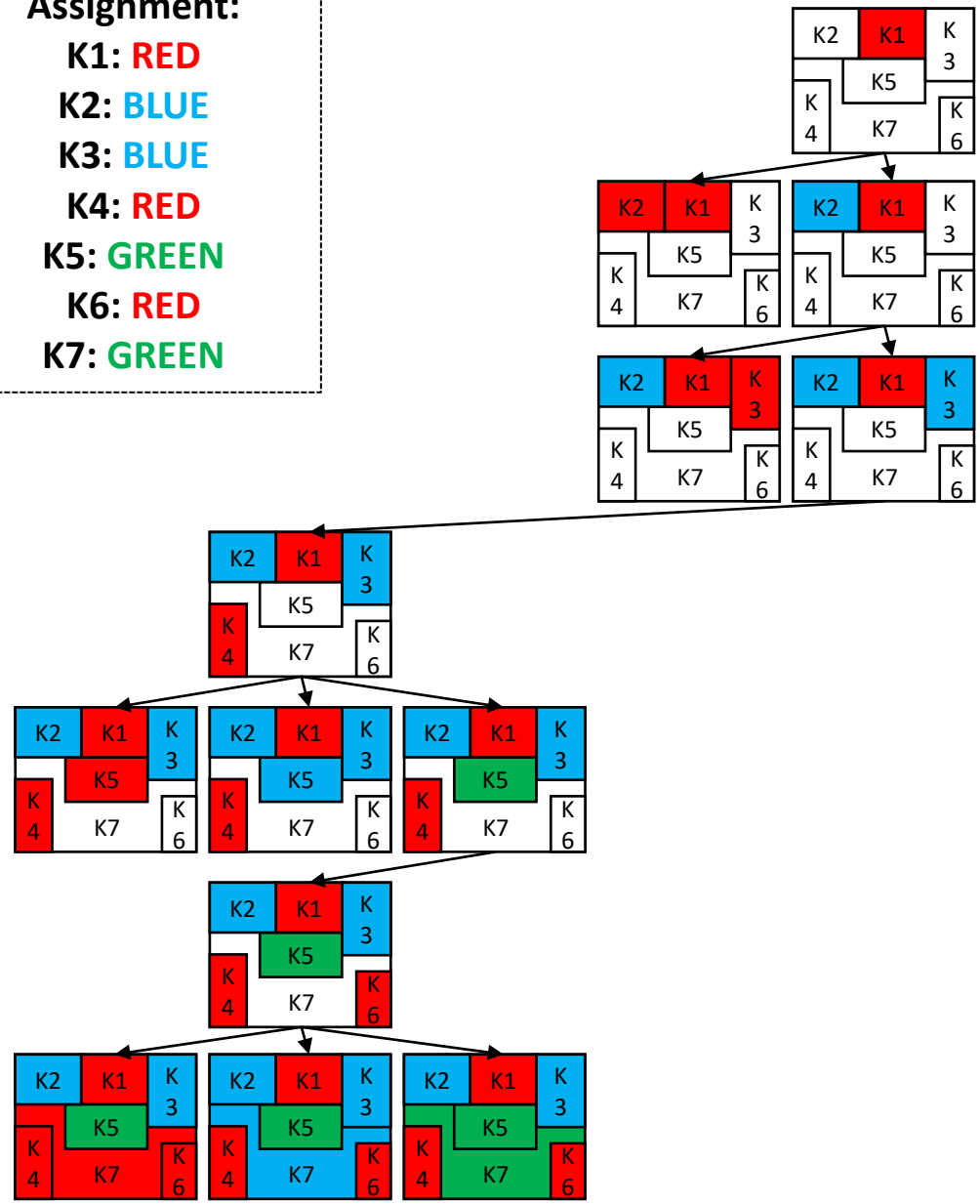
K3: BLUE

K4: RED

K5: GREEN

K6: RED

K7: GREEN



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

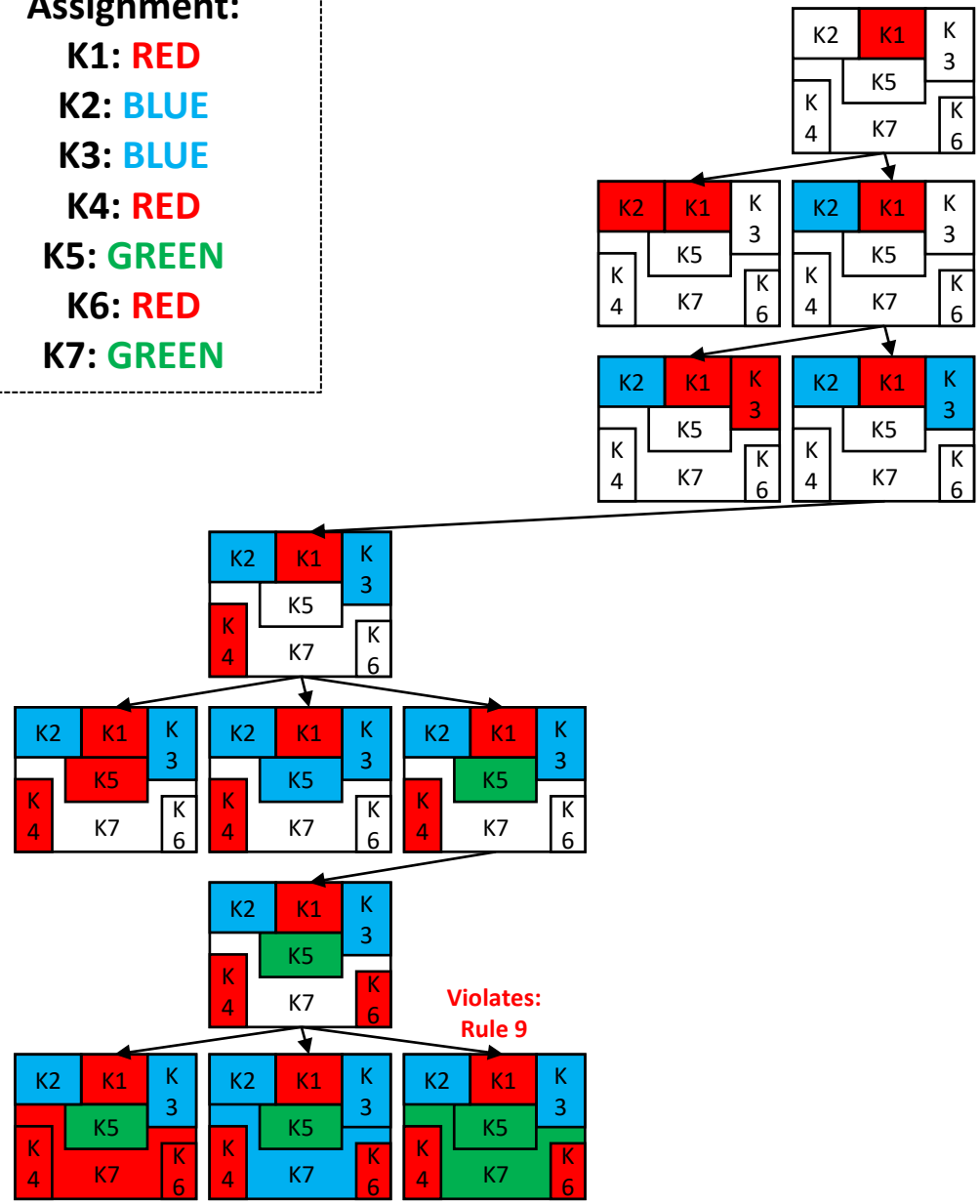
Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: RED
K7: GREEN

Constraints:

Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

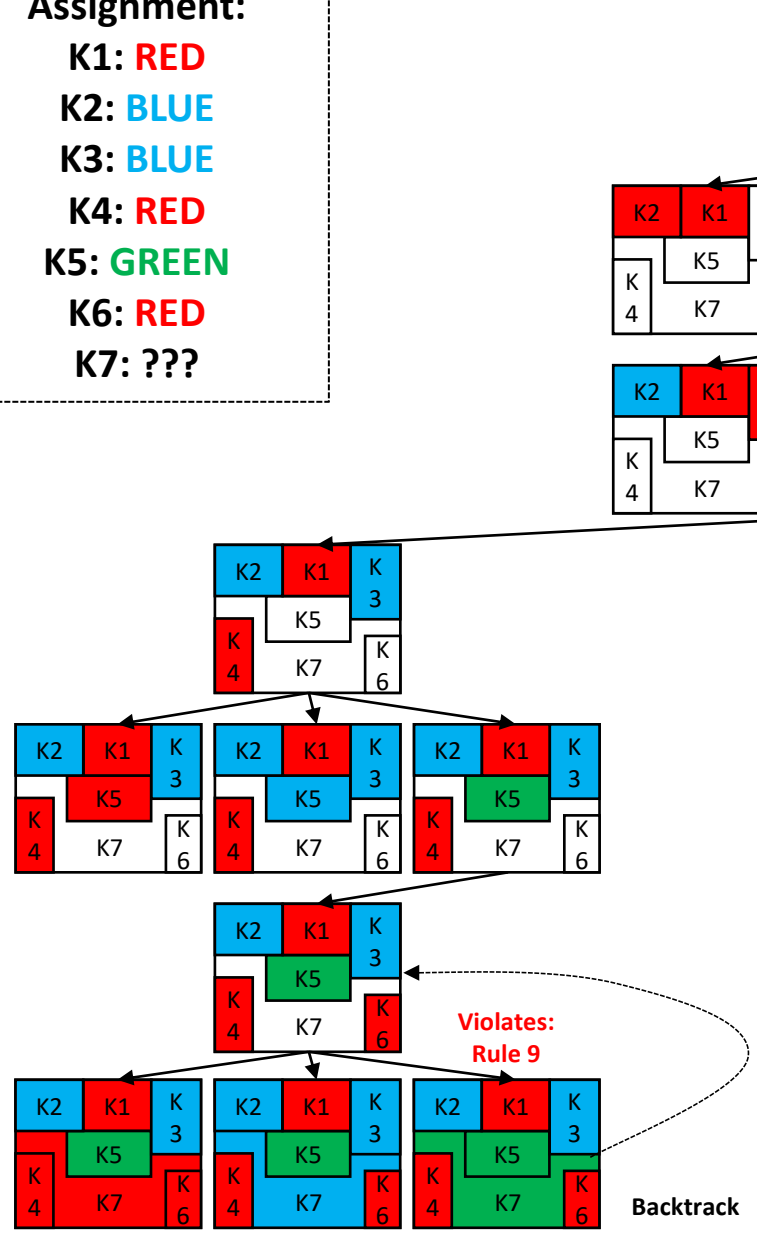
K3: BLUE

K4: RED

K5: GREEN

K6: RED

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

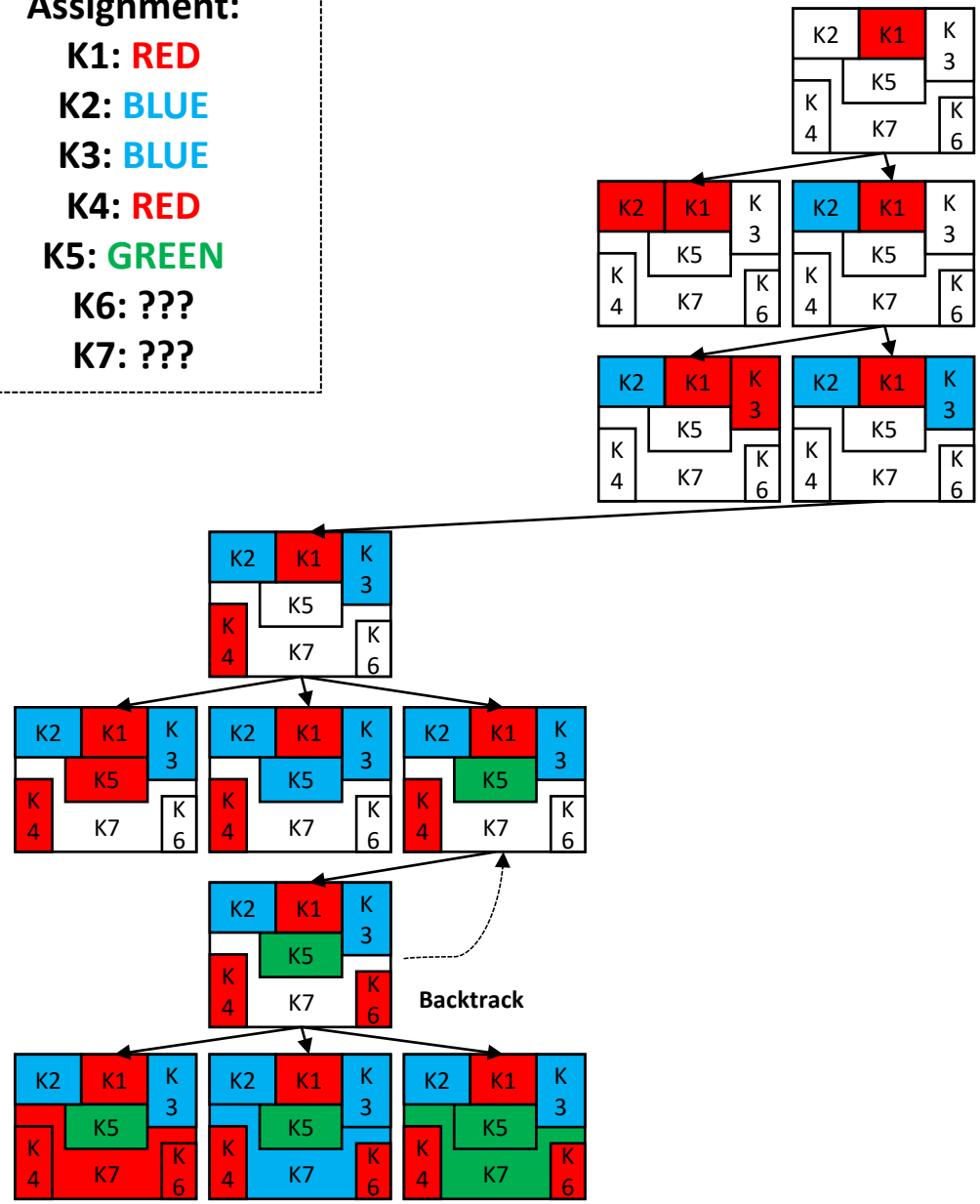
Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: ???
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

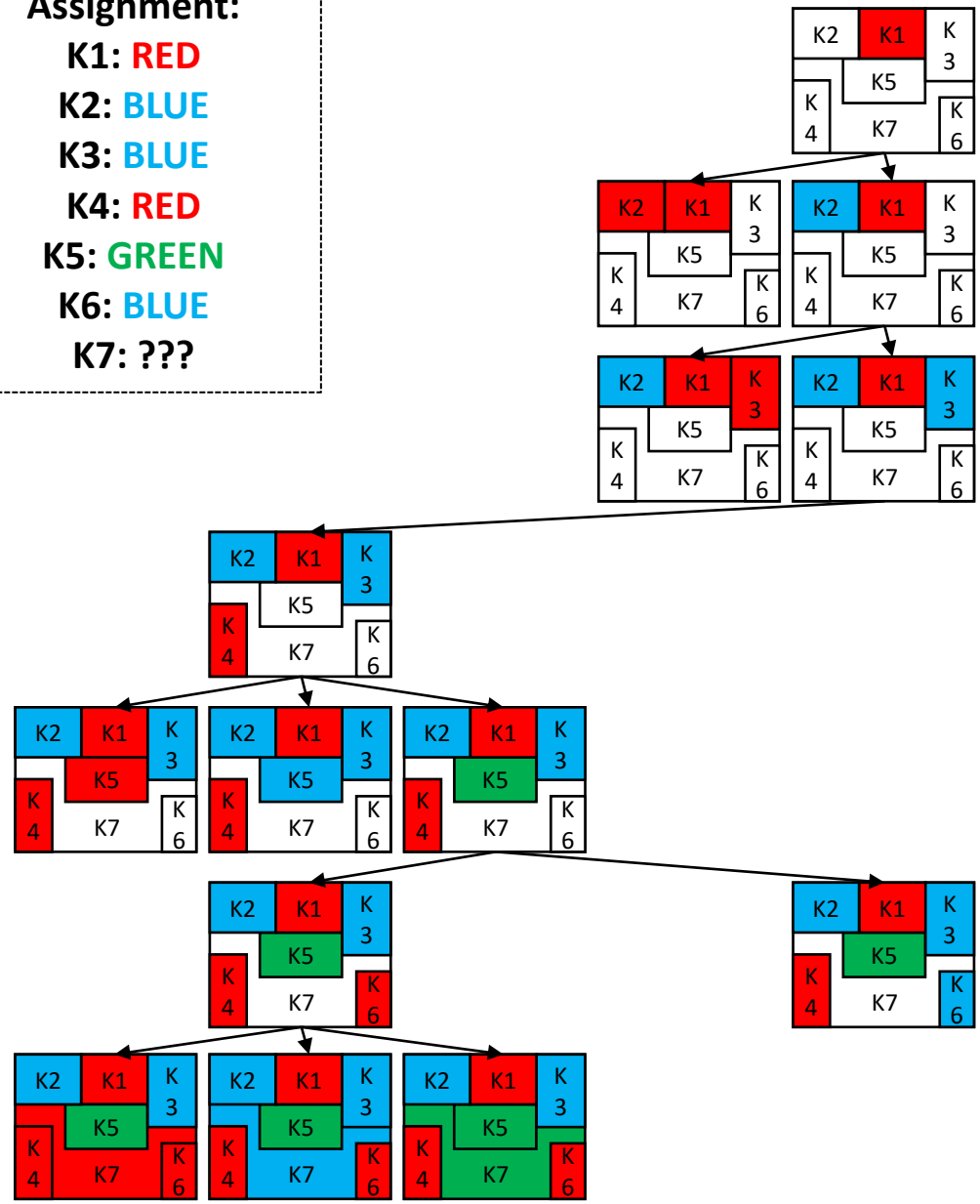
K3: BLUE

K4: RED

K5: GREEN

K6: BLUE

K7: ???



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

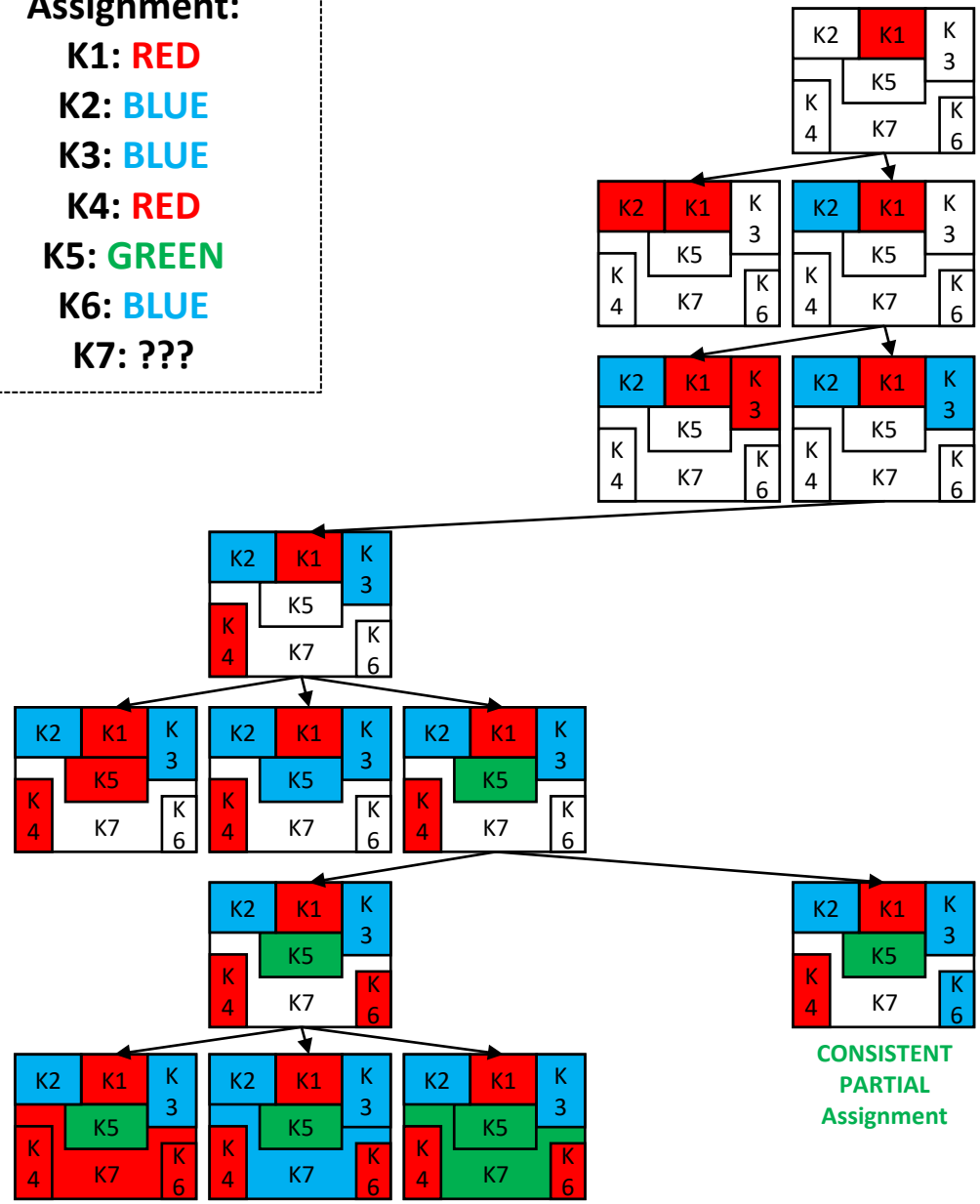
Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: BLUE
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

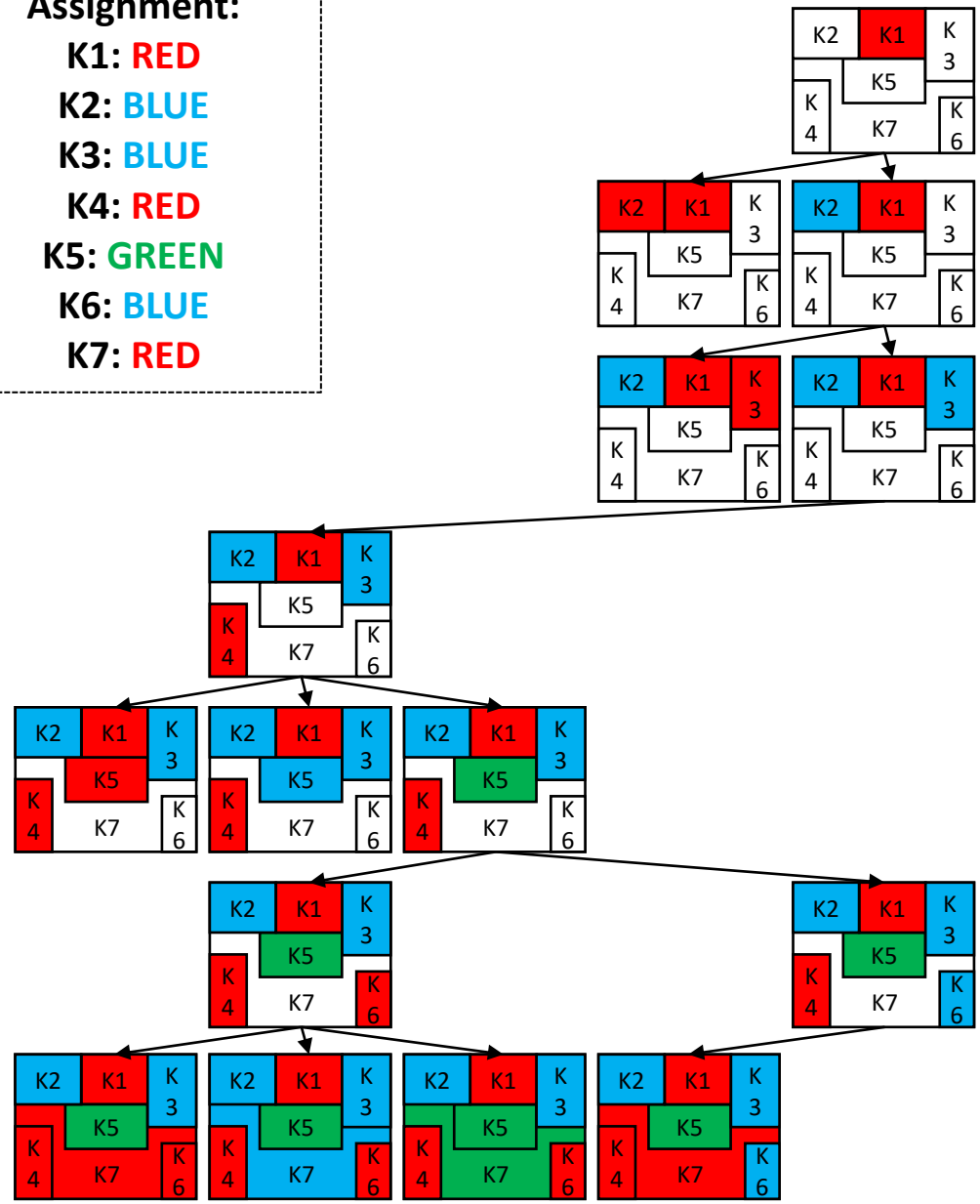
K3: BLUE

K4: RED

K5: GREEN

K6: BLUE

K7: RED



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

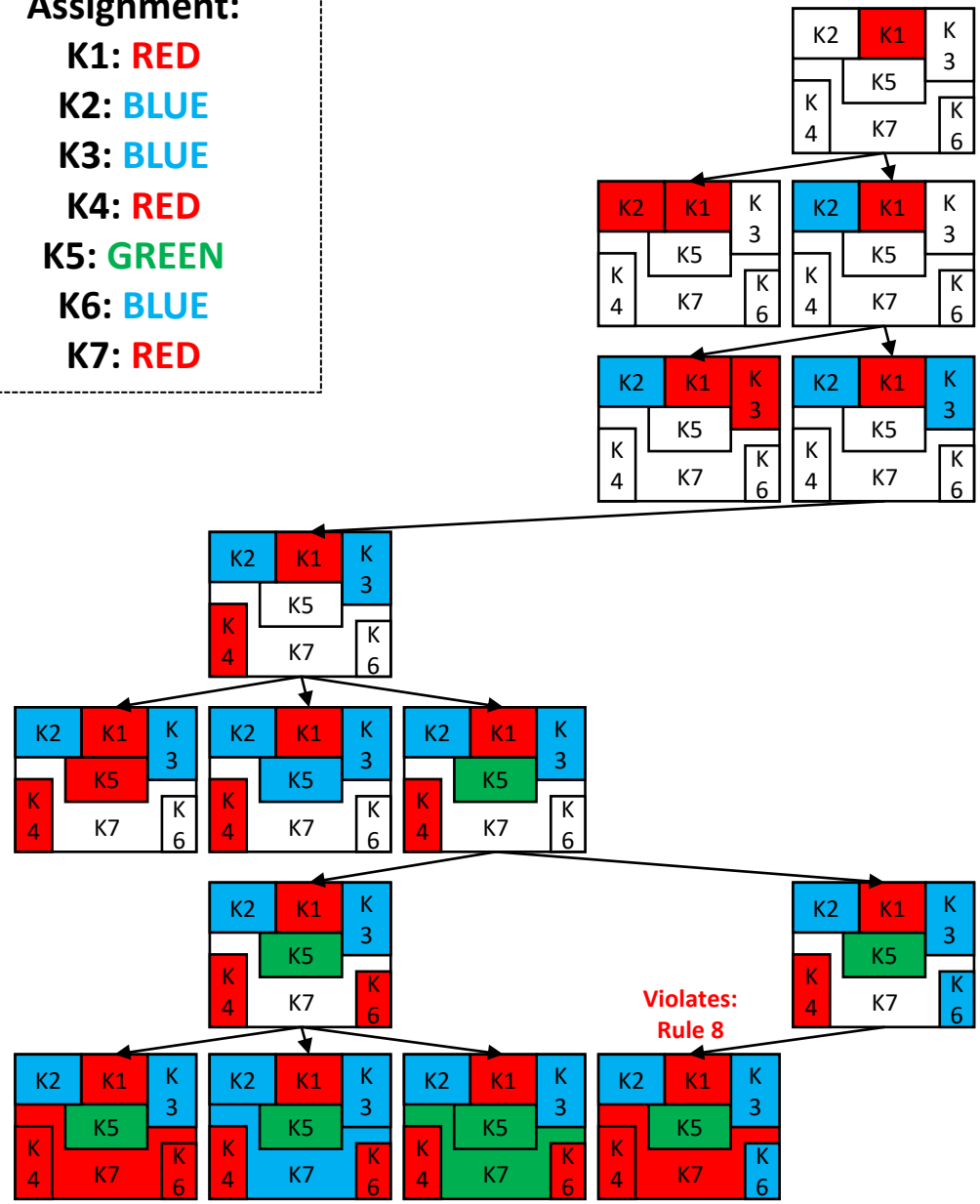
K3: BLUE

K4: RED

K5: GREEN

K6: BLUE

K7: RED



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

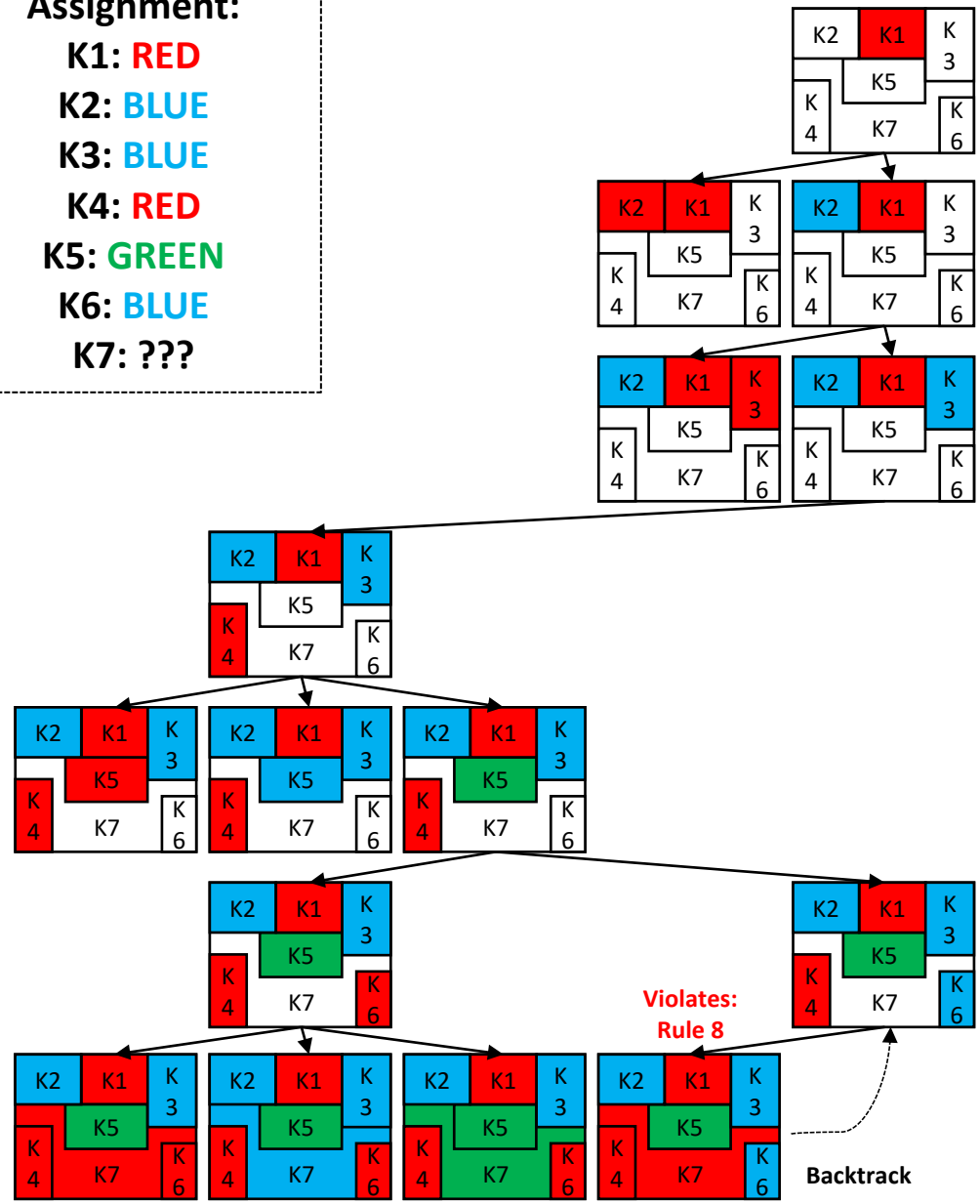
Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: BLUE
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Assignment:

K1: RED

K2: BLUE

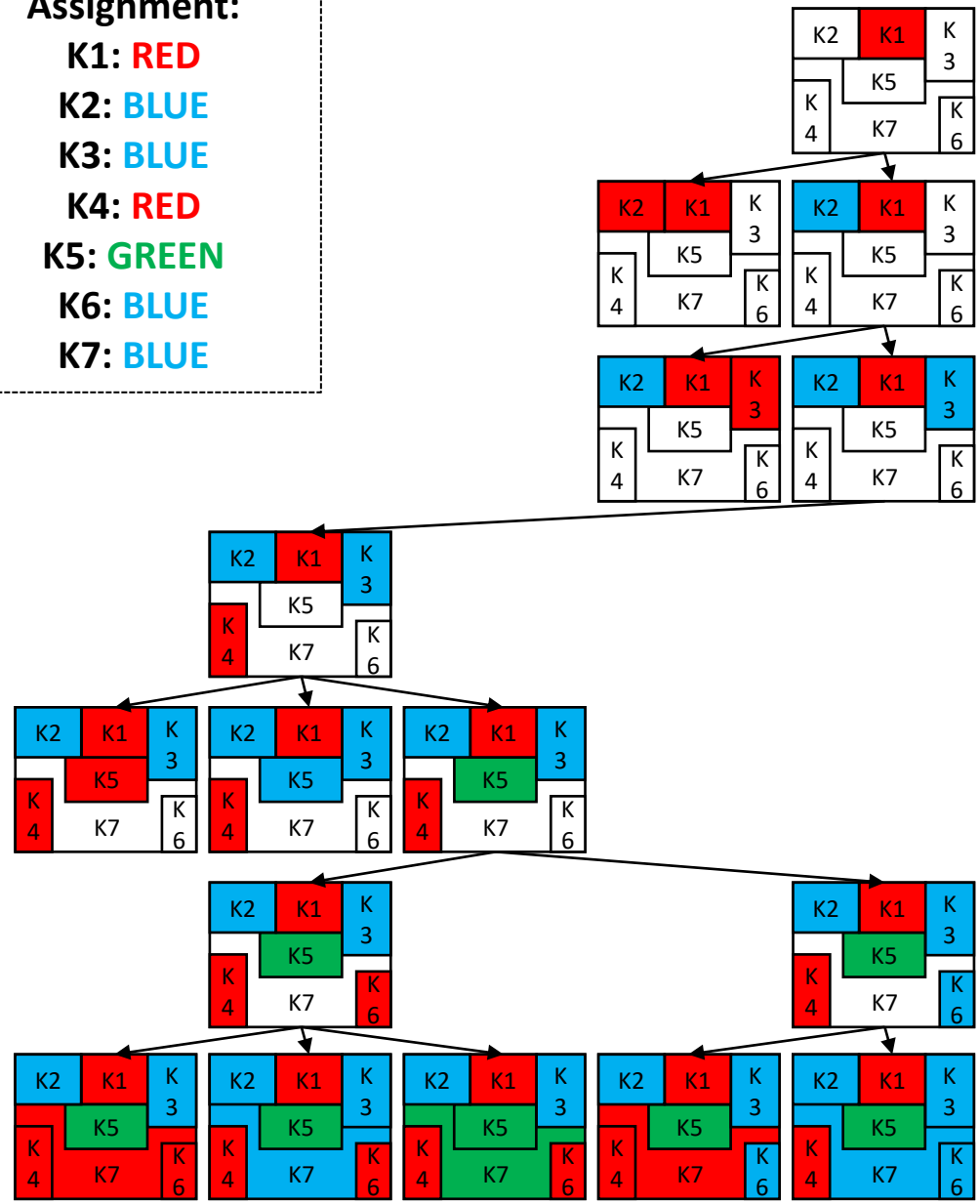
K3: BLUE

K4: RED

K5: GREEN

K6: BLUE

K7: BLUE



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

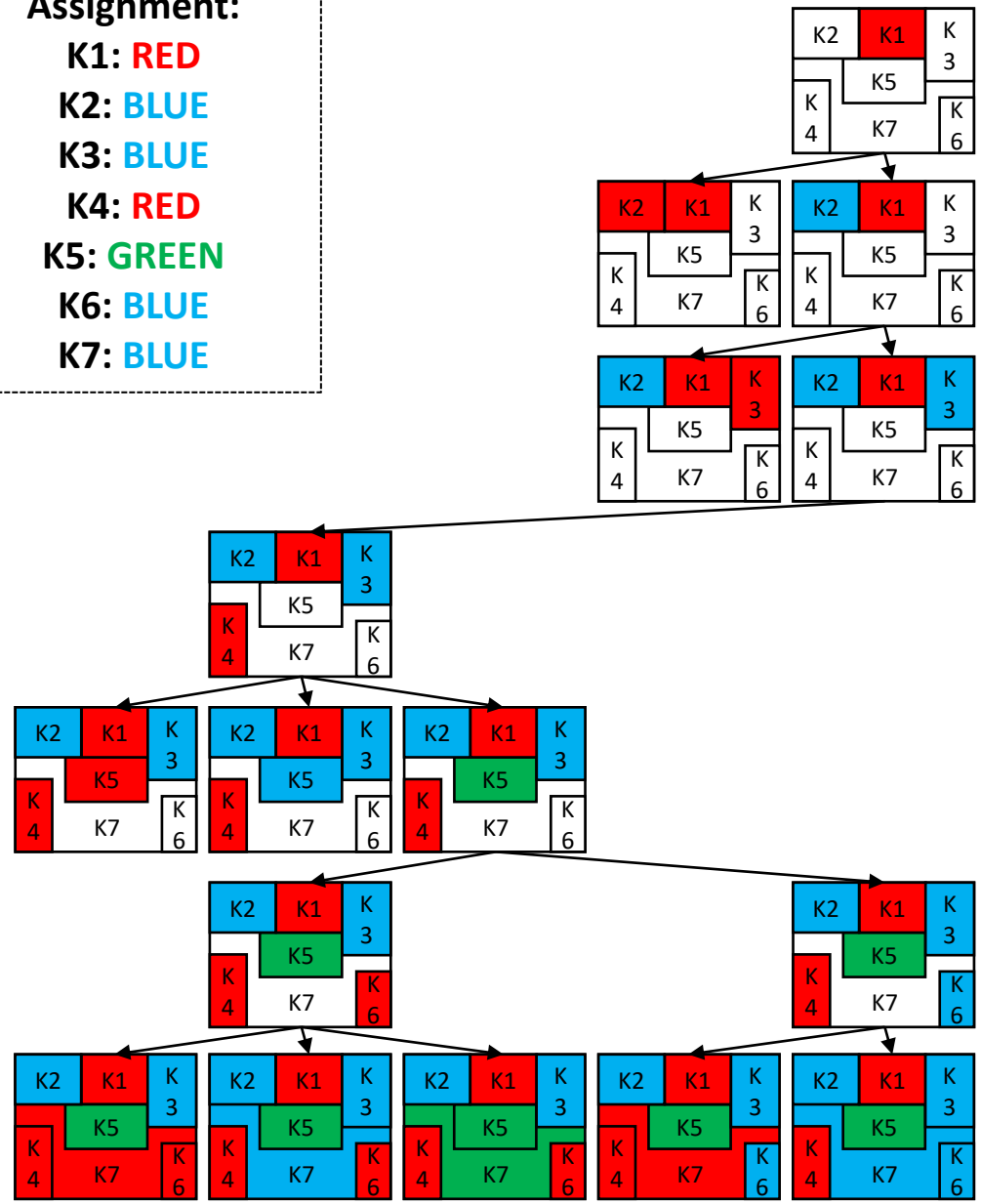
K3: BLUE

K4: RED

K5: GREEN

K6: BLUE

K7: BLUE



Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

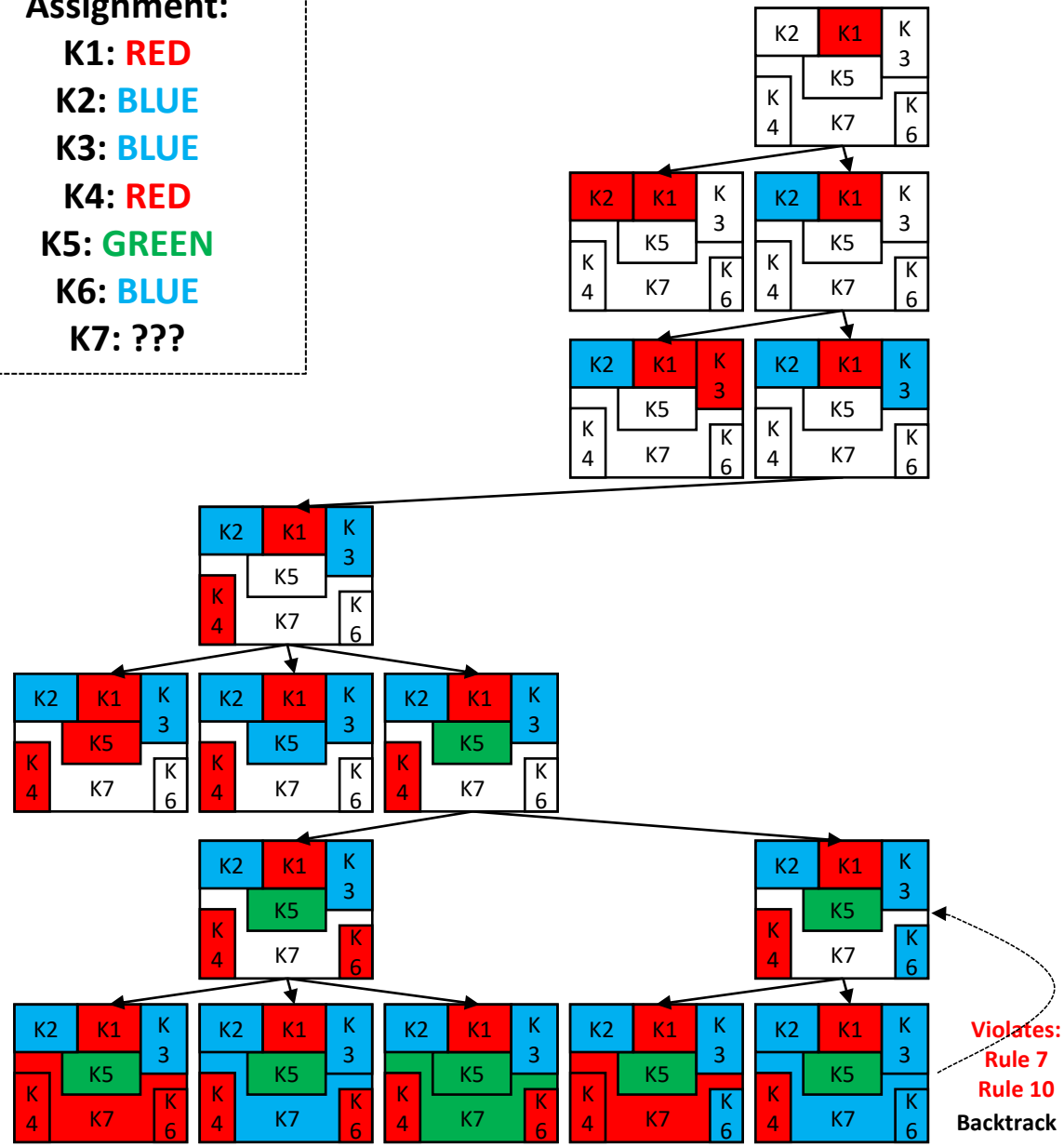
Rule 10: K6 ≠ K7

Violates:
Rule 7
Rule 10

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: BLUE
K7: ???

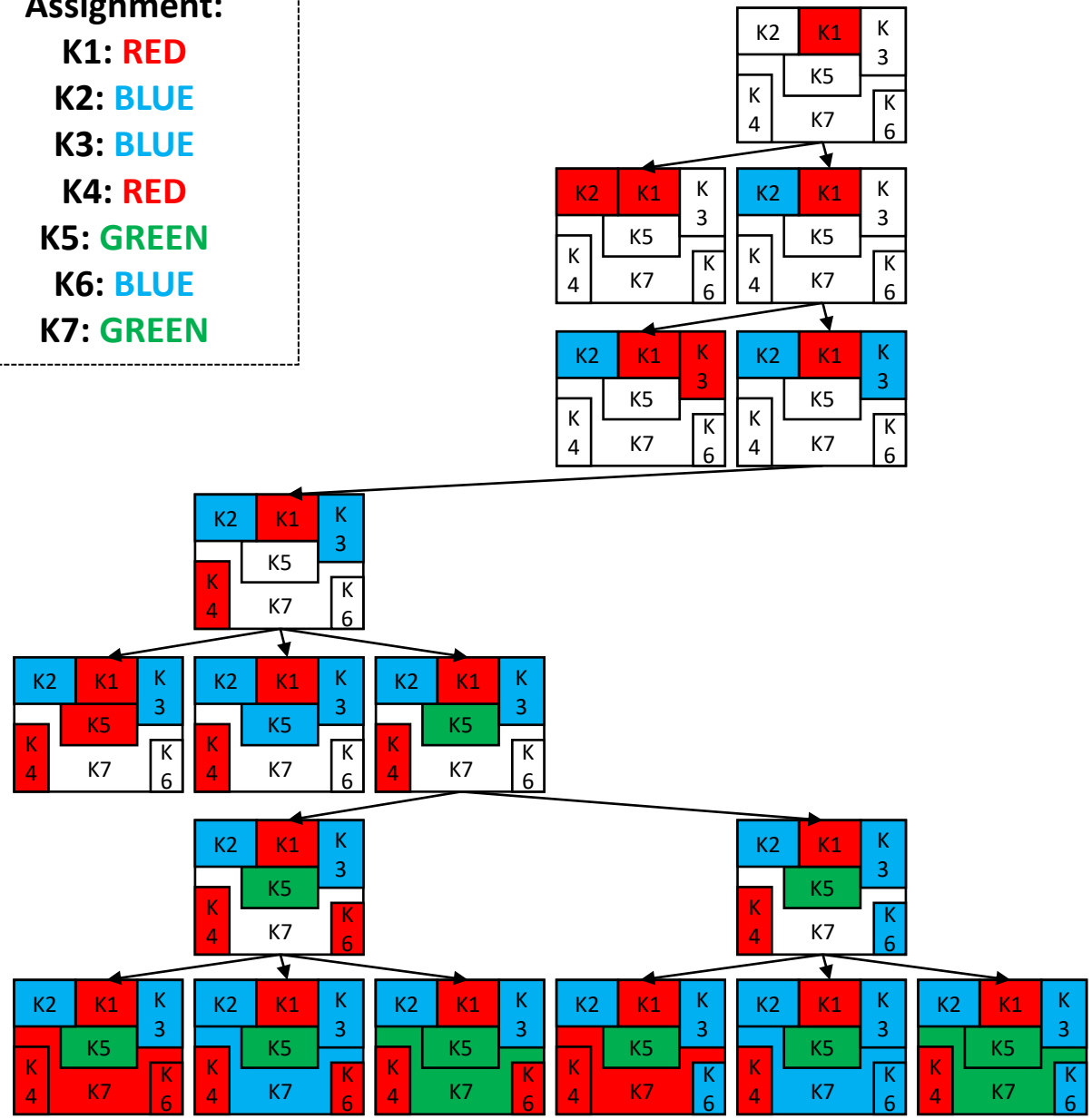
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: BLUE
K7: GREEN

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: RED

K5: GREEN

K6: BLUE

K7: GREEN

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

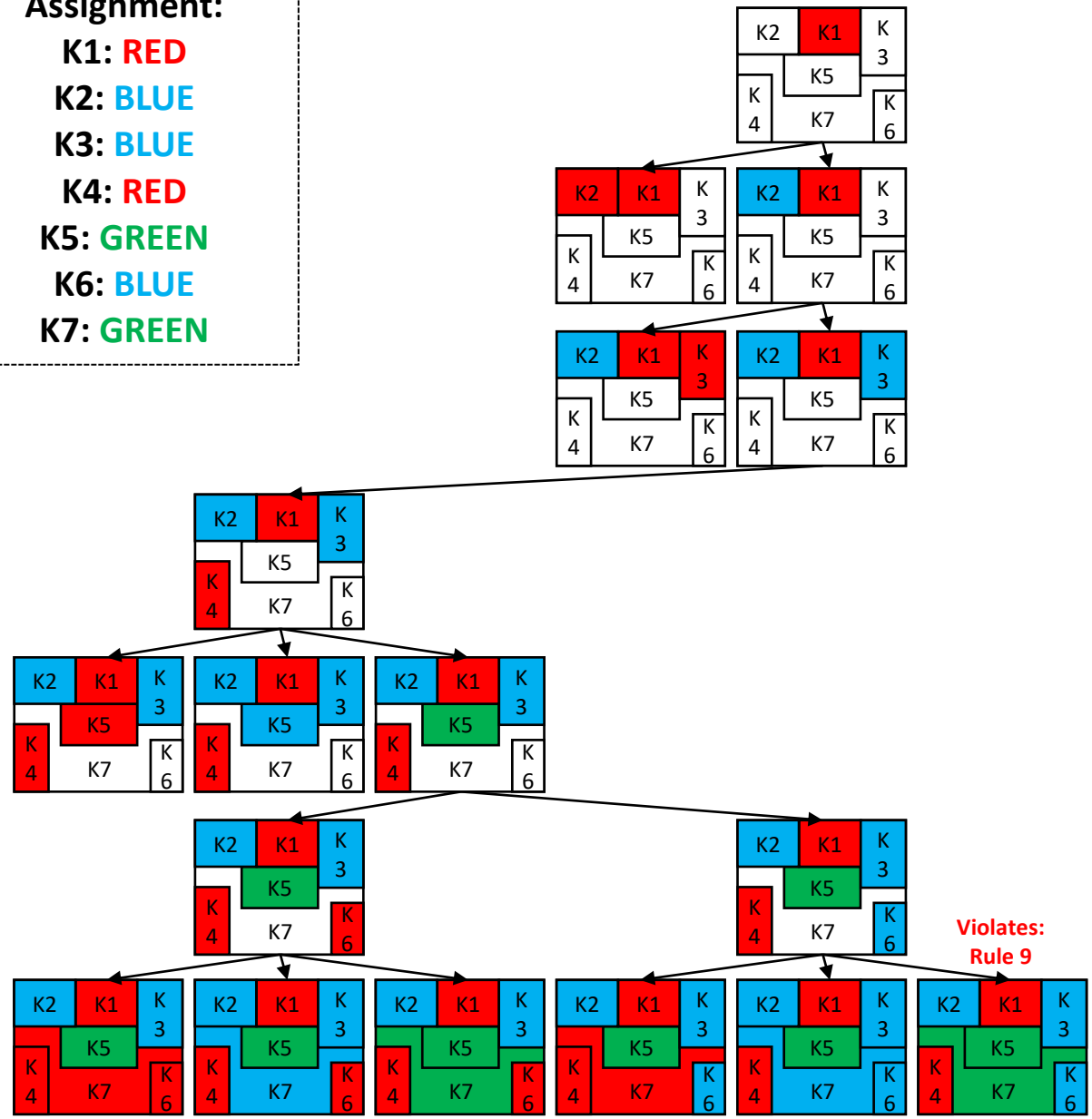
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: RED

K5: GREEN

K6: BLUE

K7: ???

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

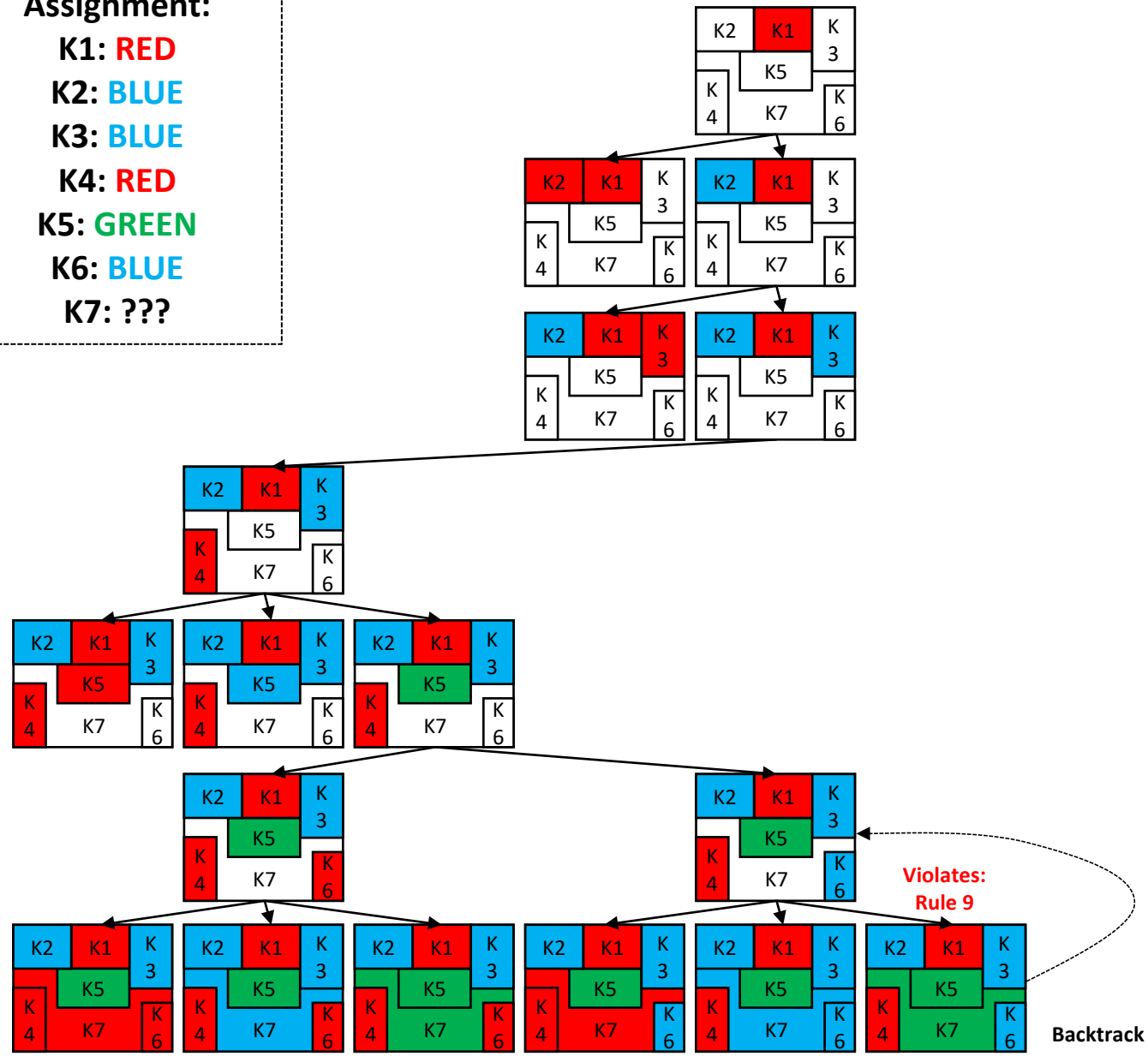
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

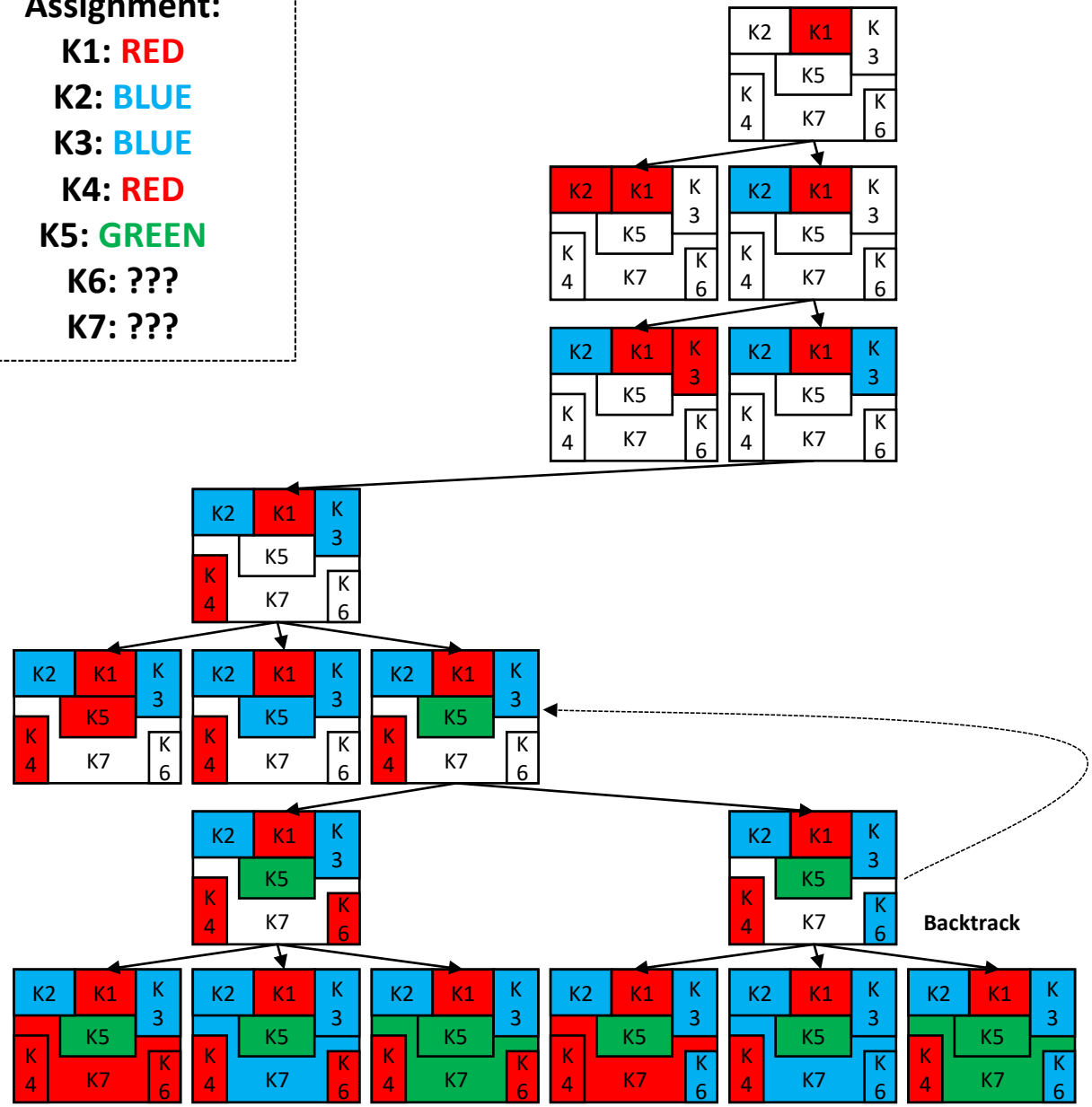
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: ???
K7: ???

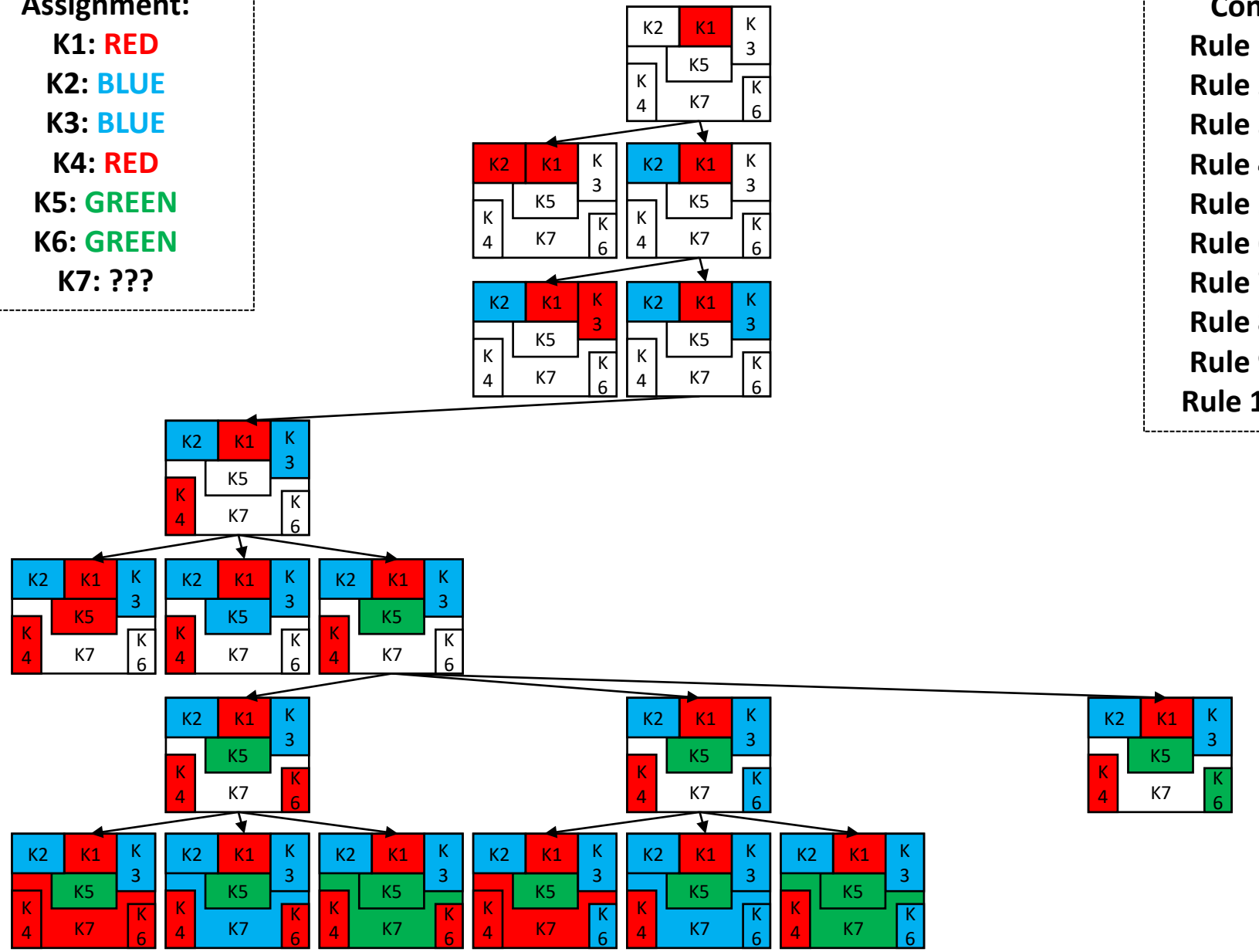
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: GREEN
K7: ???

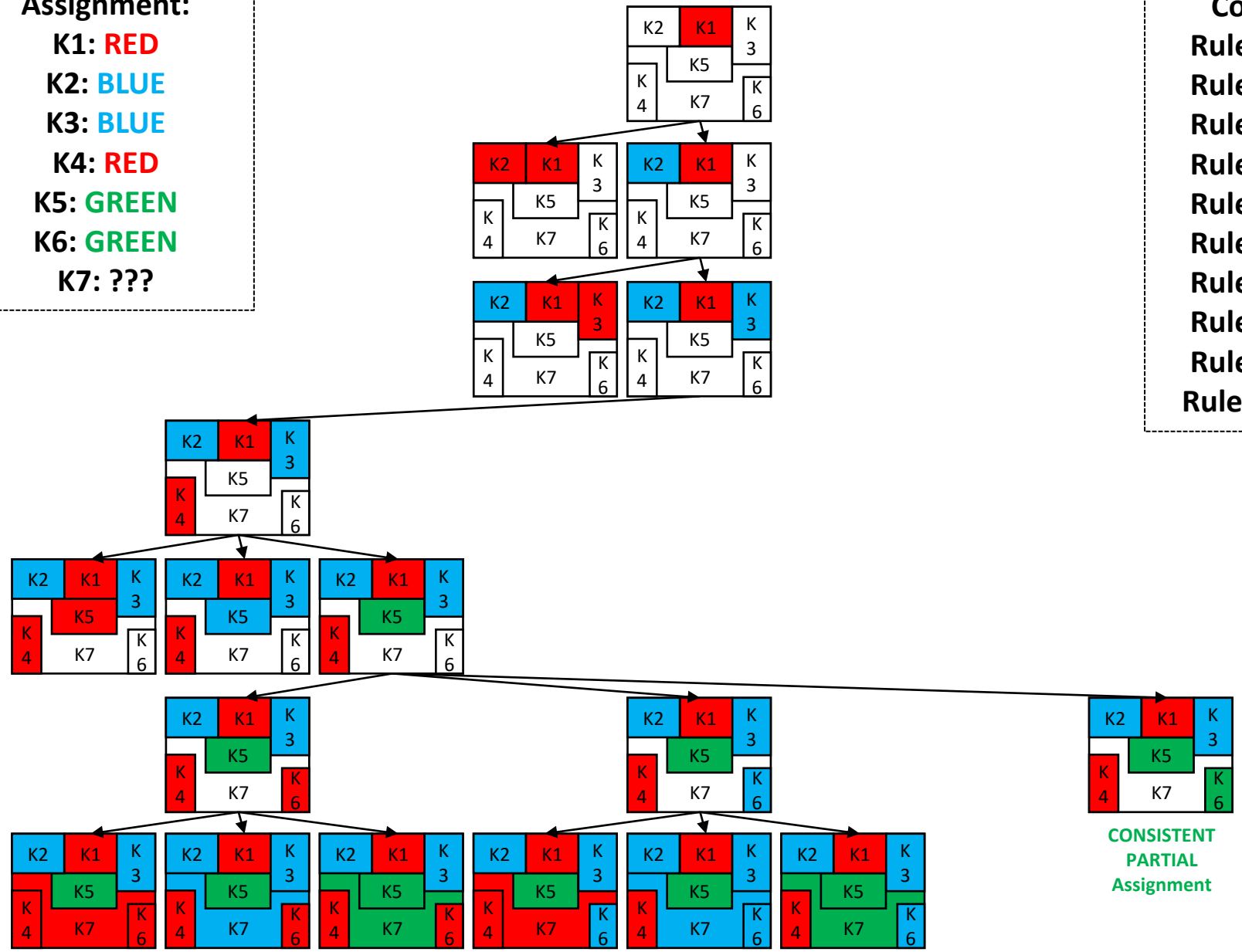
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: GREEN
K7: ???

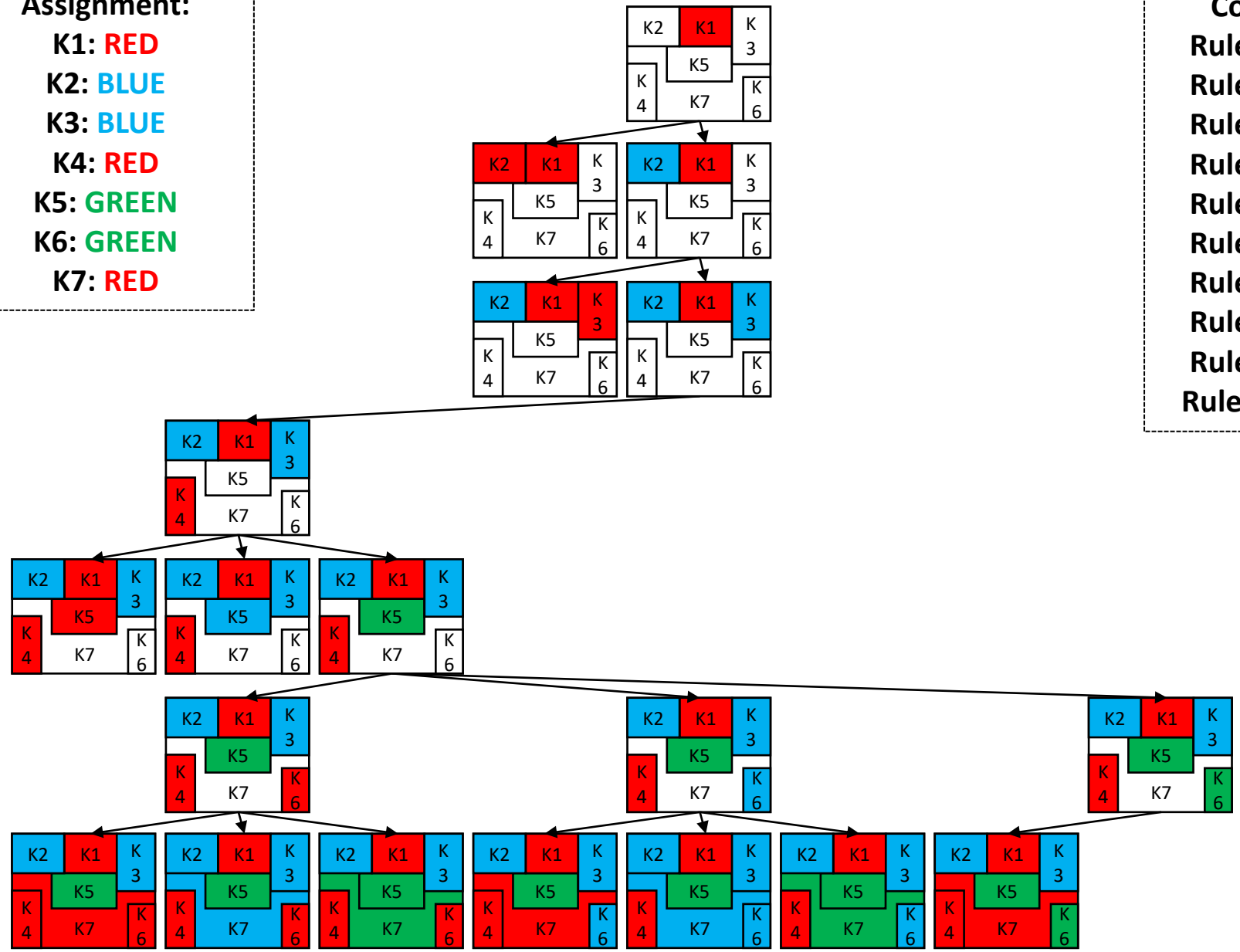
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: GREEN
K7: RED

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: RED

K5: GREEN

K6: GREEN

K7: RED

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

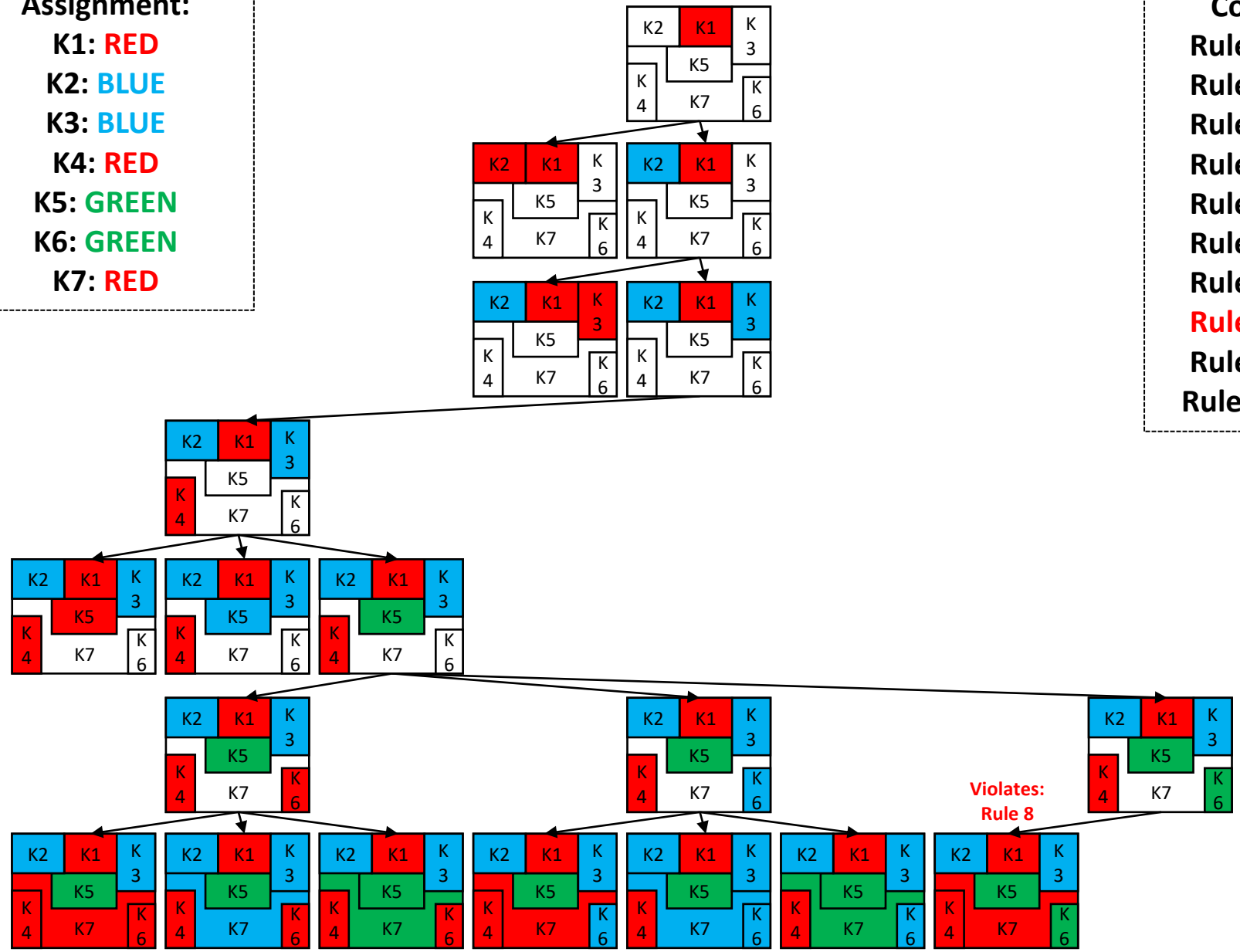
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

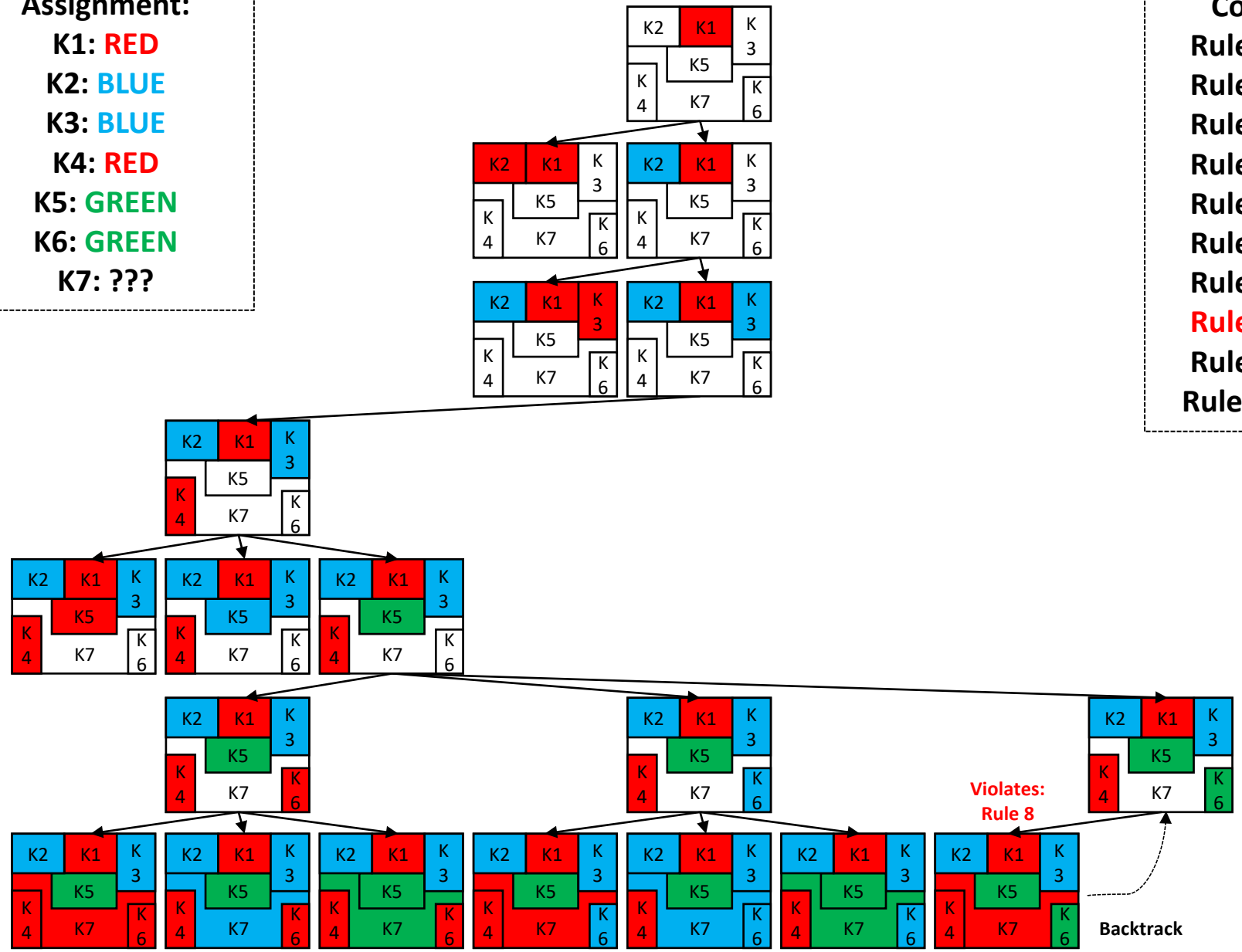
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: GREEN
K7: ???

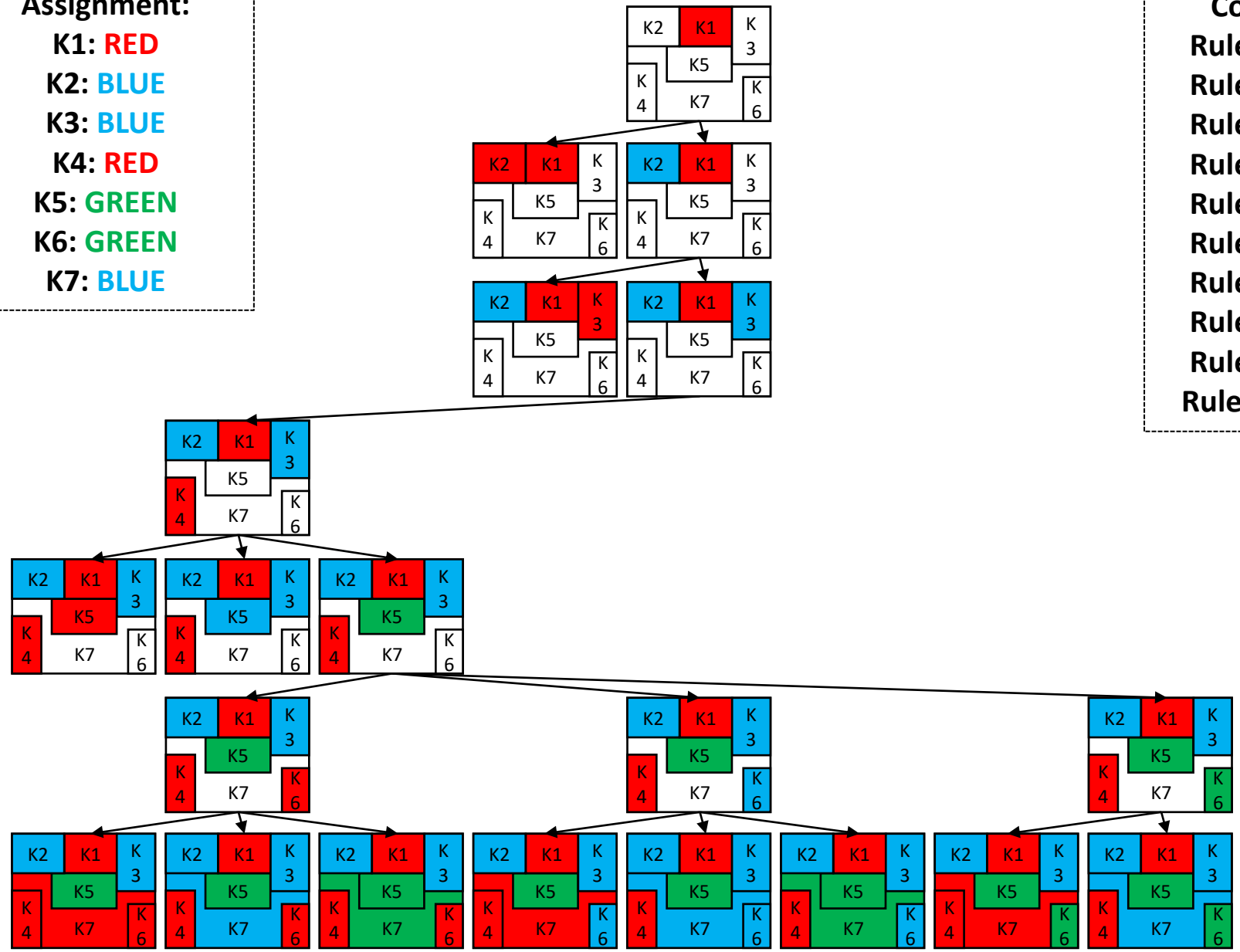
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: GREEN
K7: BLUE

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: RED

K5: GREEN

K6: GREEN

K7: BLUE

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

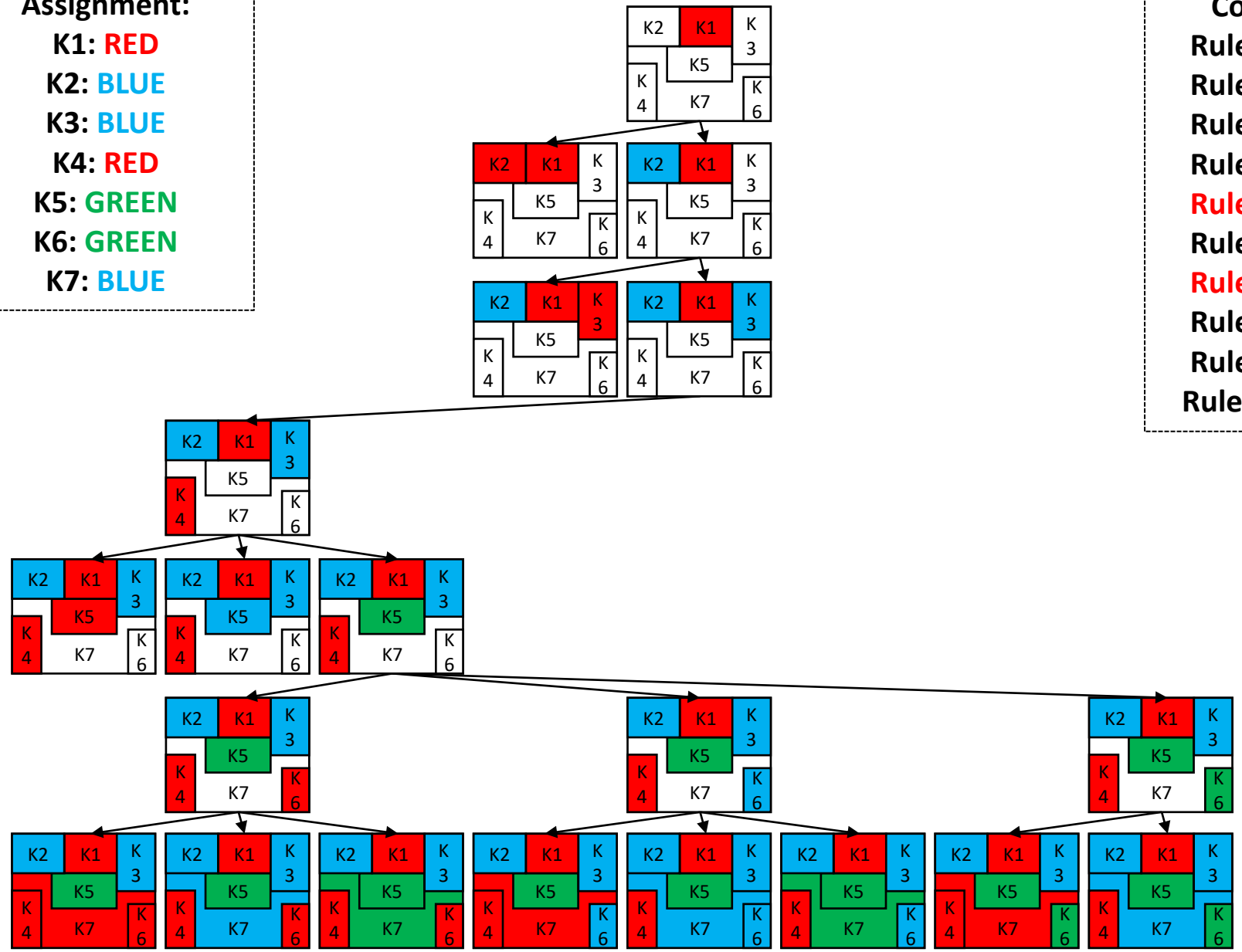
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7

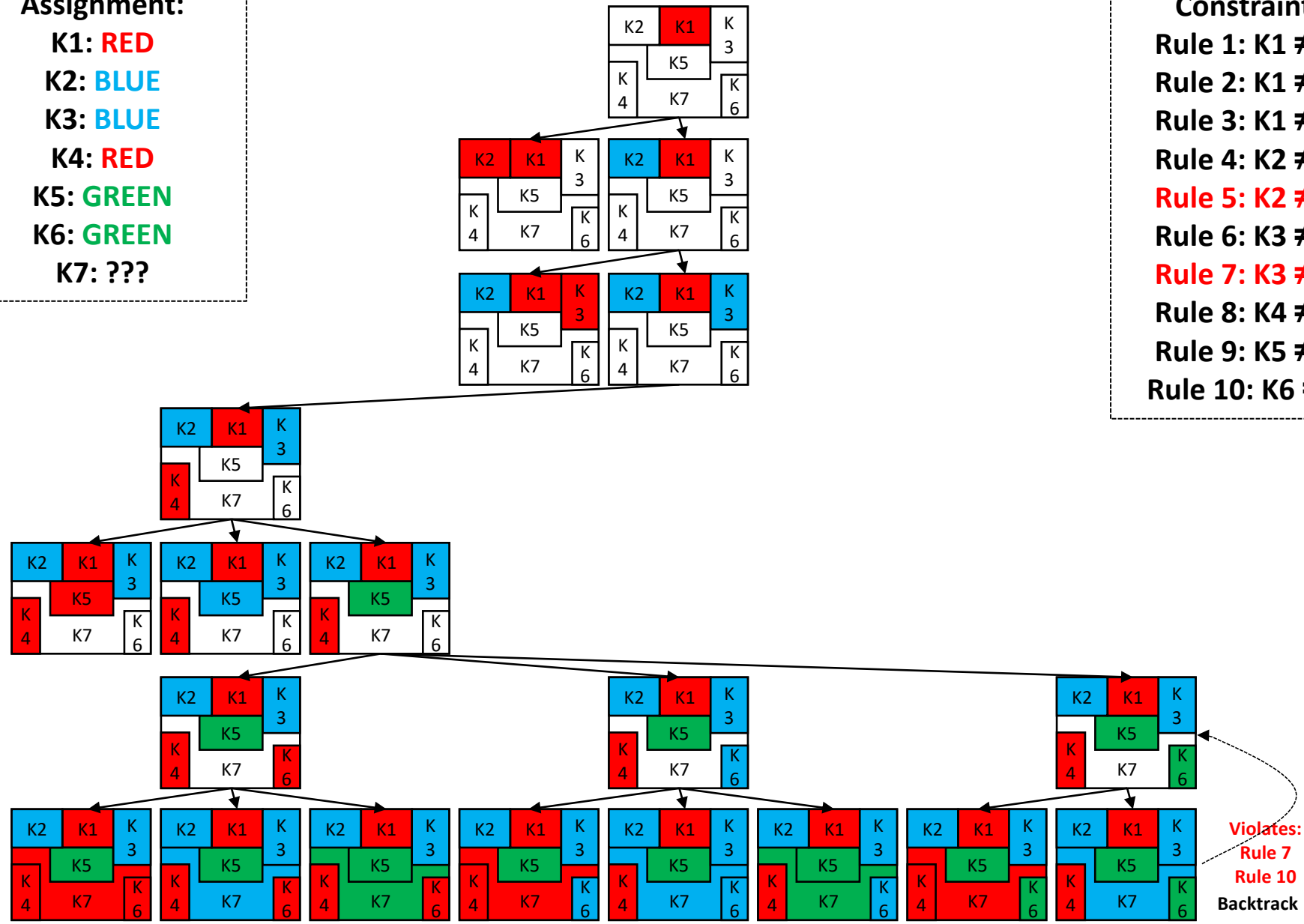


Violates:
Rule 7
Rule 10

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: GREEN
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: RED

K5: GREEN

K6: GREEN

K7: GREEN

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

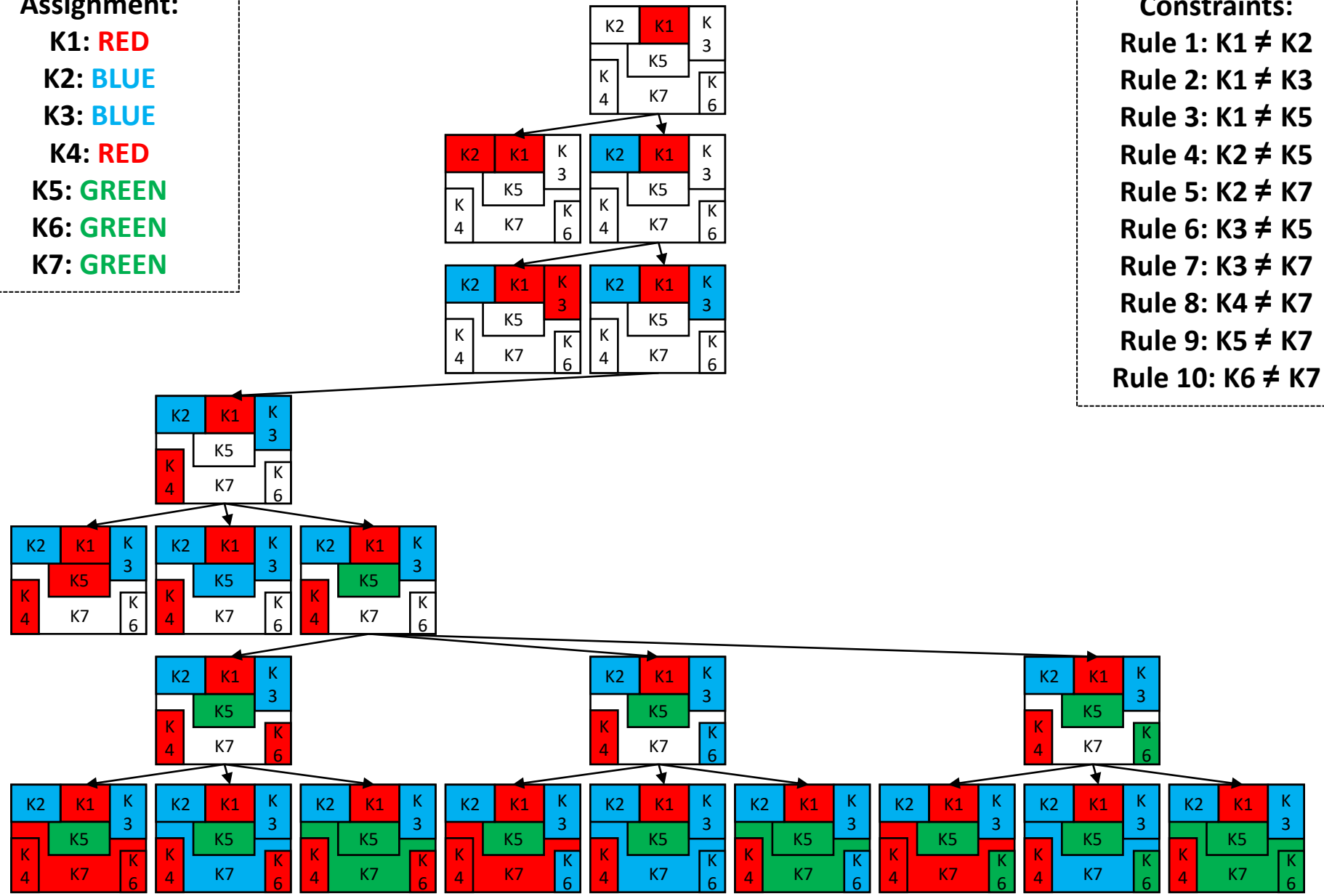
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: RED

K5: GREEN

K6: GREEN

K7: GREEN

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

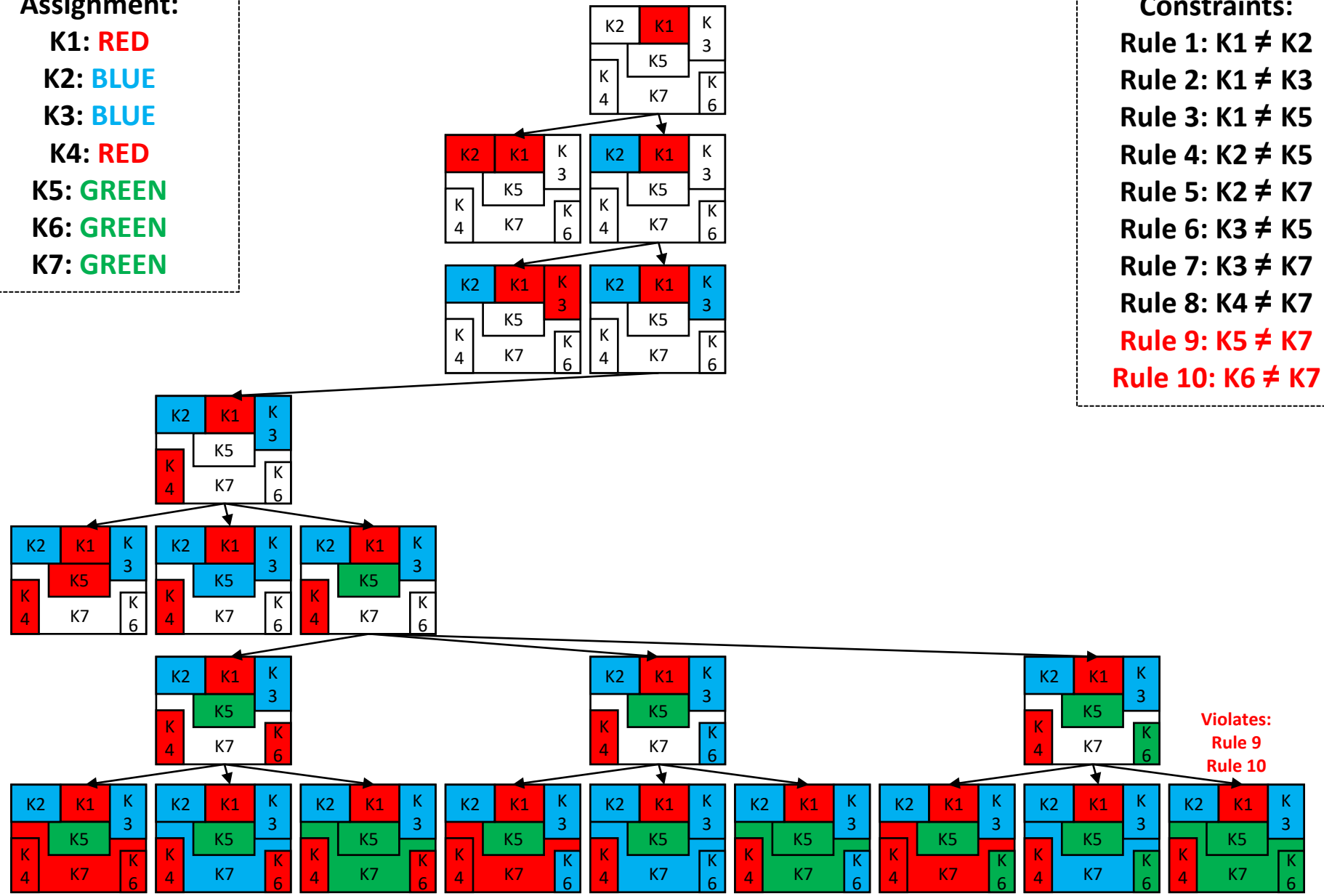
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

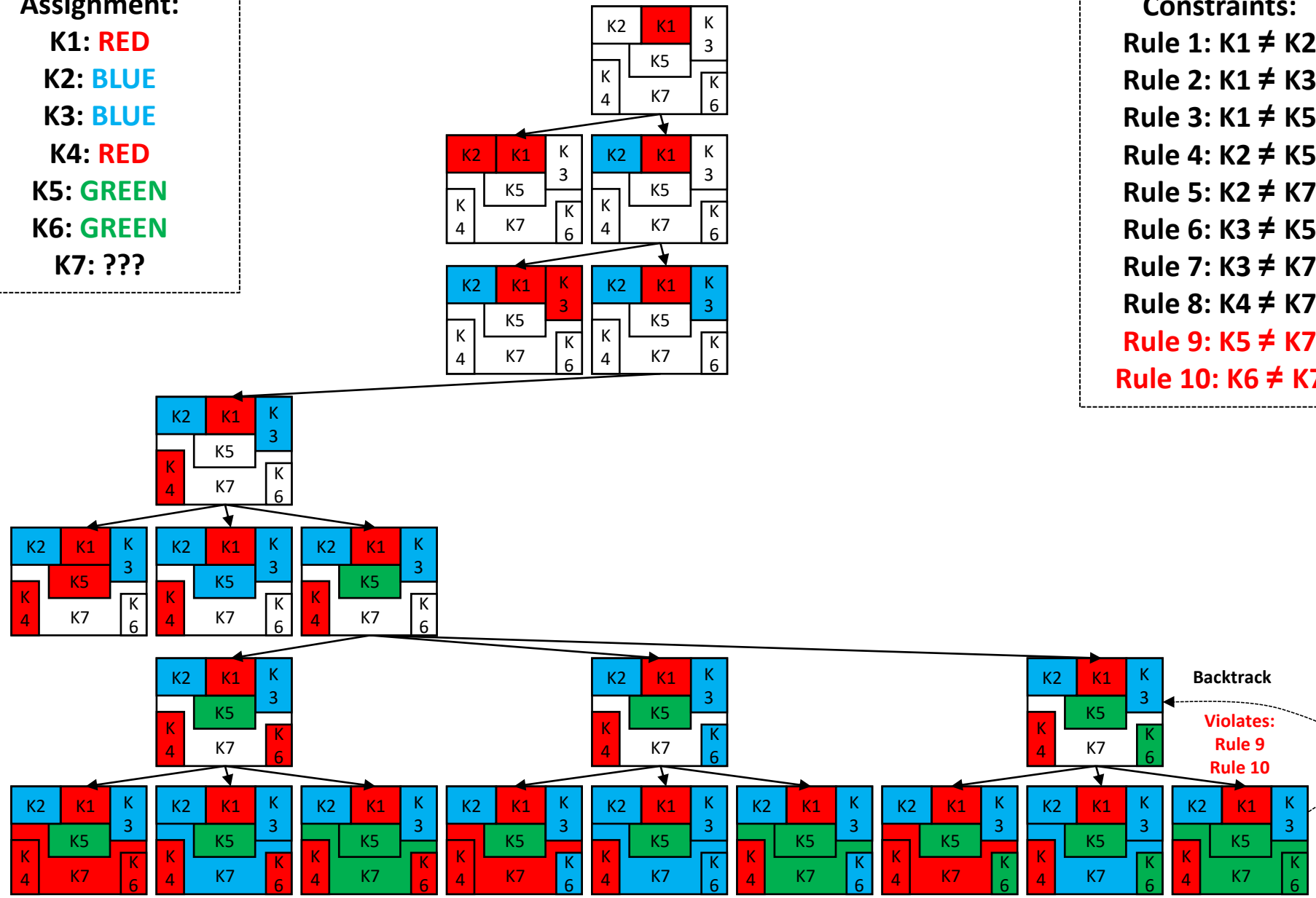
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: GREEN
K7: ???

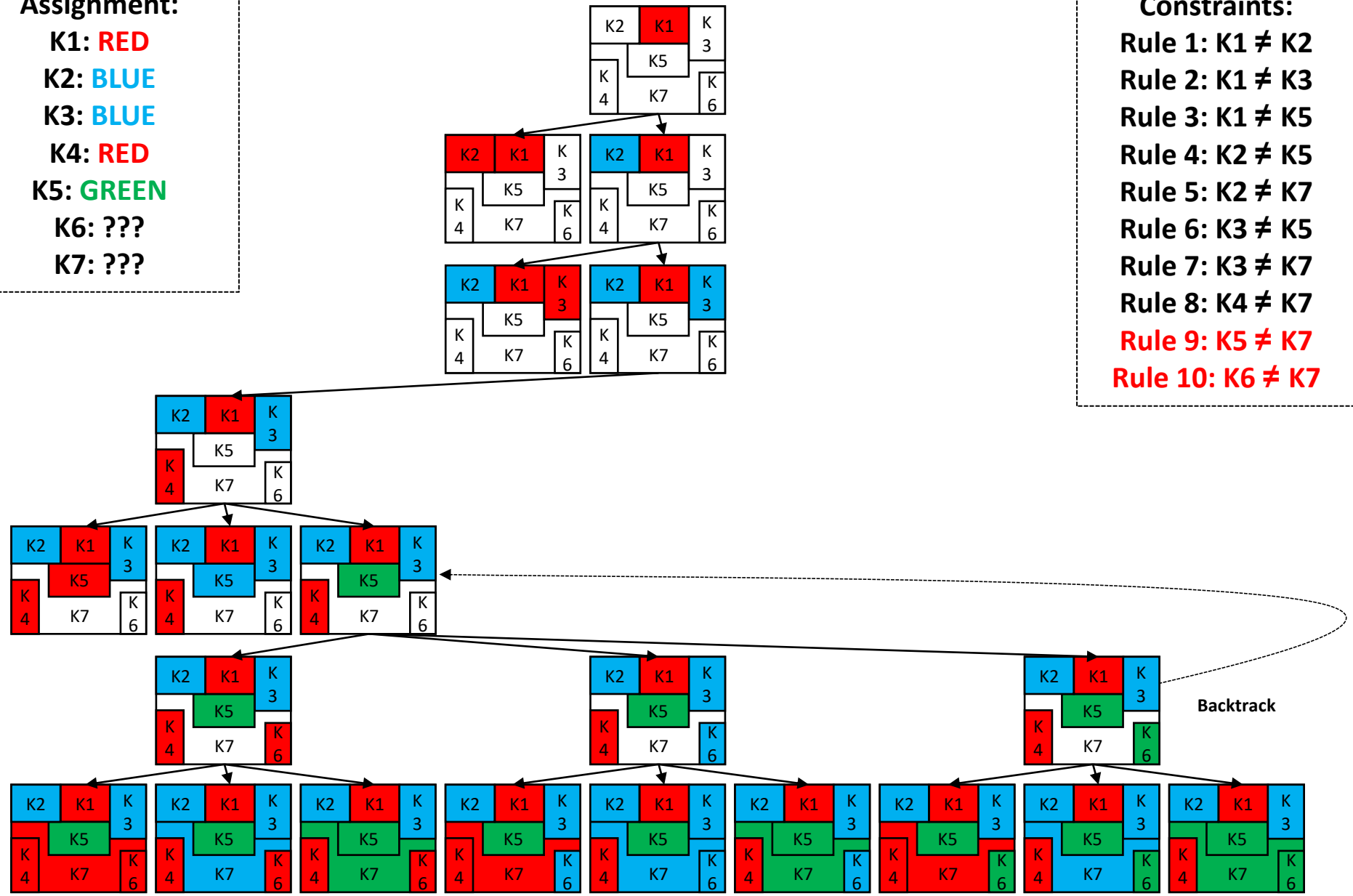
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: ???
K7: ???

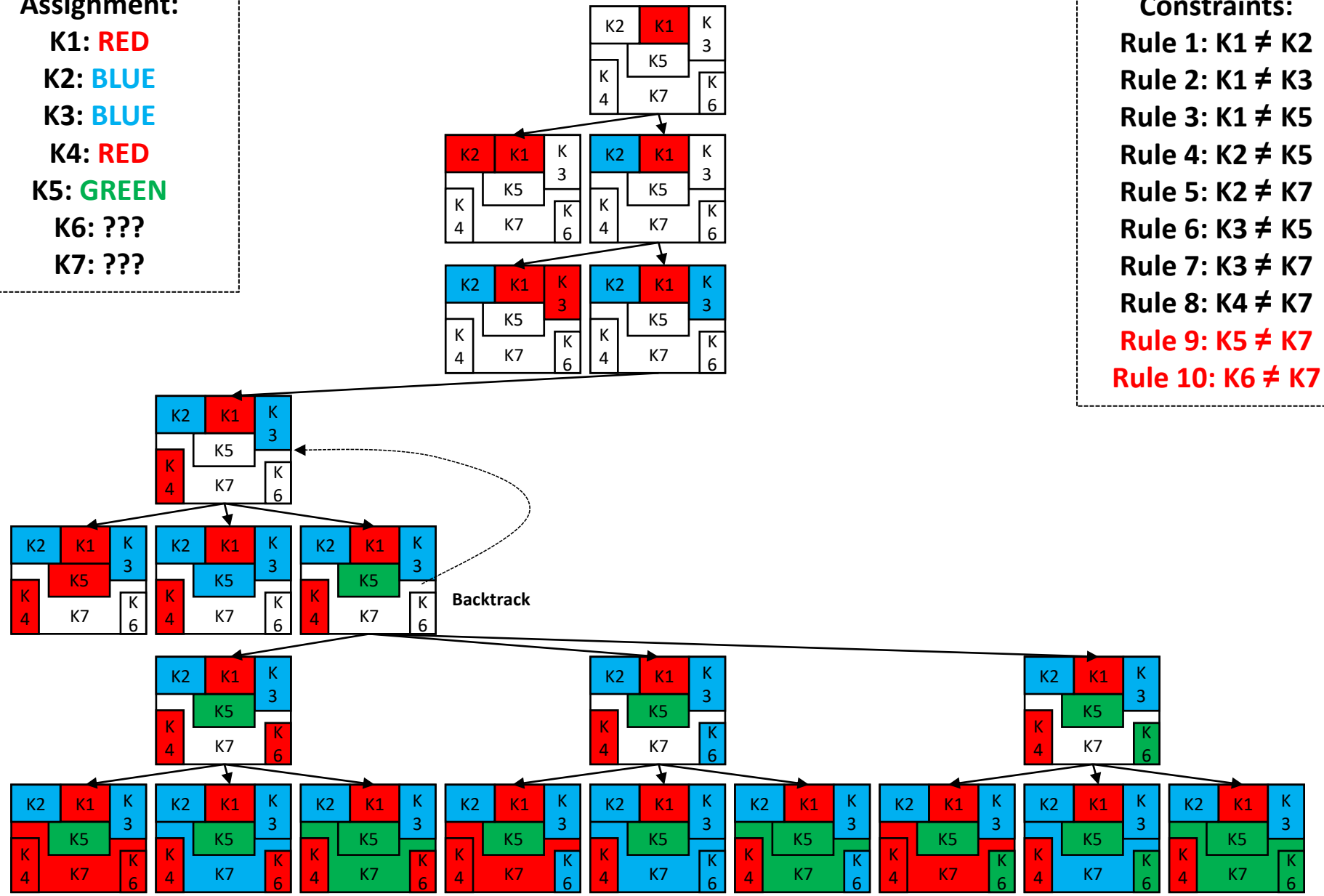
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: GREEN
K6: ???
K7: ???

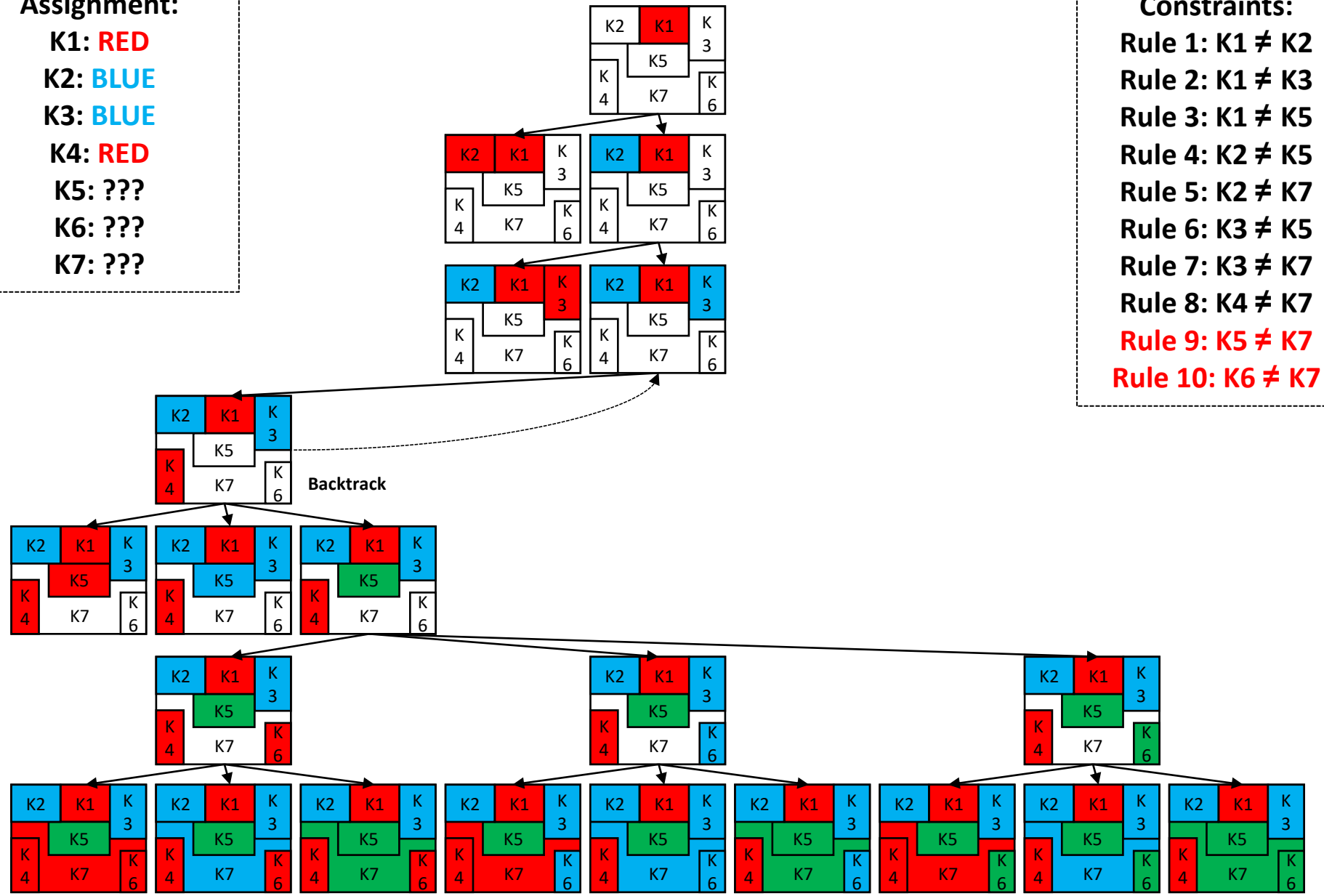
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: RED
K5: ???
K6: ???
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: BLUE

K5: ???

K6: ???

K7: ???

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

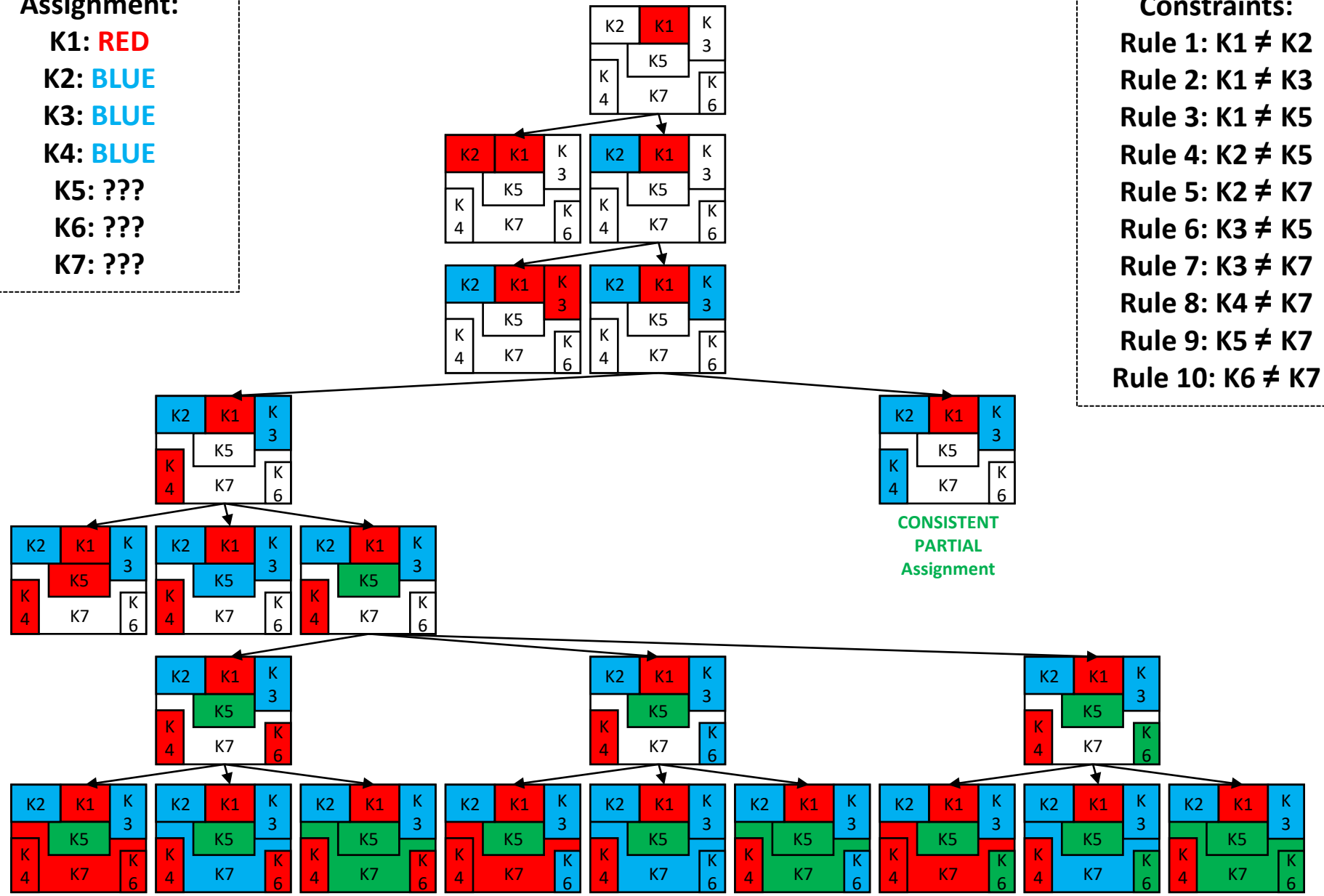
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: BLUE

K5: RED

K6: ???

K7: ???

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

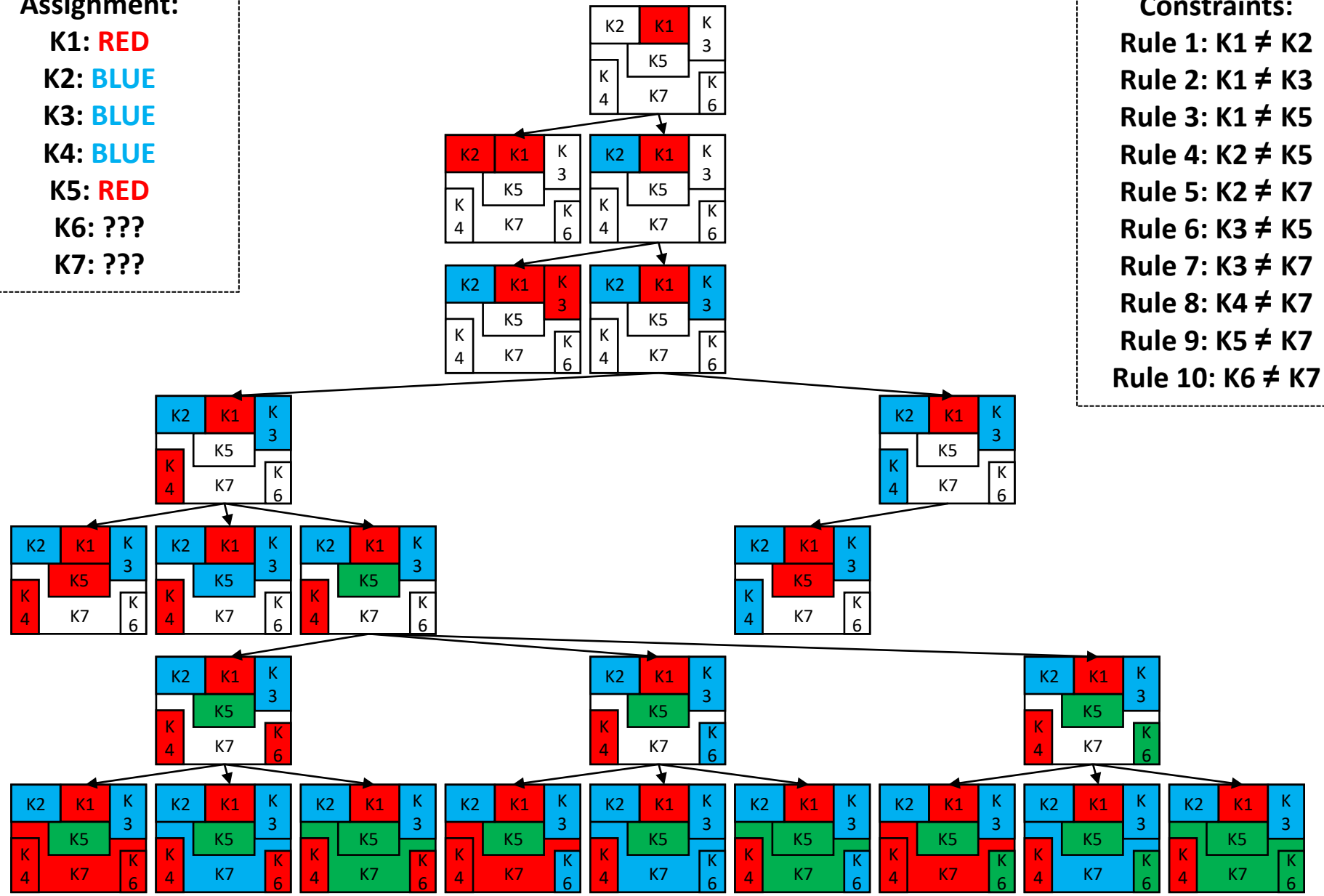
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: BLUE

K5: RED

K6: ???

K7: ???

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

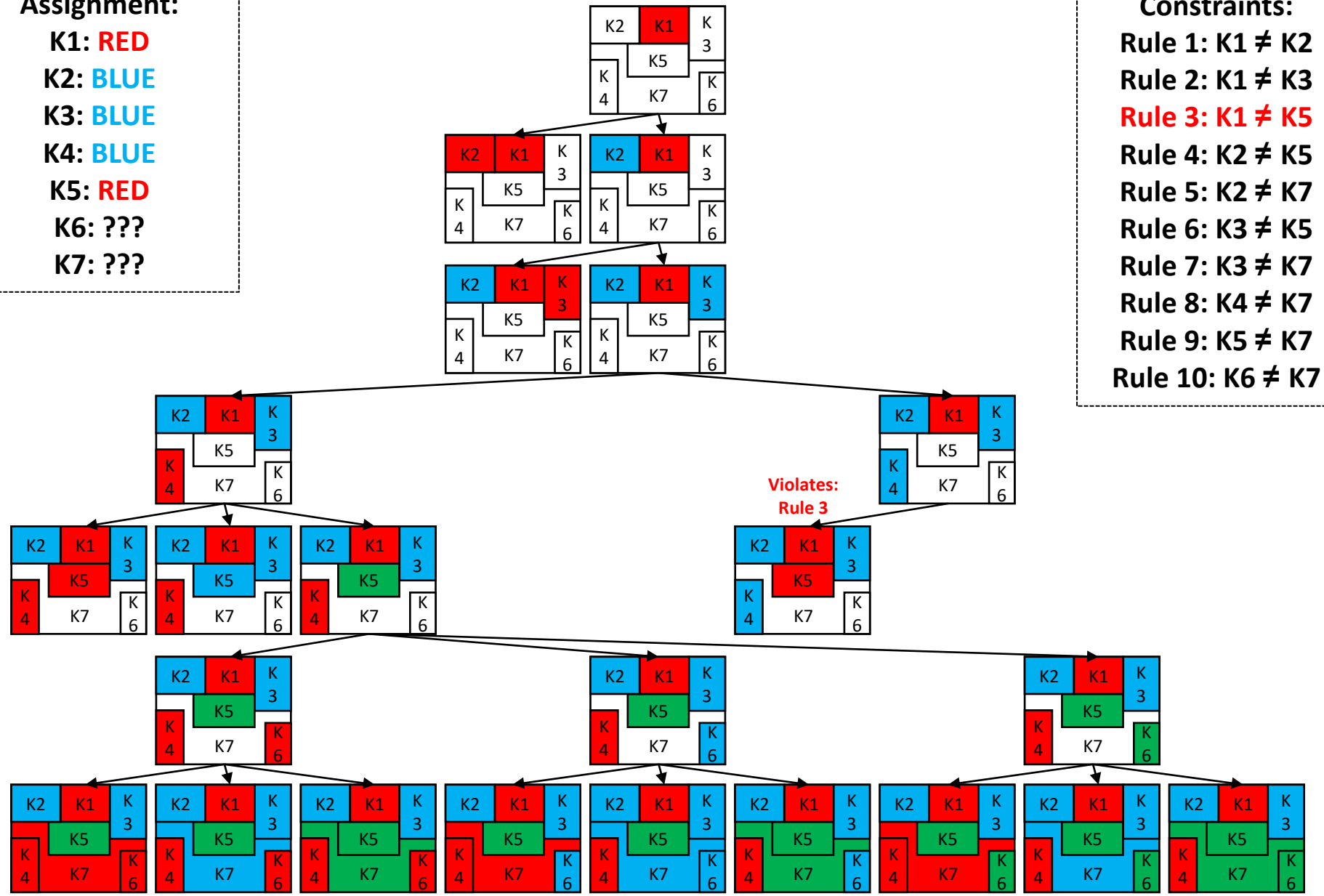
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

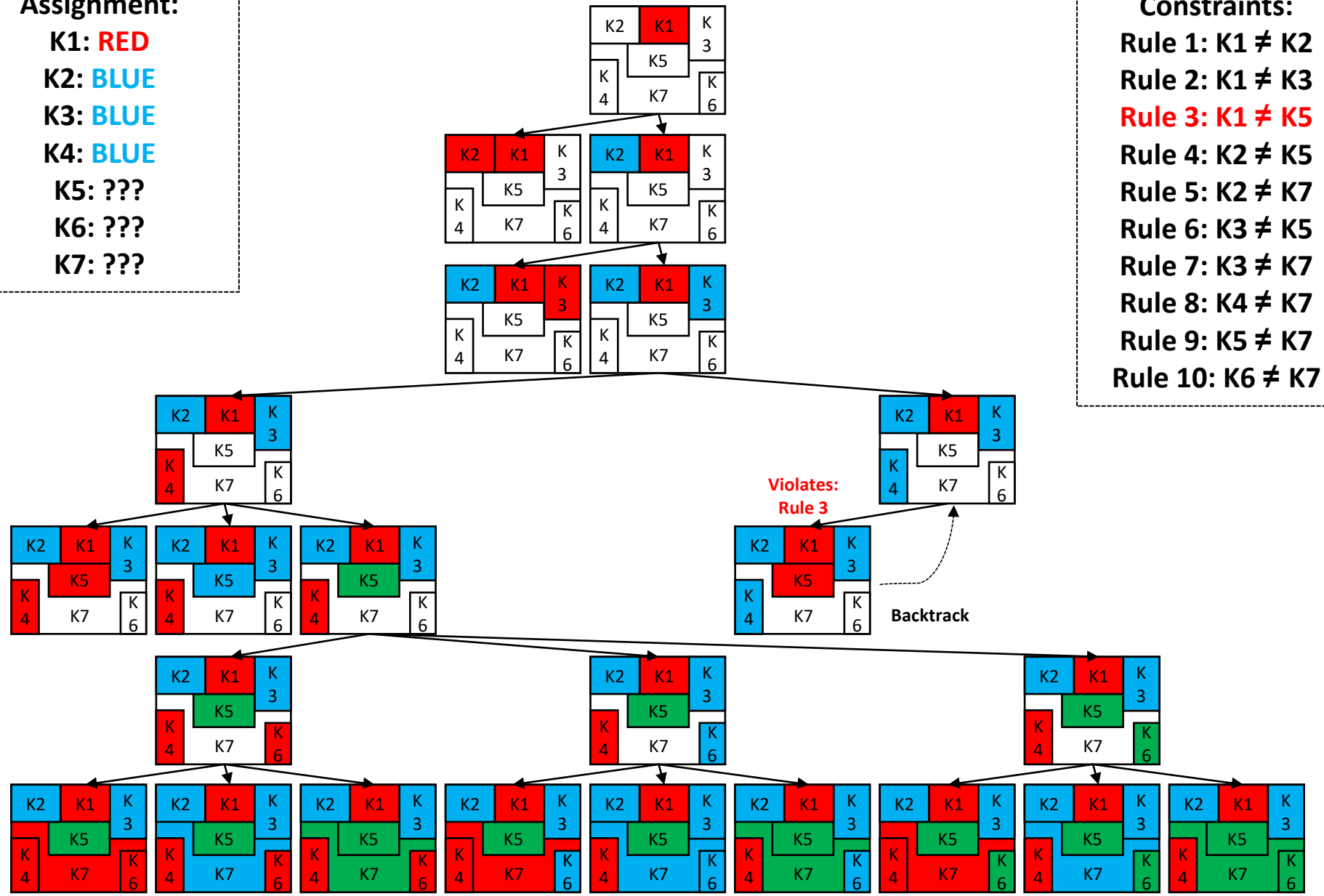
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: ???
K6: ???
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: BLUE

K5: BLUE

K6: ???

K7: ???

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

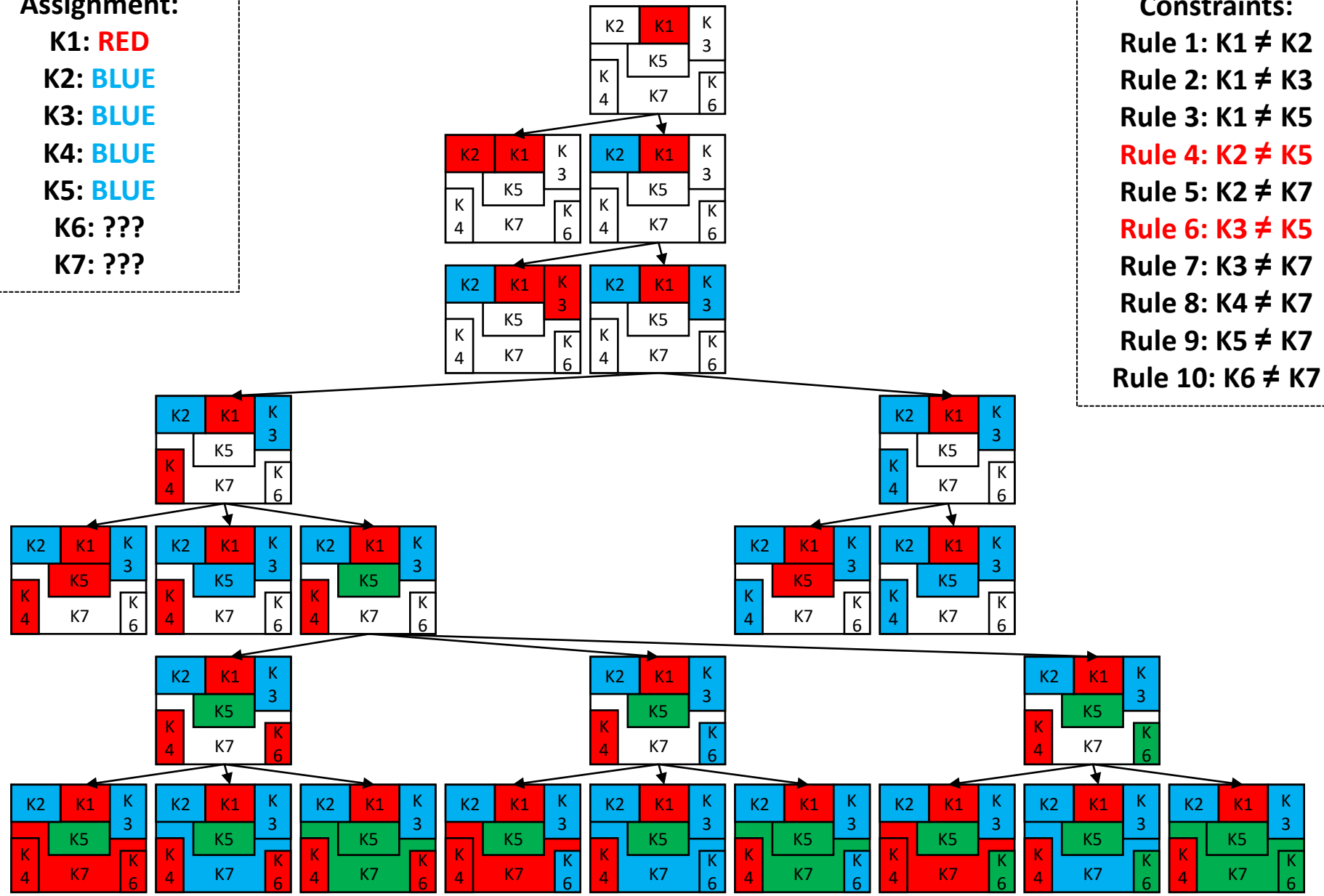
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:

K1: RED

K2: BLUE

K3: BLUE

K4: BLUE

K5: BLUE

K6: ???

K7: ???

Constraints:

Rule 1: K1 ≠ K2

Rule 2: K1 ≠ K3

Rule 3: K1 ≠ K5

Rule 4: K2 ≠ K5

Rule 5: K2 ≠ K7

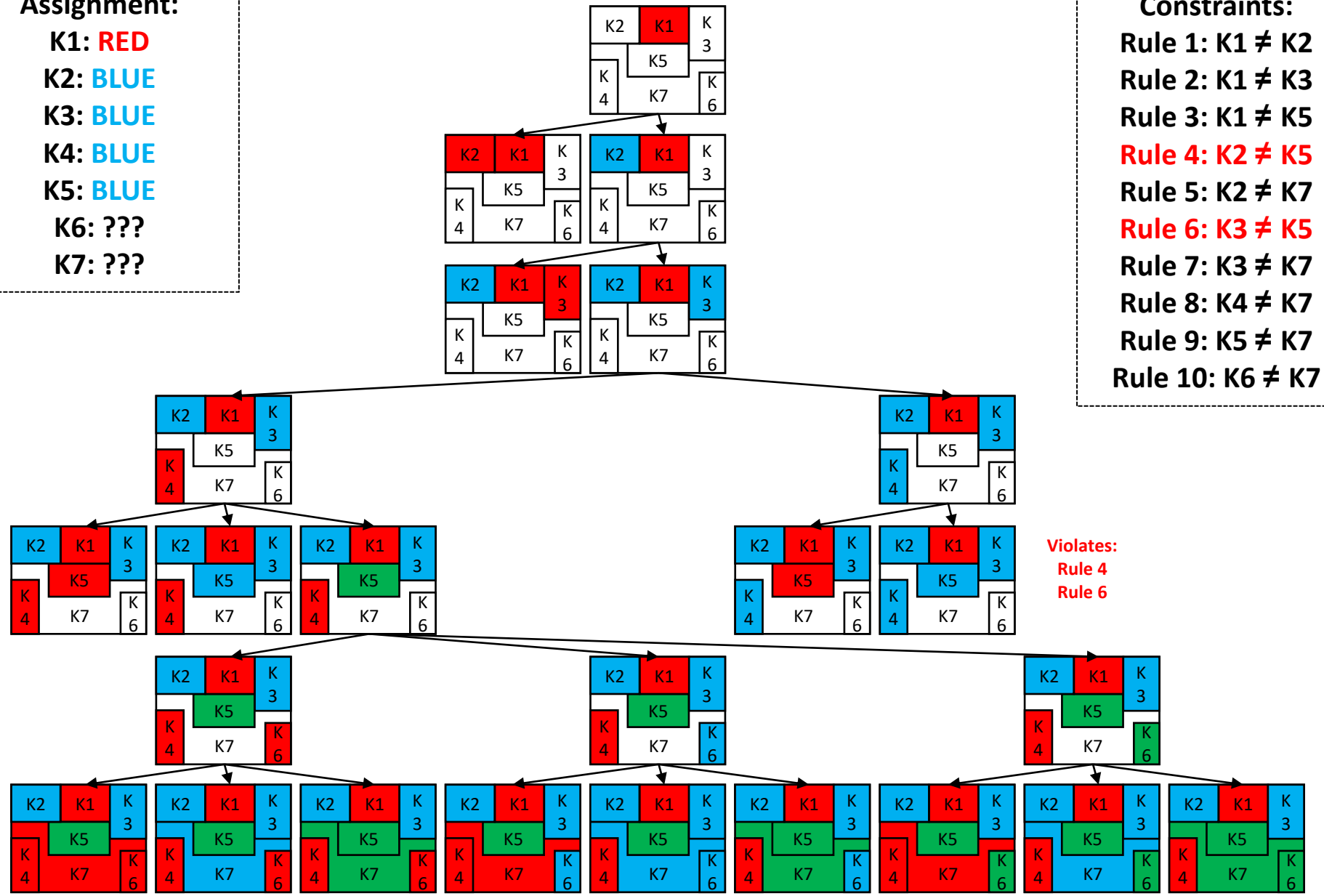
Rule 6: K3 ≠ K5

Rule 7: K3 ≠ K7

Rule 8: K4 ≠ K7

Rule 9: K5 ≠ K7

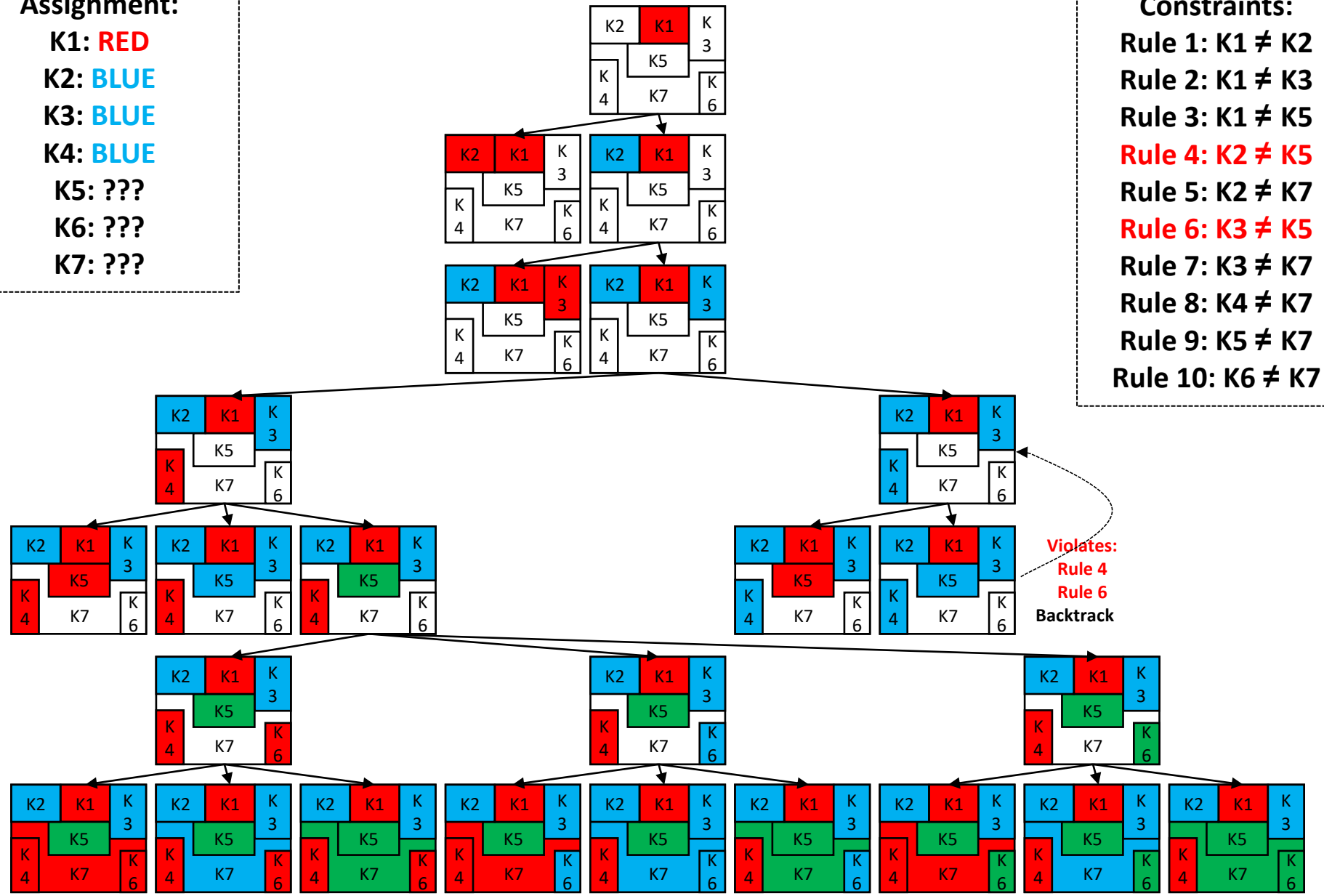
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: ???
K6: ???
K7: ???

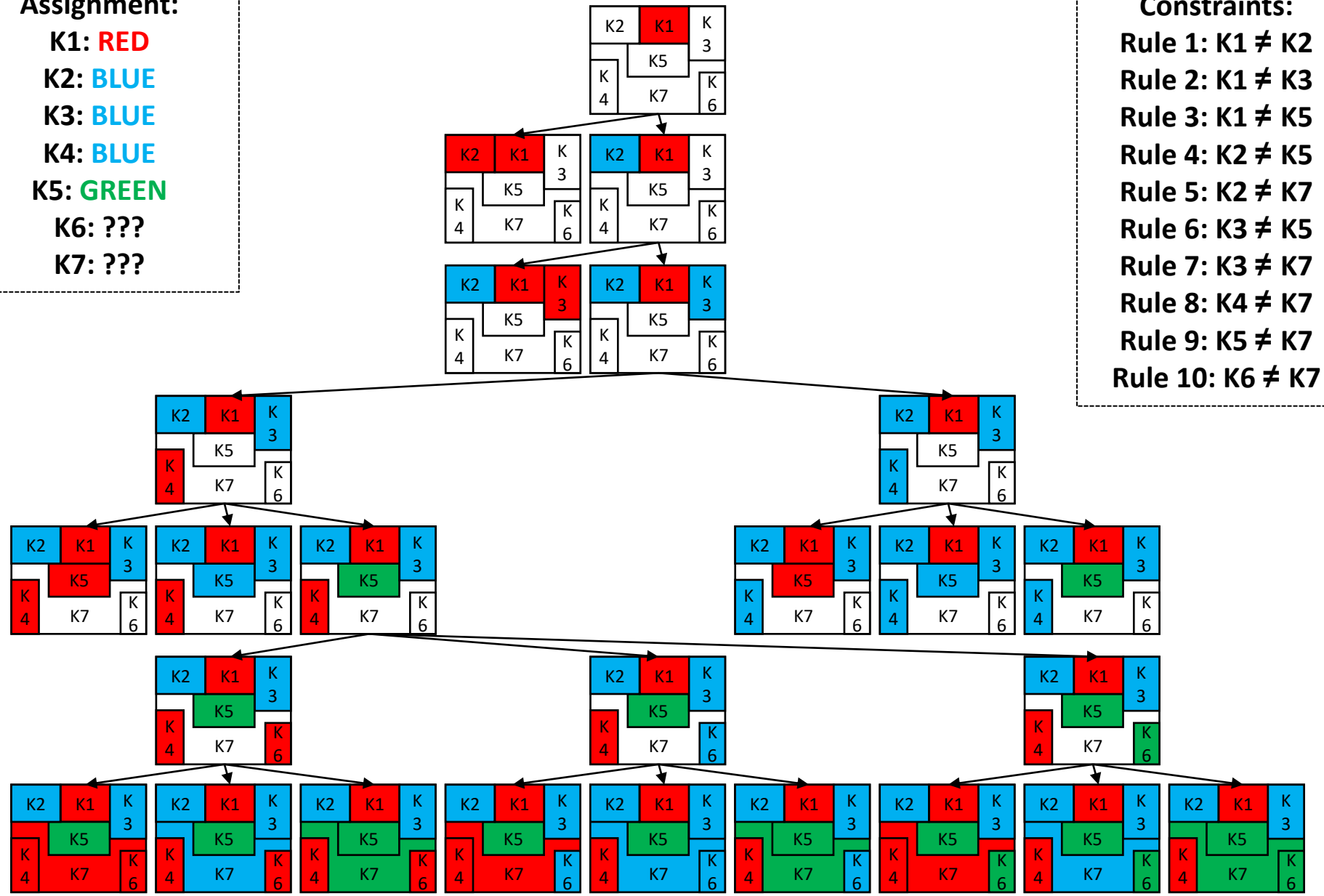
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: ???
K7: ???

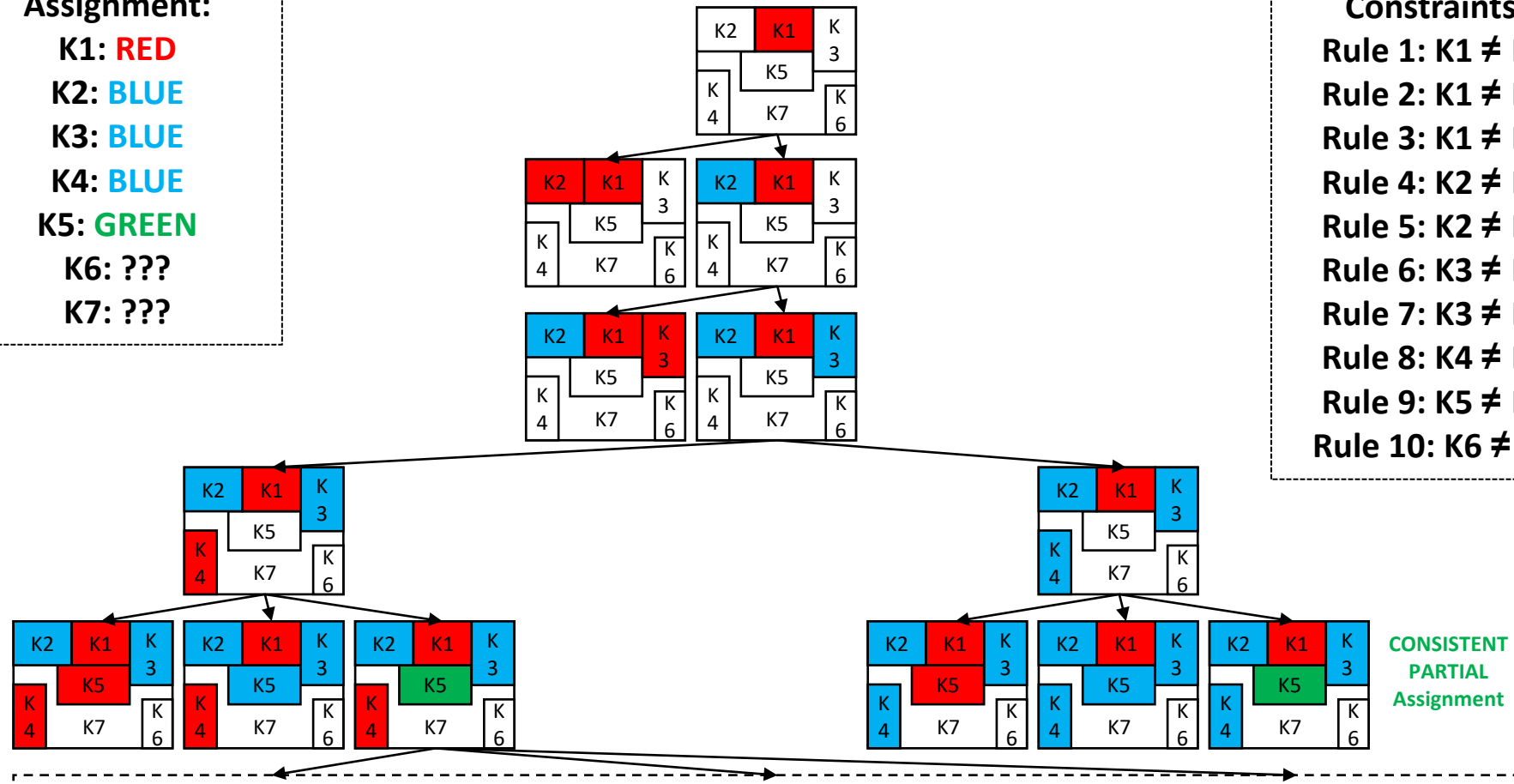
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: ???
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

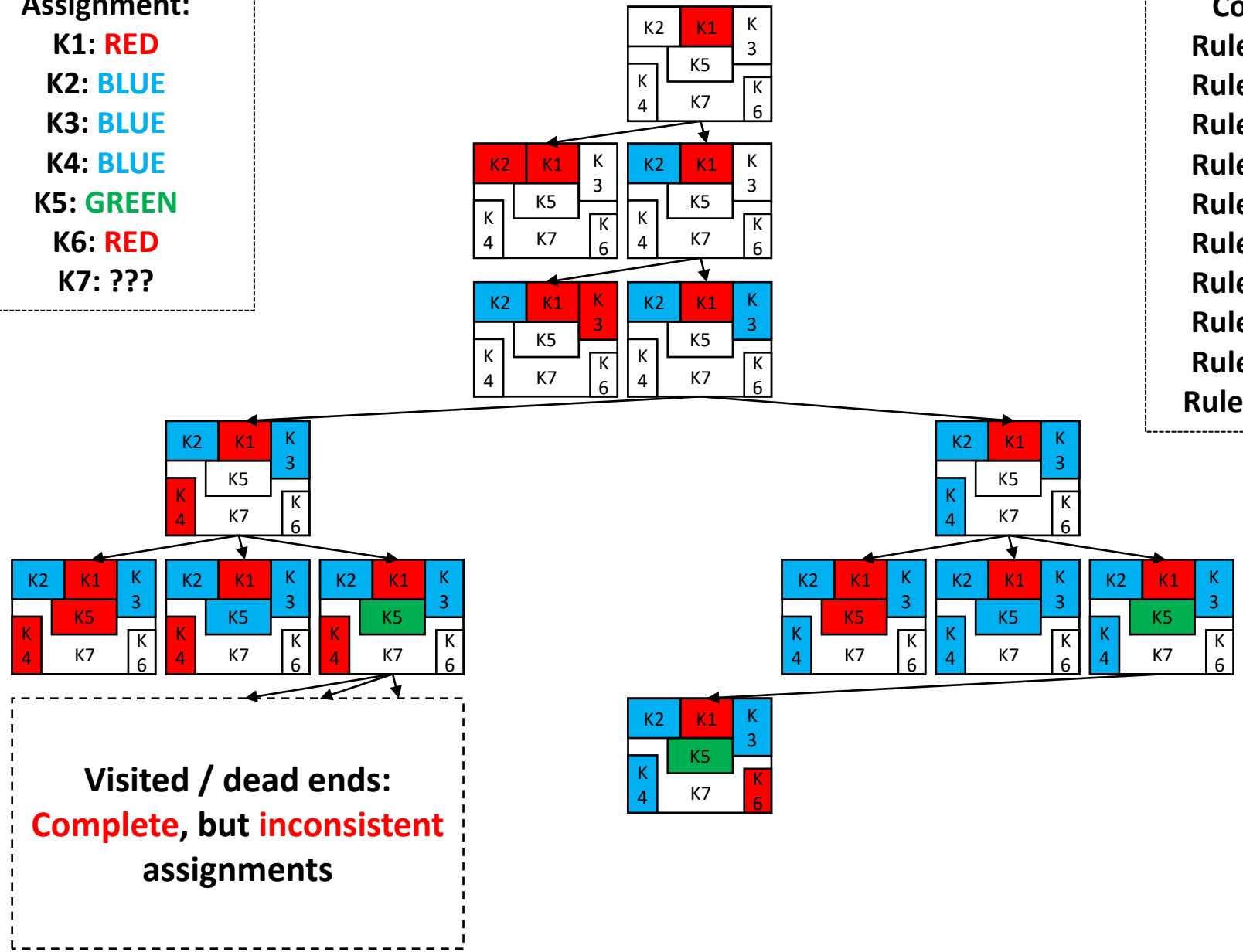


Visited / dead ends
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order: RED, BLUE, GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

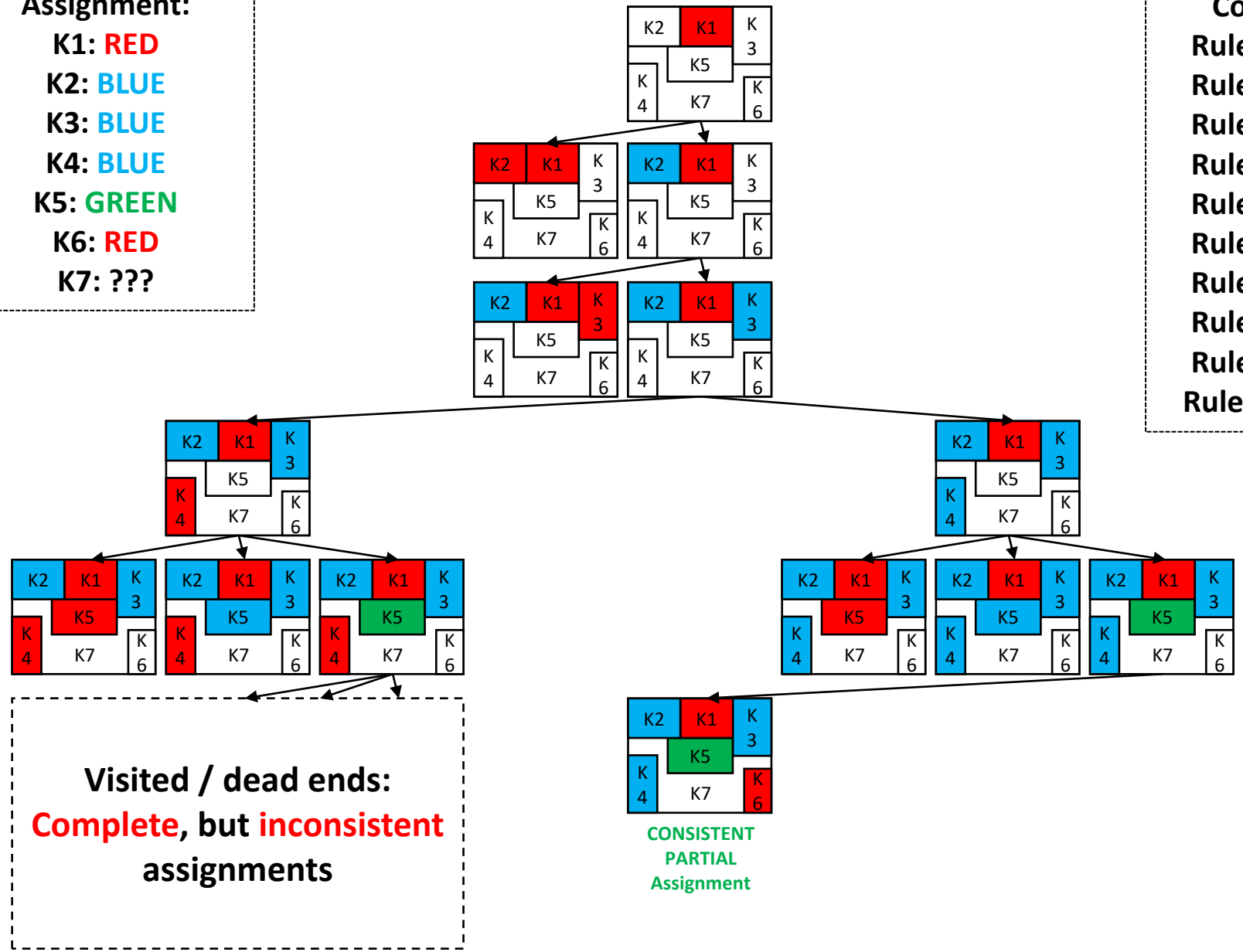


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

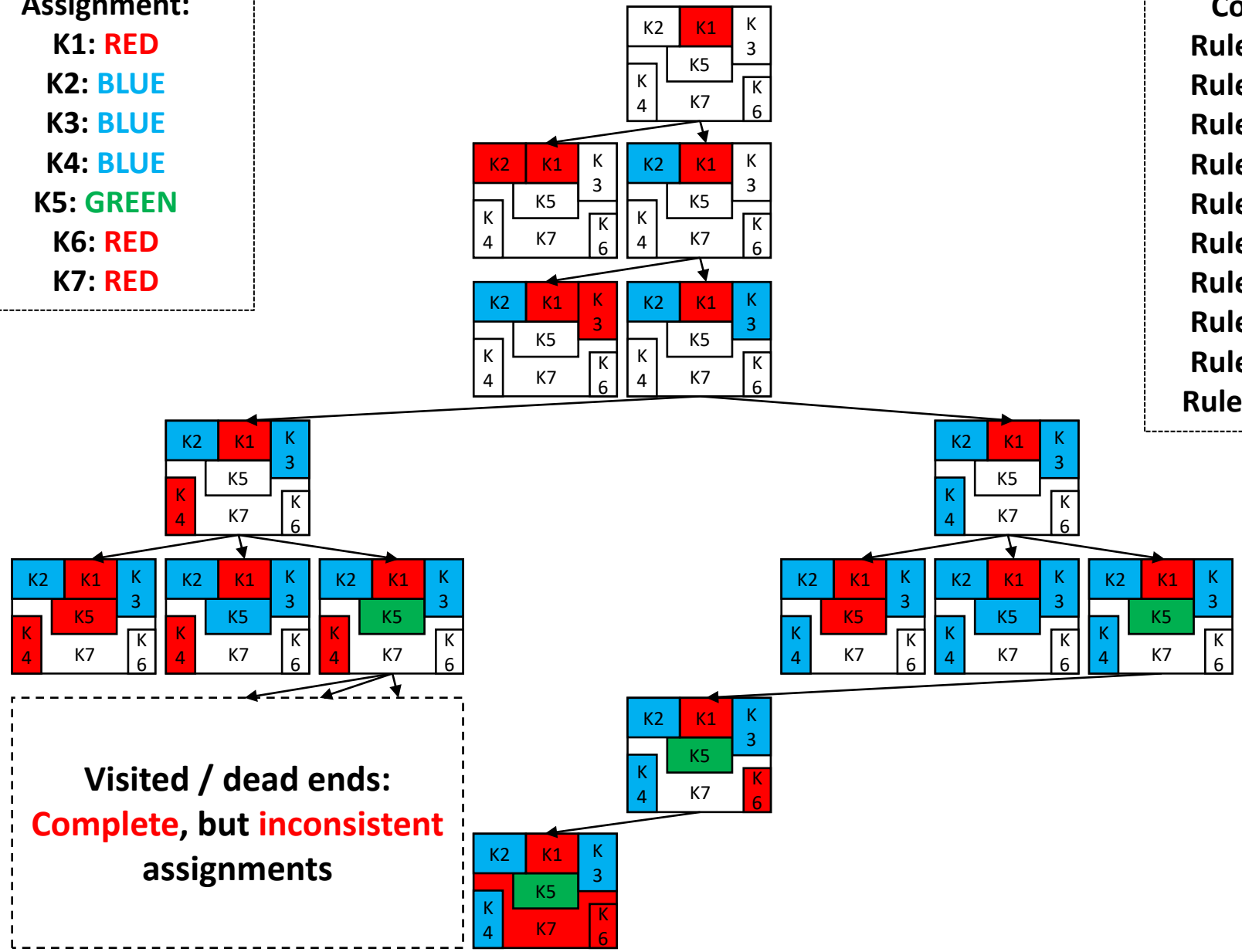


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: RED

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

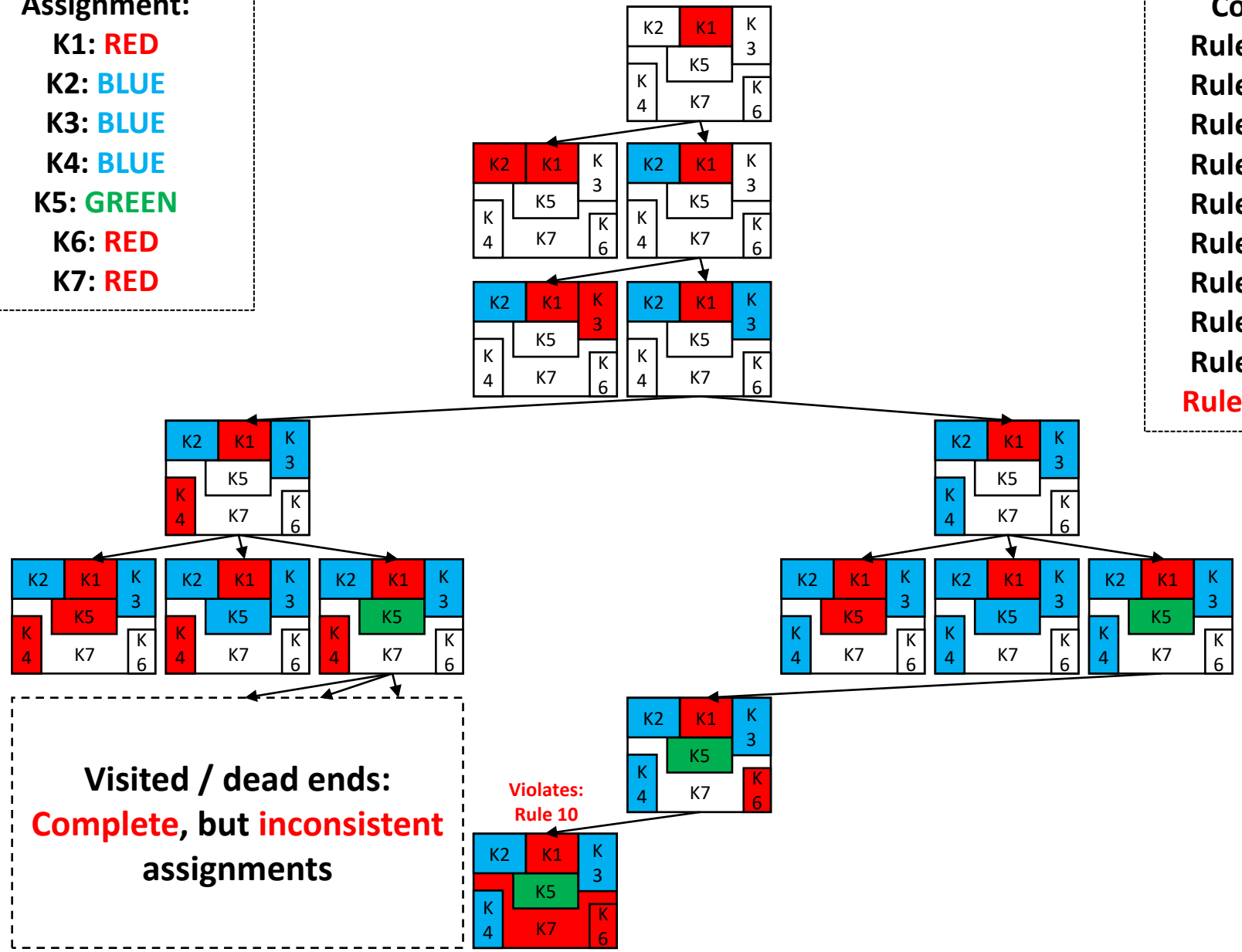


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: RED

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

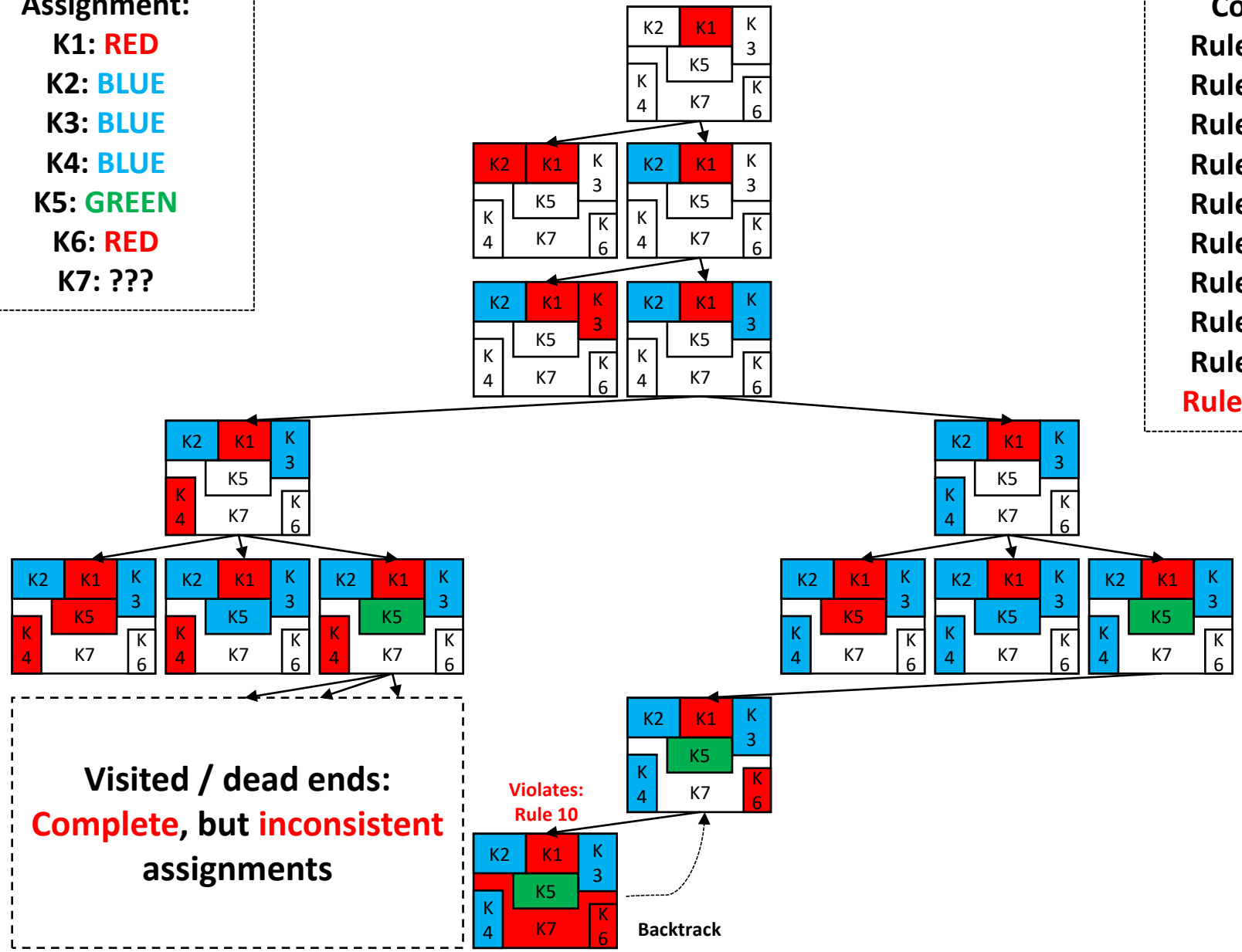


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

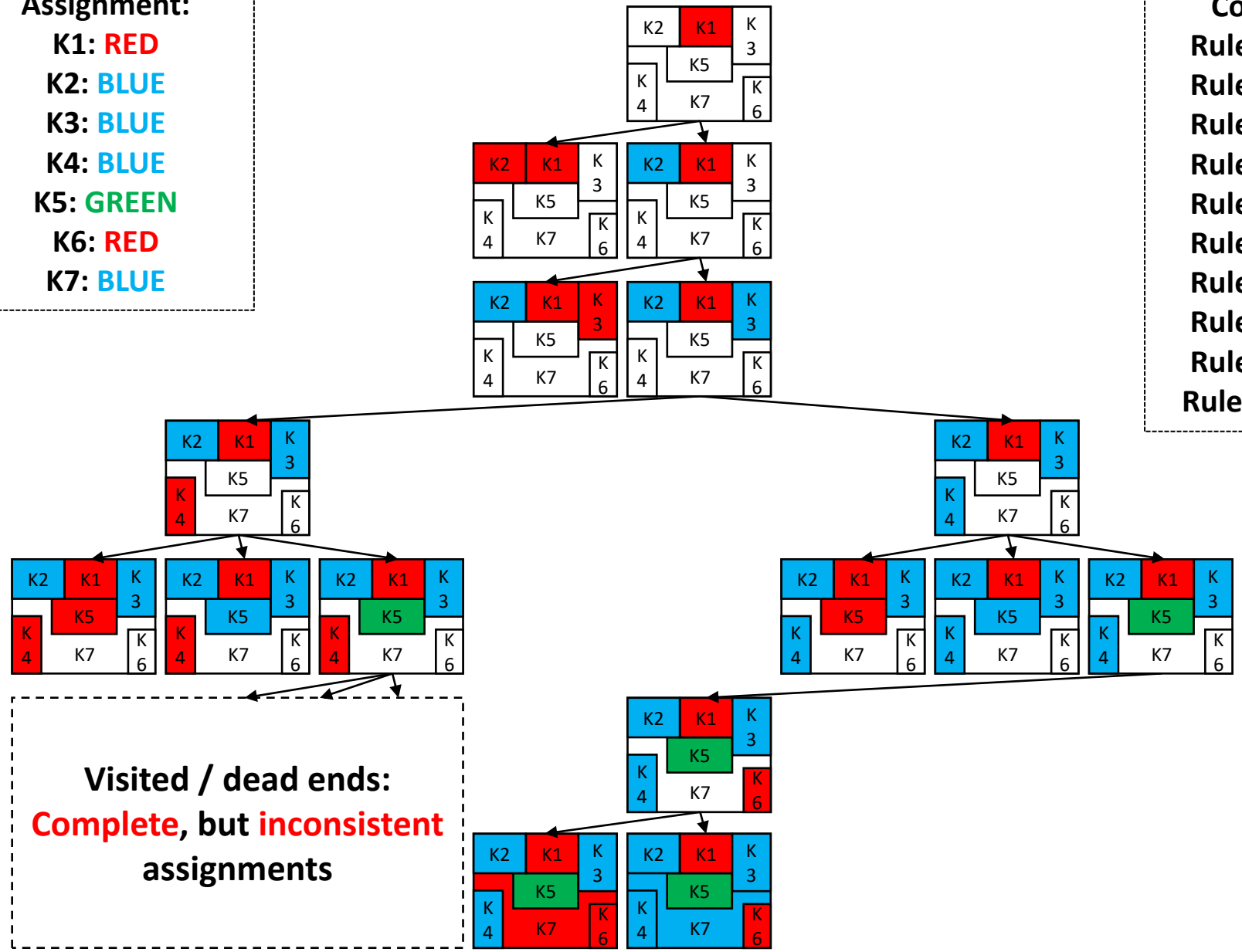


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: BLUE

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

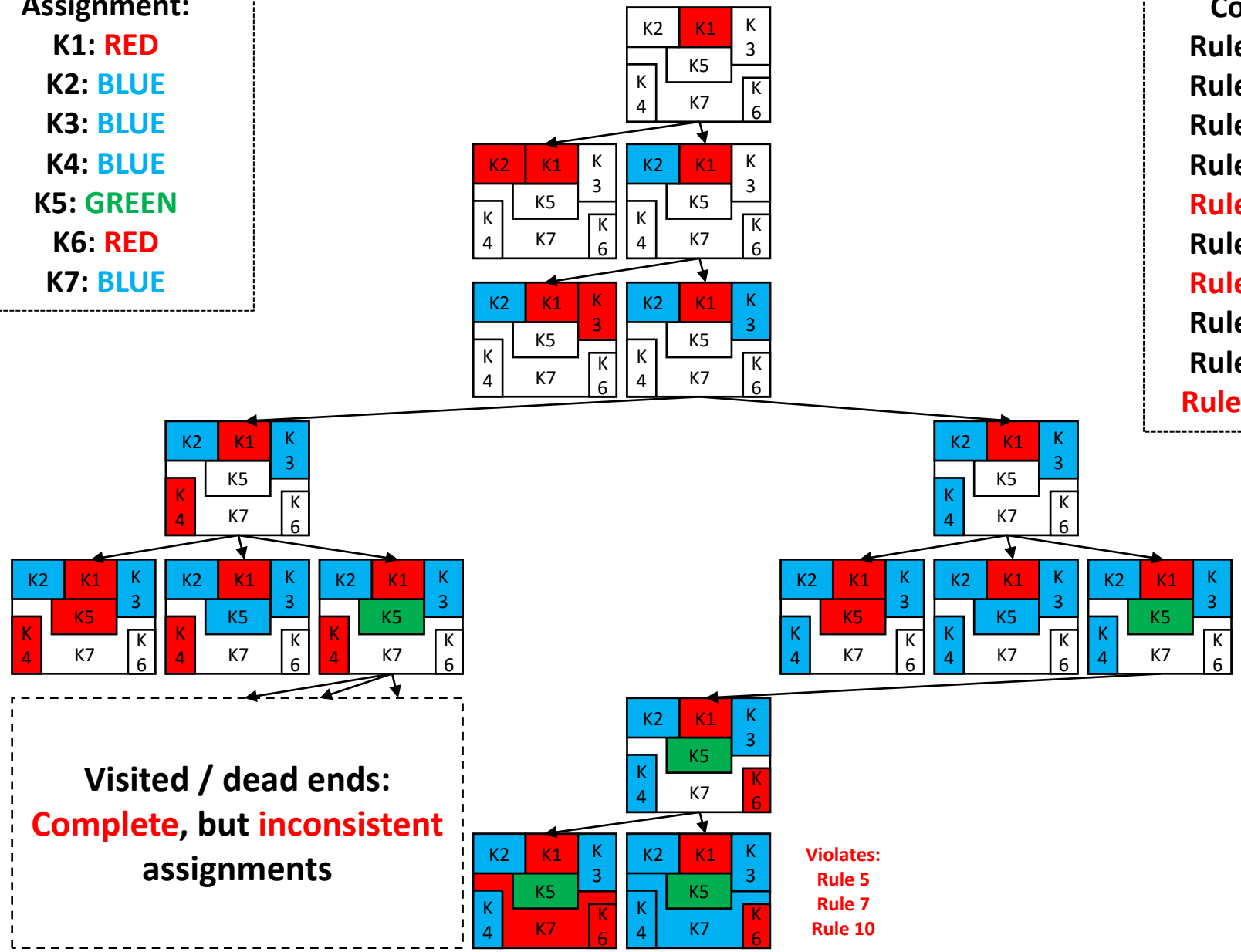


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: BLUE

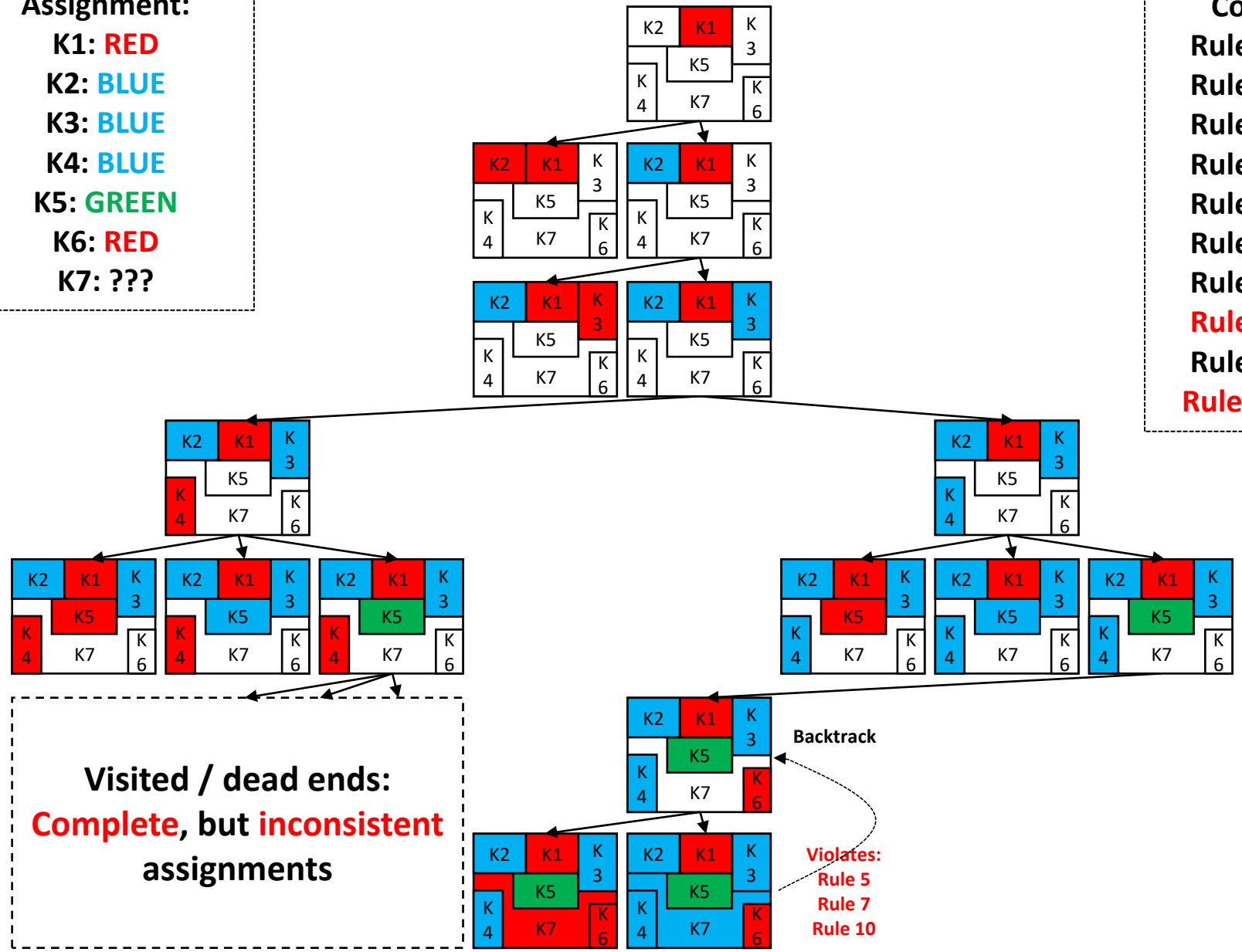
Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

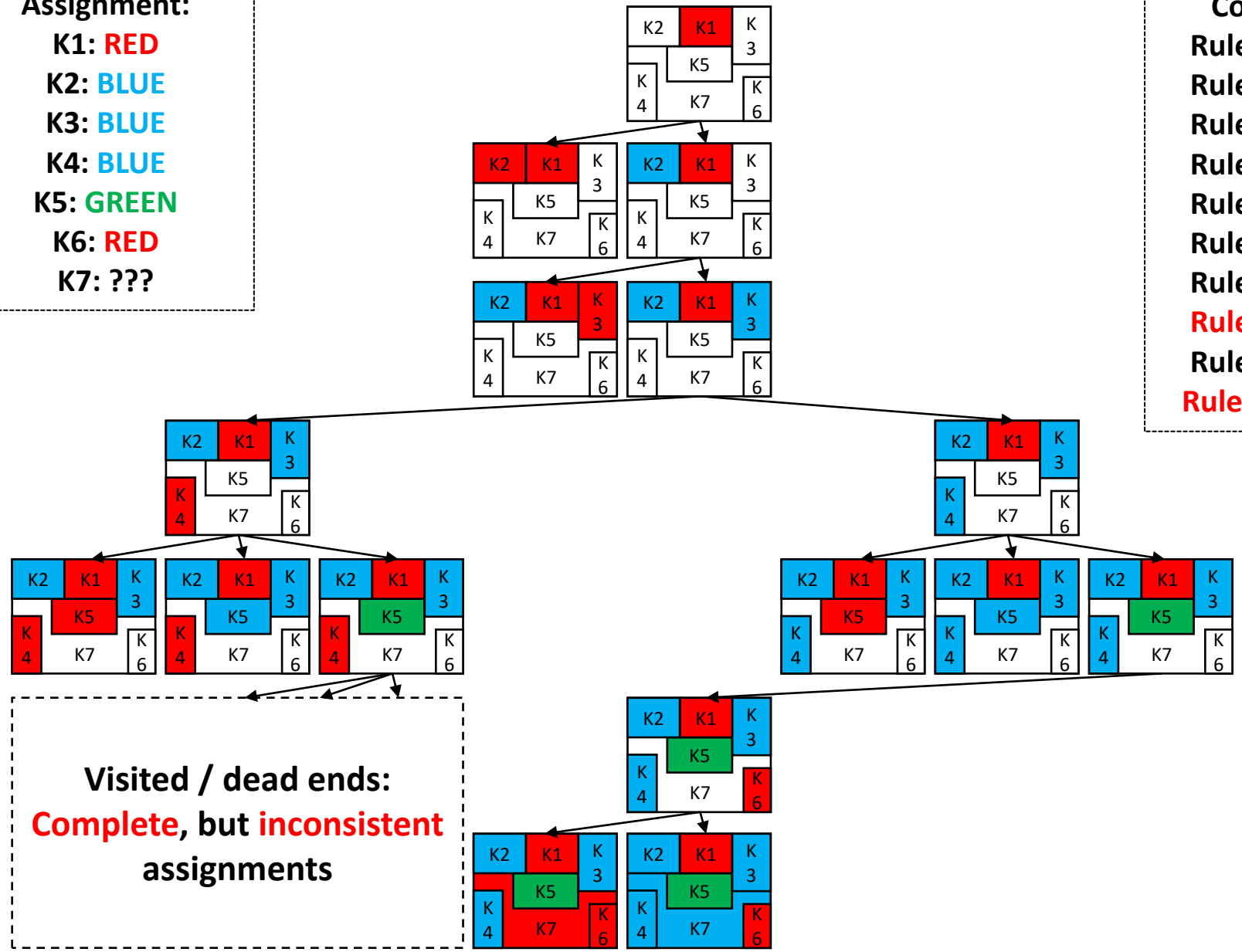


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

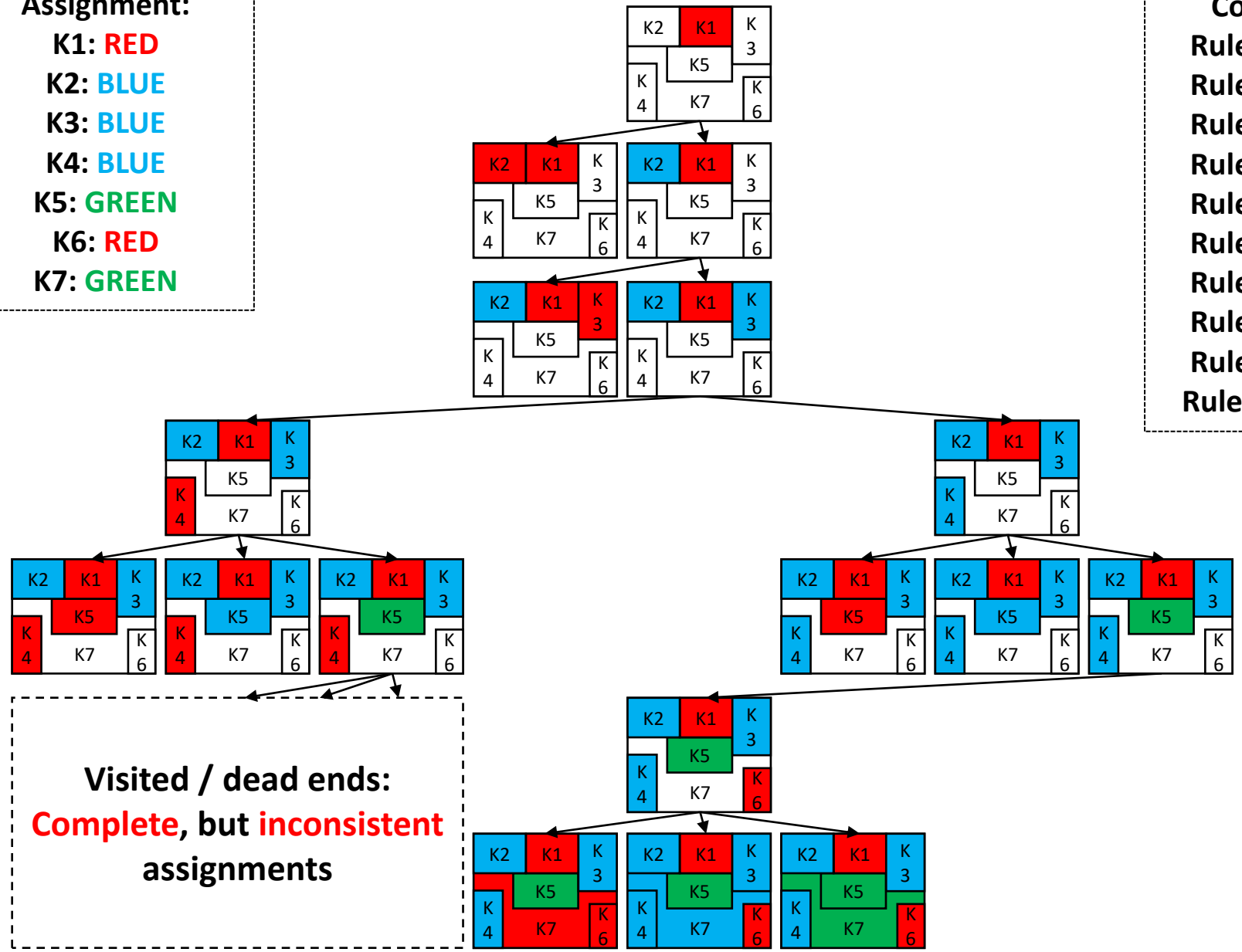


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: GREEN

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

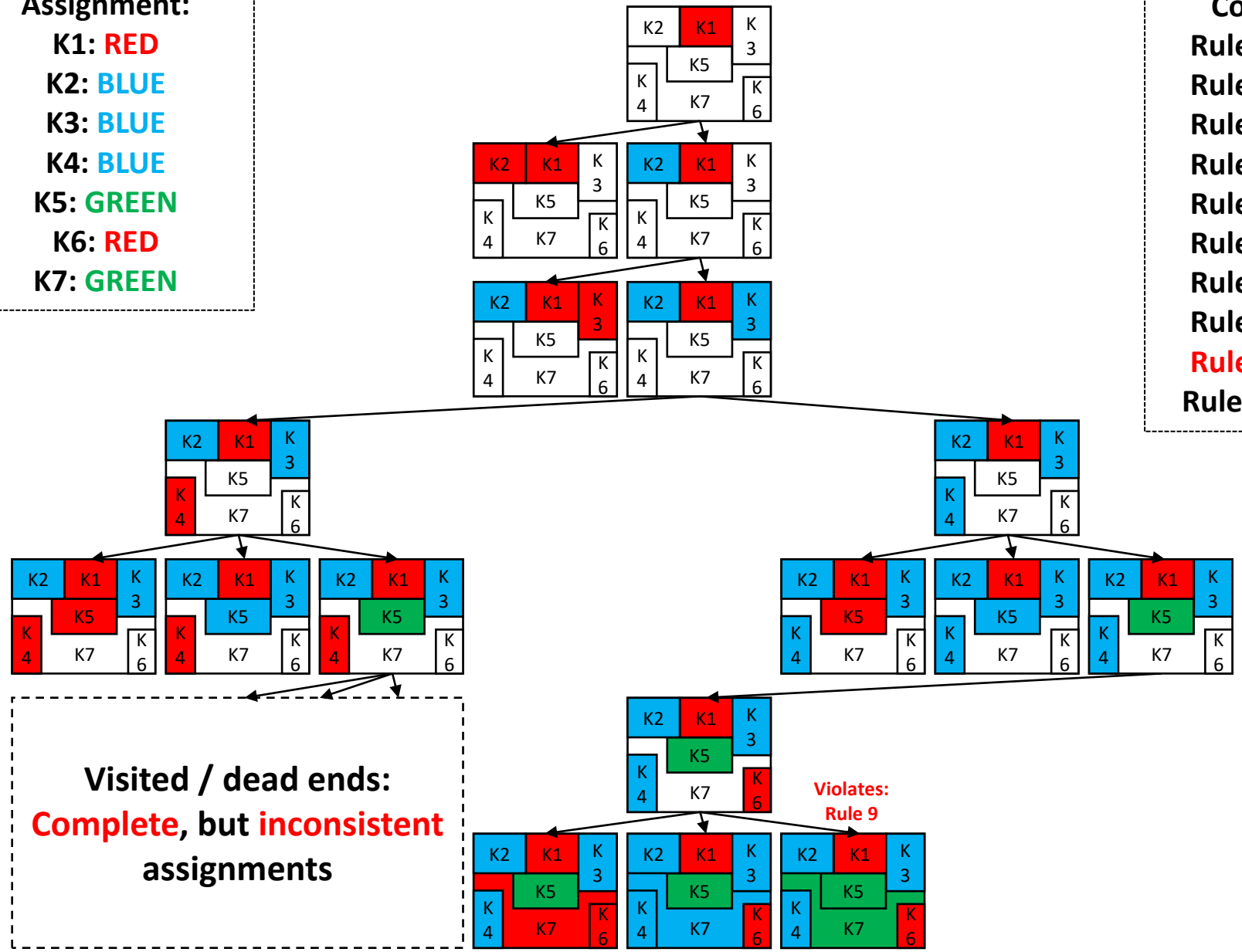


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: GREEN

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

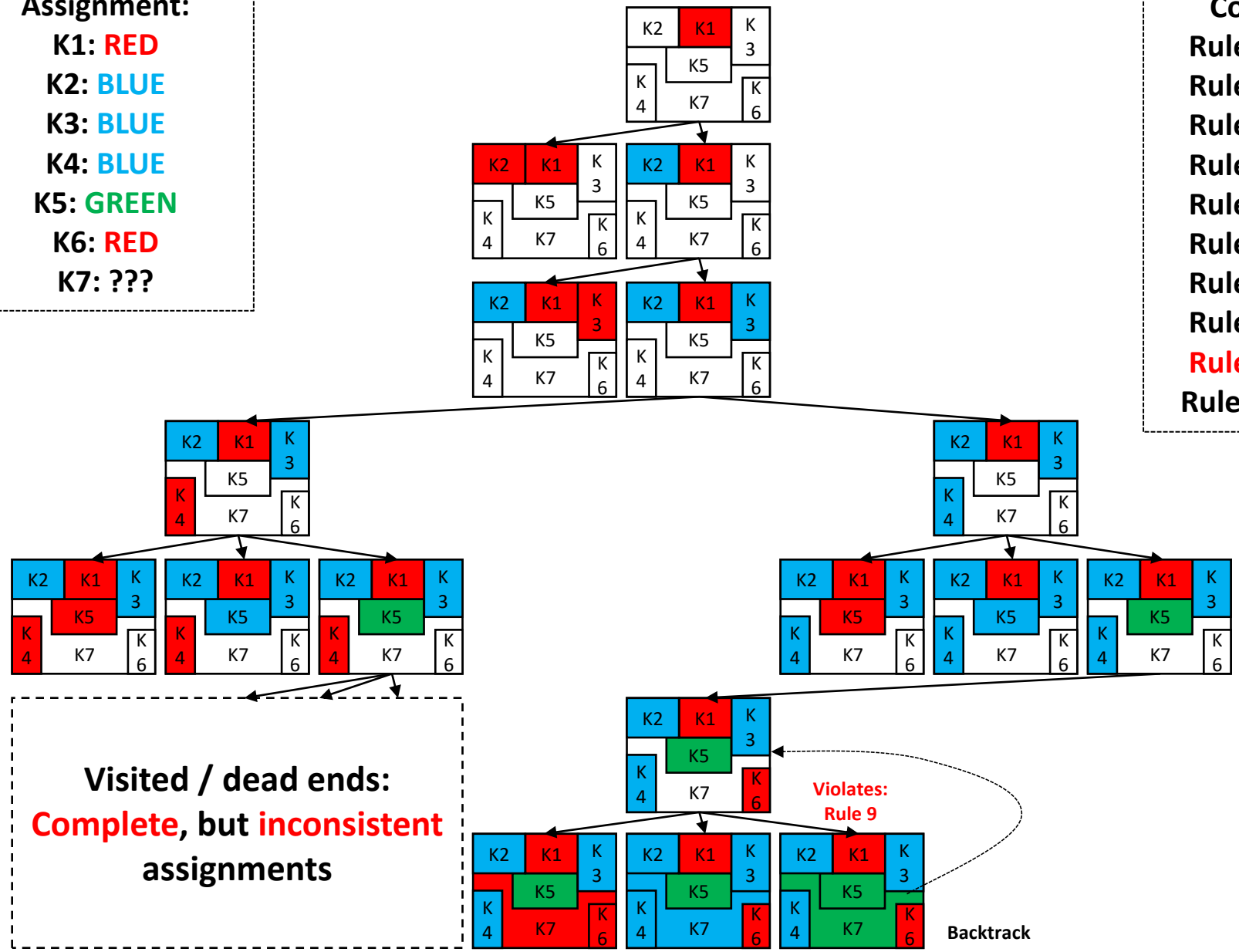


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: RED
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

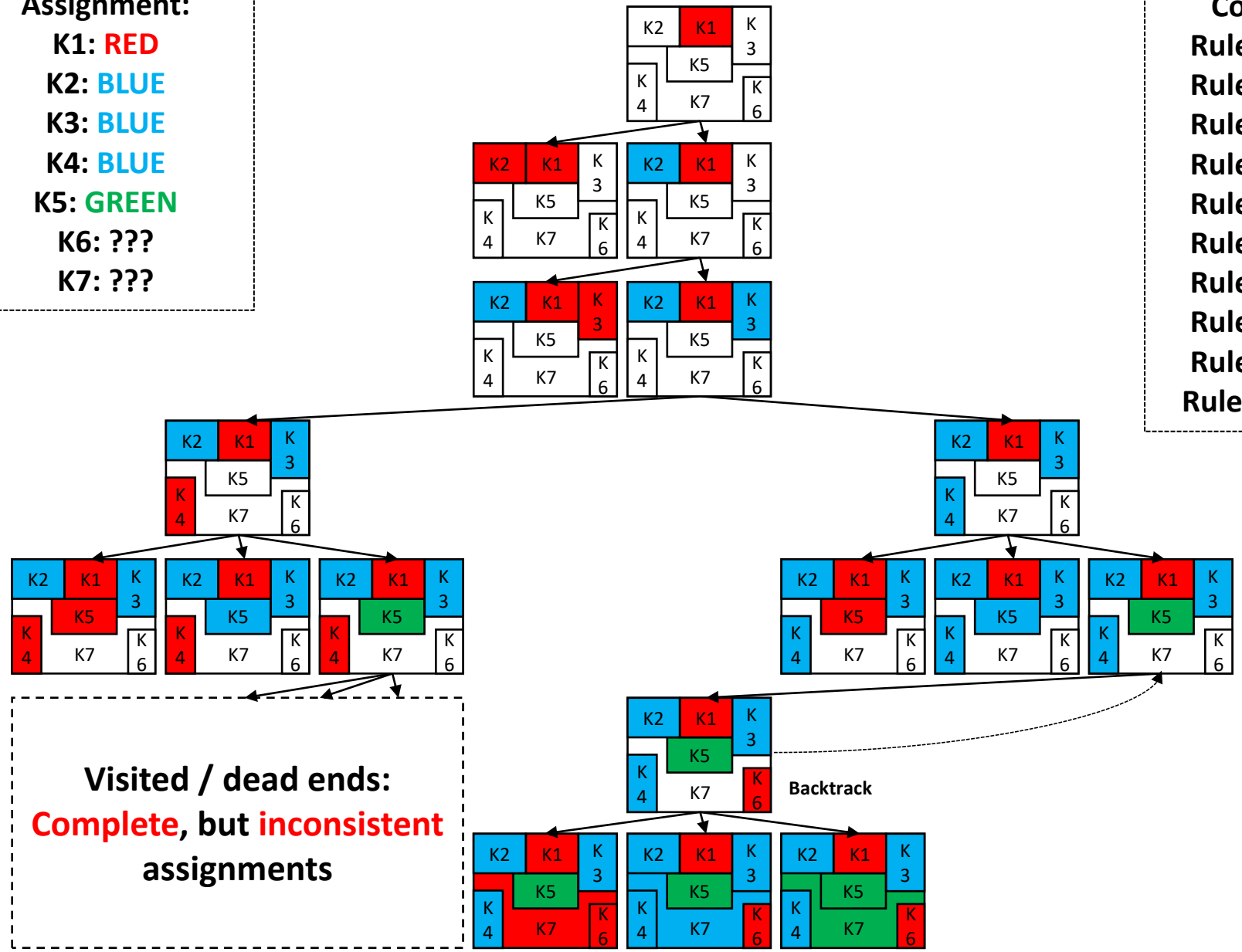


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: ???
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

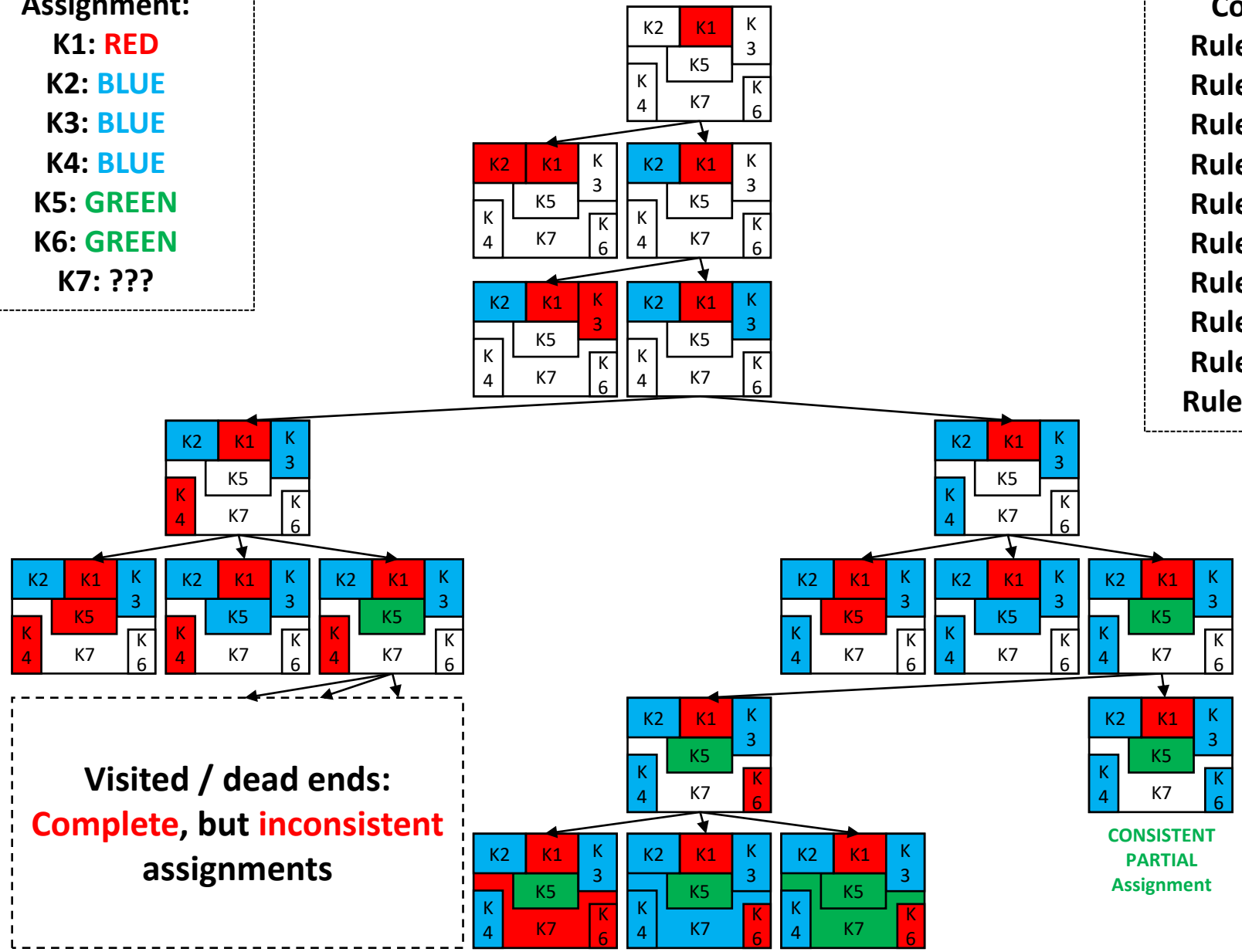


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: GREEN
K7: ???

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7

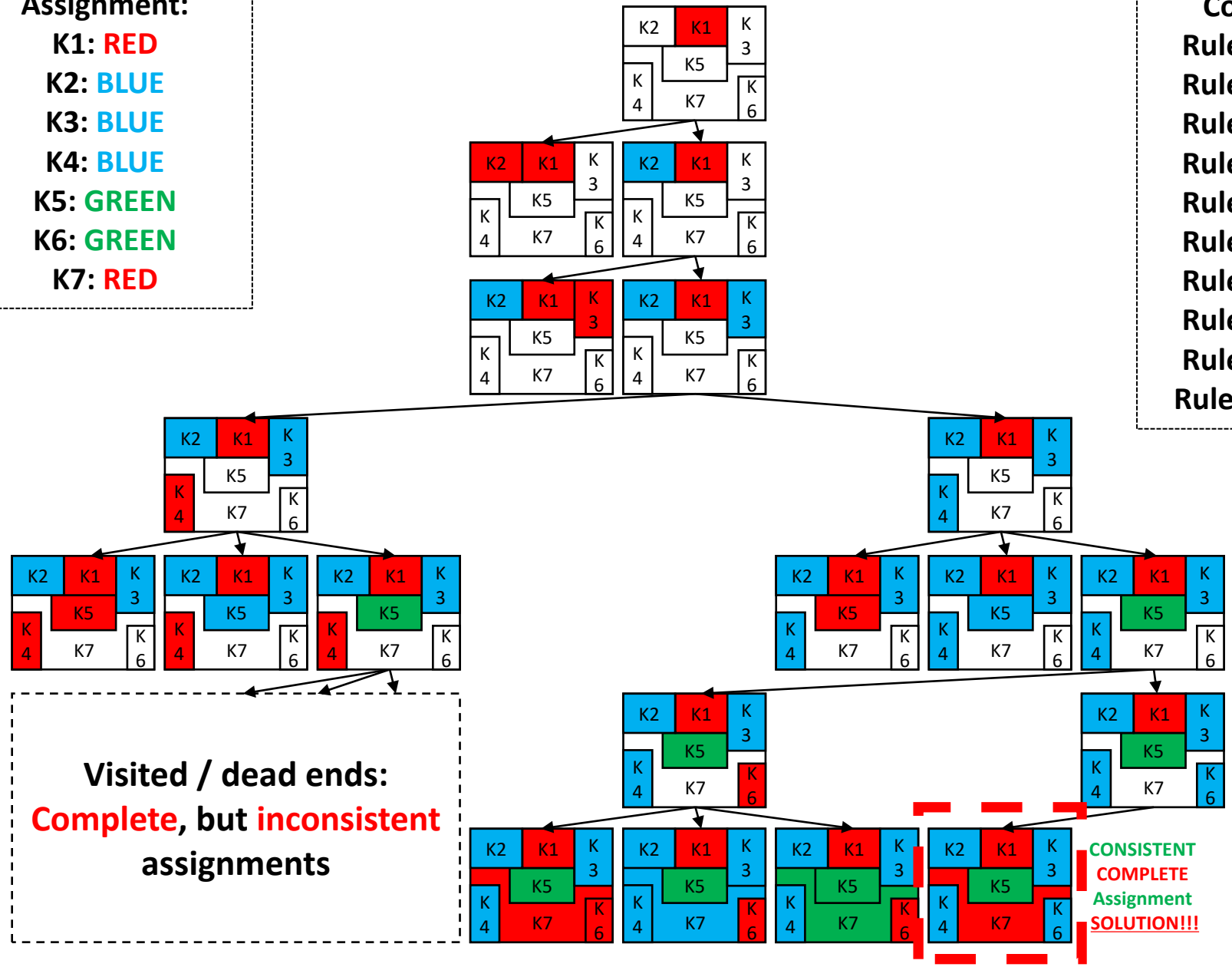


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Assignment:
K1: RED
K2: BLUE
K3: BLUE
K4: BLUE
K5: GREEN
K6: GREEN
K7: RED

Constraints:
Rule 1: K1 ≠ K2
Rule 2: K1 ≠ K3
Rule 3: K1 ≠ K5
Rule 4: K2 ≠ K5
Rule 5: K2 ≠ K7
Rule 6: K3 ≠ K5
Rule 7: K3 ≠ K7
Rule 8: K4 ≠ K7
Rule 9: K5 ≠ K7
Rule 10: K6 ≠ K7



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

Can We Do Better?

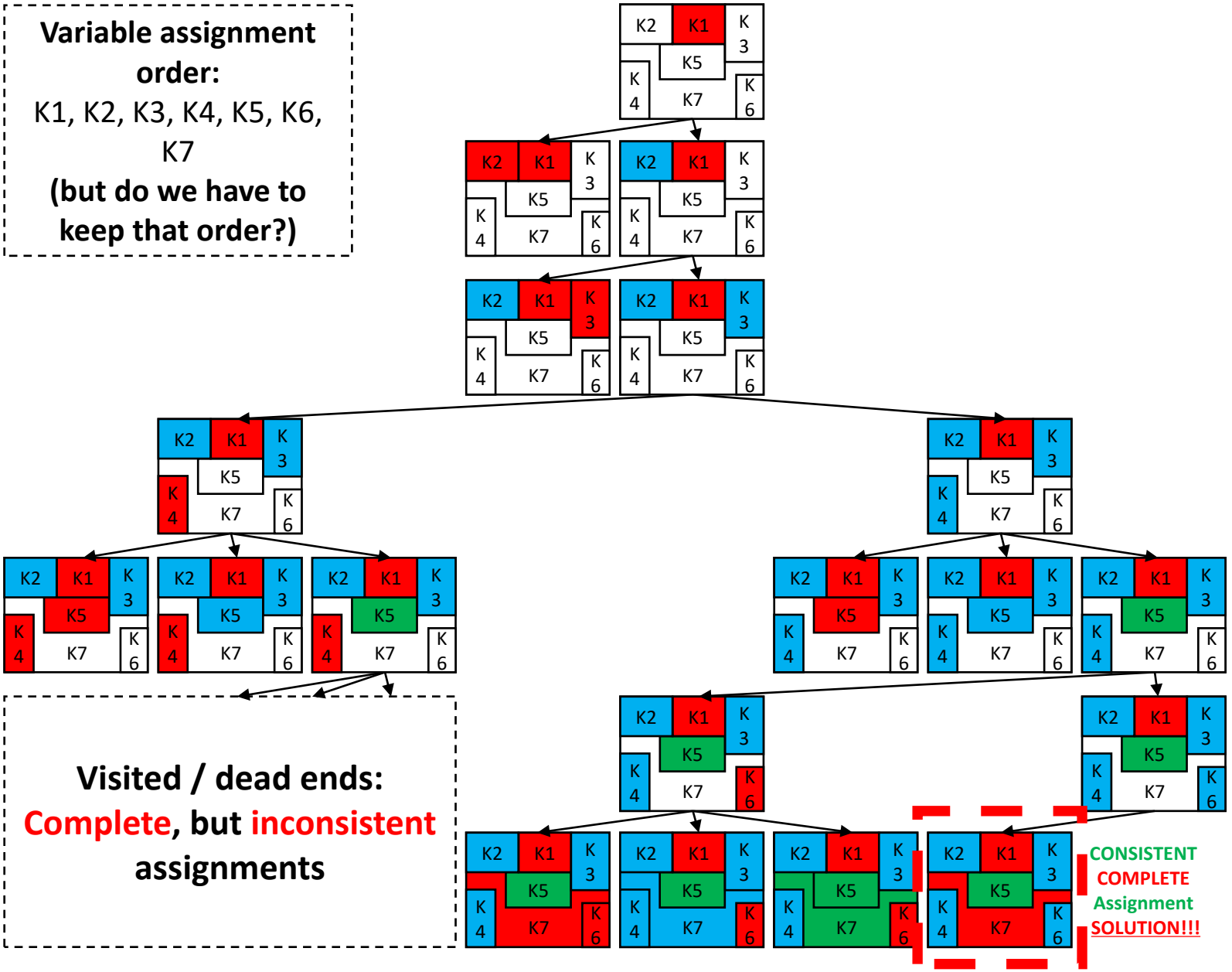
CSP Backtracking: Pseudocode

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
    return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
            remove {var = value} from assignment
    return failure
```

Which variable should we choose to assign a value to next?

Does it matter?

Variable assignment order:
K1, K2, K3, K4, K5, K6, K7
(but do we have to keep that order?)



Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7

Variable Ordering: Alternatives

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
    return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
            remove {var = value} from assignment
    return failure
```

You can modify this function to change the variable ordering and potentially improve performance

Variable Ordering: Alternatives

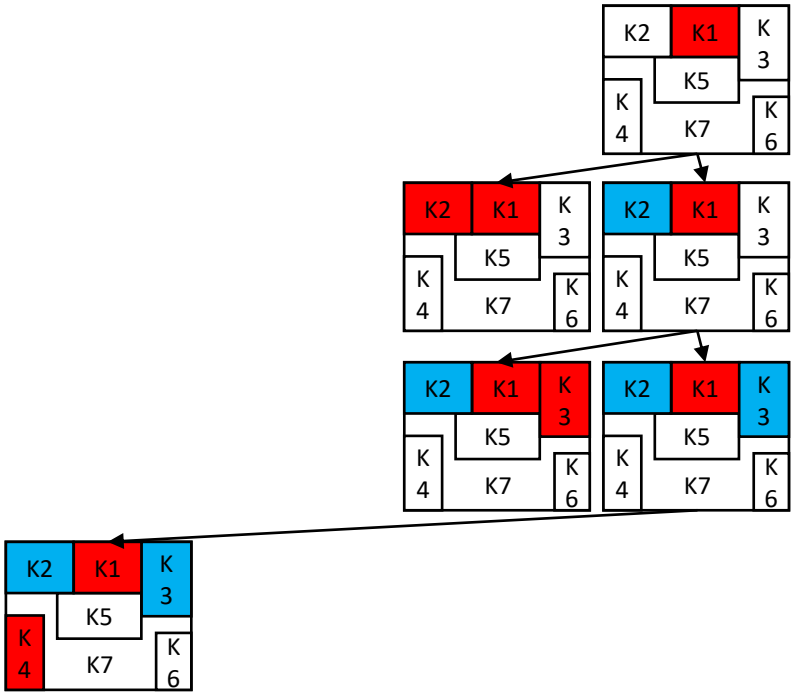
CSP Backtracking algorithm can use a number of variable ordering strategies:

- **Static: choose the variables in order (we did that)**
- **Random: order variables in random sequence**
- **Minimum-remaining-values (MRV) heuristic:**
  - **choose the variable with the "fewest" legal values**
- **Degree heuristic:**
  - **choose the variable involved in the largest amount of constraints on other unassigned variables**
  - **choose the variable with highest node degree on a constraint graph**

Variable Ordering: MRV Heuristic

As CSP Backtracking algorithm progresses, the number of possible value assignments for each variable will shrink (due to constraints):

- **MRV uses "fail-first" heuristics (also called "most constrained variable" heuristics)**
- **MRV picks a variable with lowest value assignment options "left"**
  - **expecting to limit exploration depth**
  - **likely to find a failure assignment faster**
- **Usually better than static and random orderings on average**



K1 = ???

K2 = ???

K3 = ???

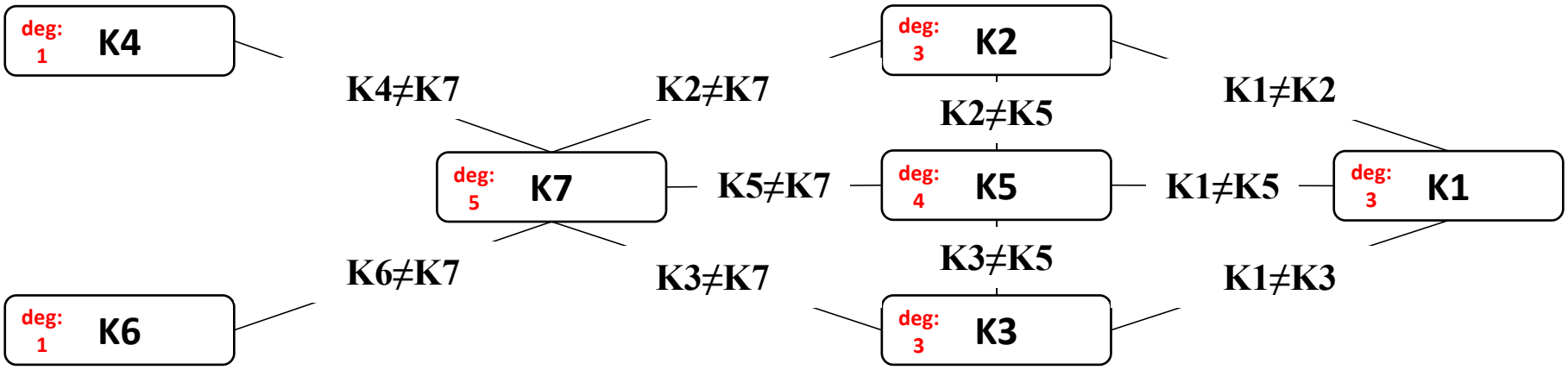
K4 = ???

Which variable to explore next (ignore the EXPECTED sequence on the right)?
Available options:
K5: {GREEN}
K6: {RED, BLUE, GREEN}
K7: {GREEN}

MRV should pick K5 or K7 ("fail first" variable).
Tie needs to be resolved.

Variable Ordering: Degree Heuristics

Consider the following constraint graph representation of the problem we analyzed:



- degree heuristics is considered less effective than MRV
- degree heuristics can be used as a tie-breaker (two variables with the same "potential" according to MRV)
- attempts to reduce the branching factor on future choices

Value Ordering: Alternatives

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

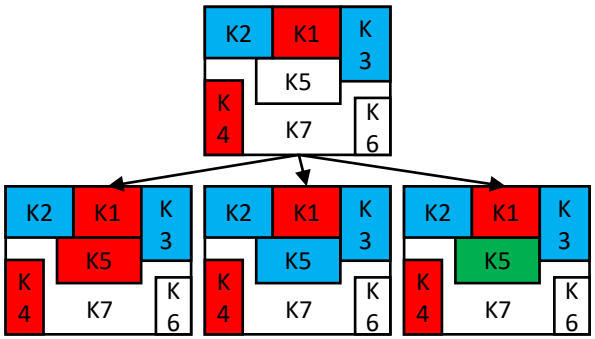
You can modify this order to change the value assignment ordering and potentially improve performance

Least-Constraining-Value Heuristics

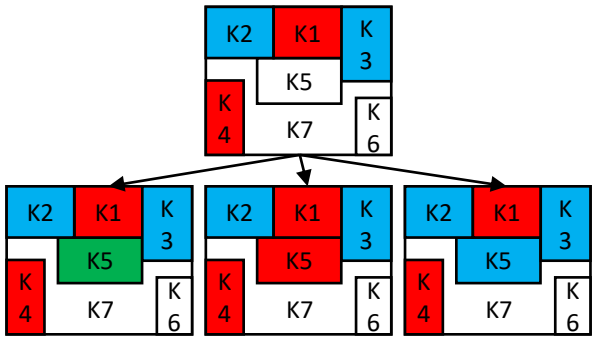
We picked (SELECT-UNASSIGNED-VARIABLE) the next variable to assign a value to and we have a number of values to choose from. What next?

- use the least-constraining-value heuristic
  - picks a value that rules out the fewest choices for neighboring variables in the constraing graph (increase flexibility for FUTURE assignments)
  - ORDER-DOMAIN-VALUES is the function that orders values here

Next variable to assign a value to: K5. Three options available: {RED, BLUE, GREEN}. How to order them?



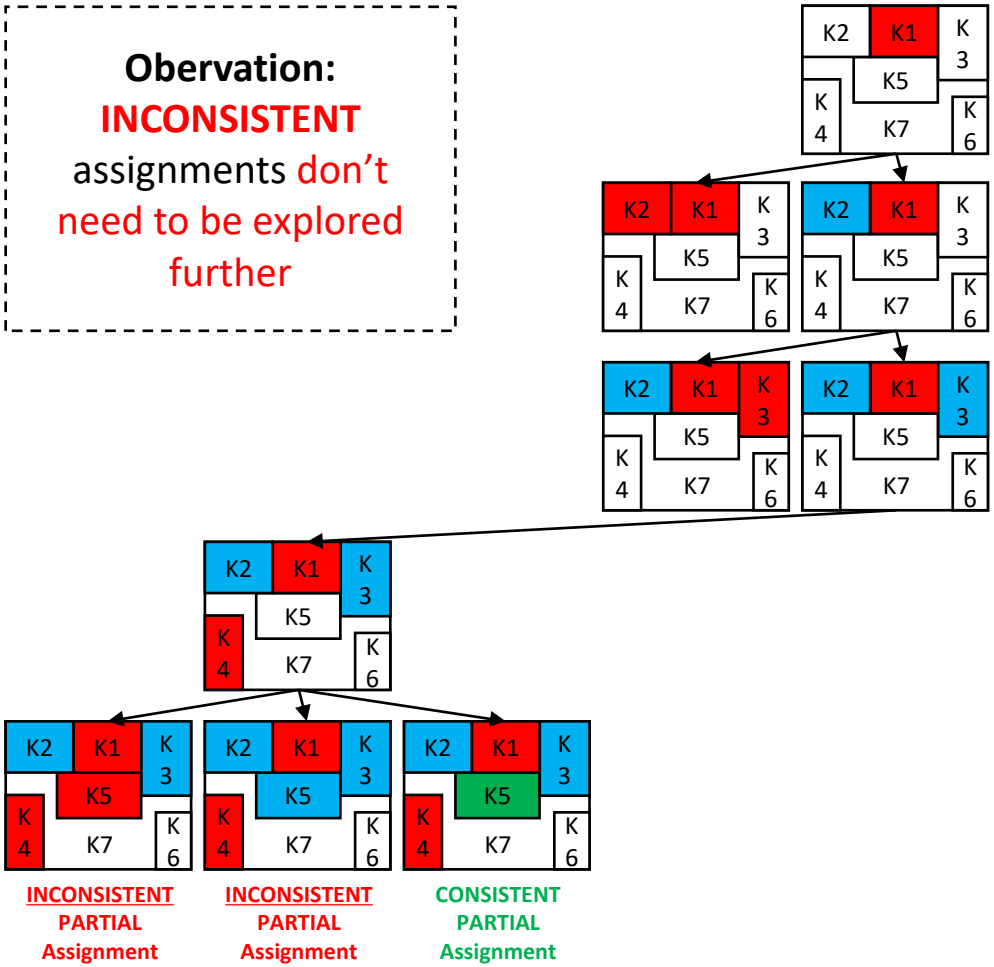
STATIC value ordering:
poor, best option is last



LEAST-CONSTRAINING-VALUE value ordering:
better, best option is first

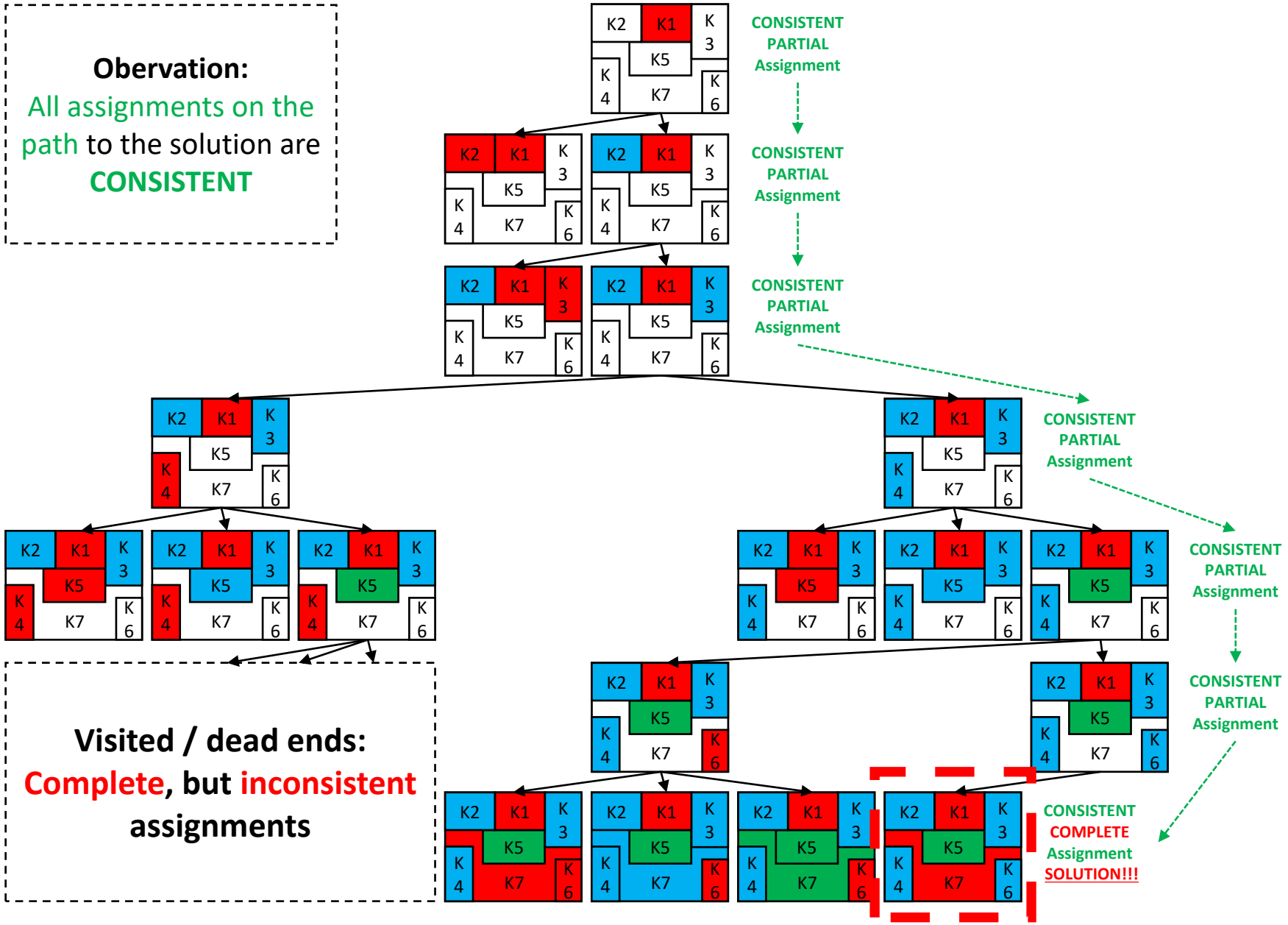
Can We Do Better?

Obervation:
INCONSISTENT assignments don't need to be explored further



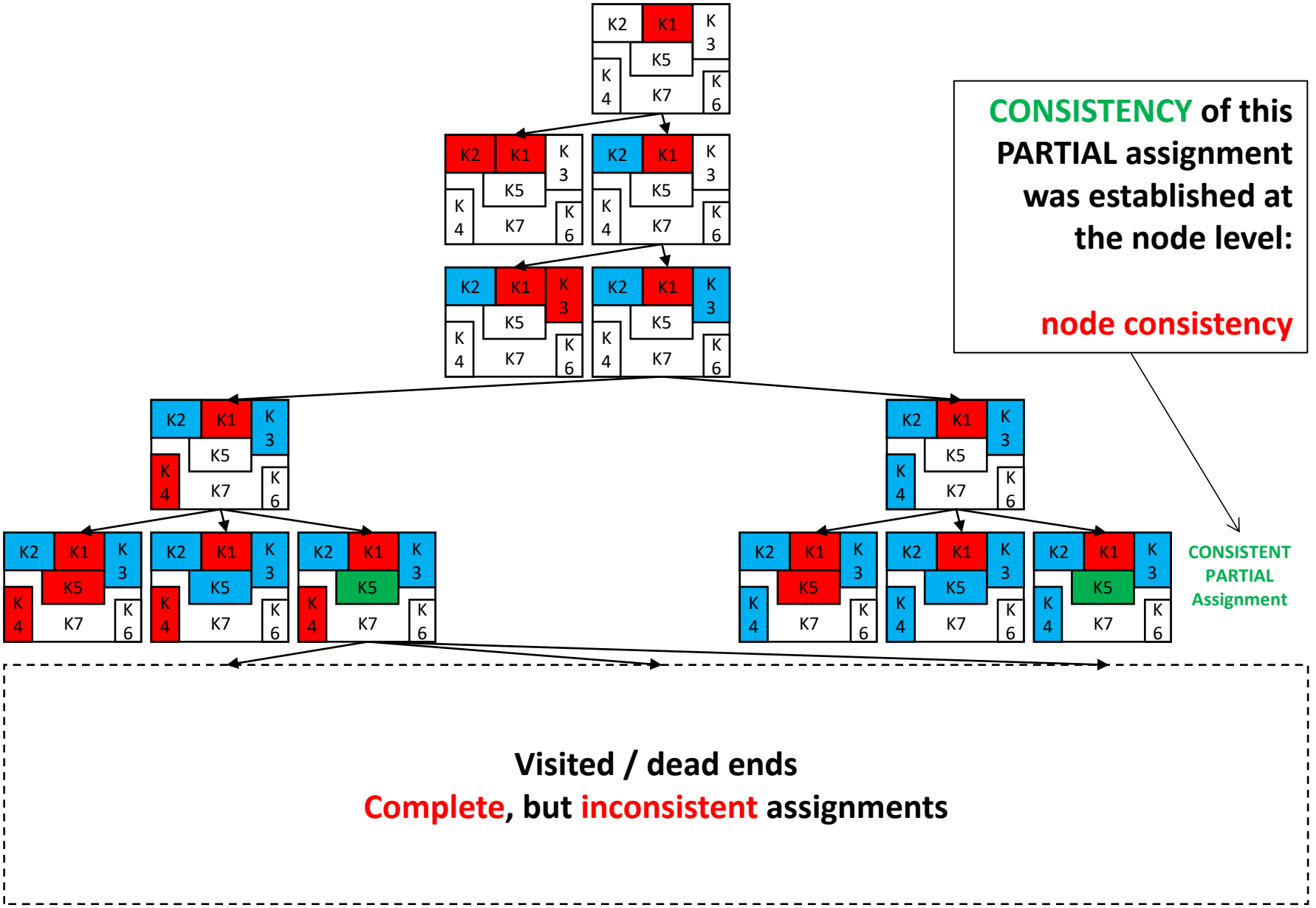
Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** **RED**, **BLUE**, **GREEN**

Obervation:
All assignments on the path to the solution are **CONSISTENT**

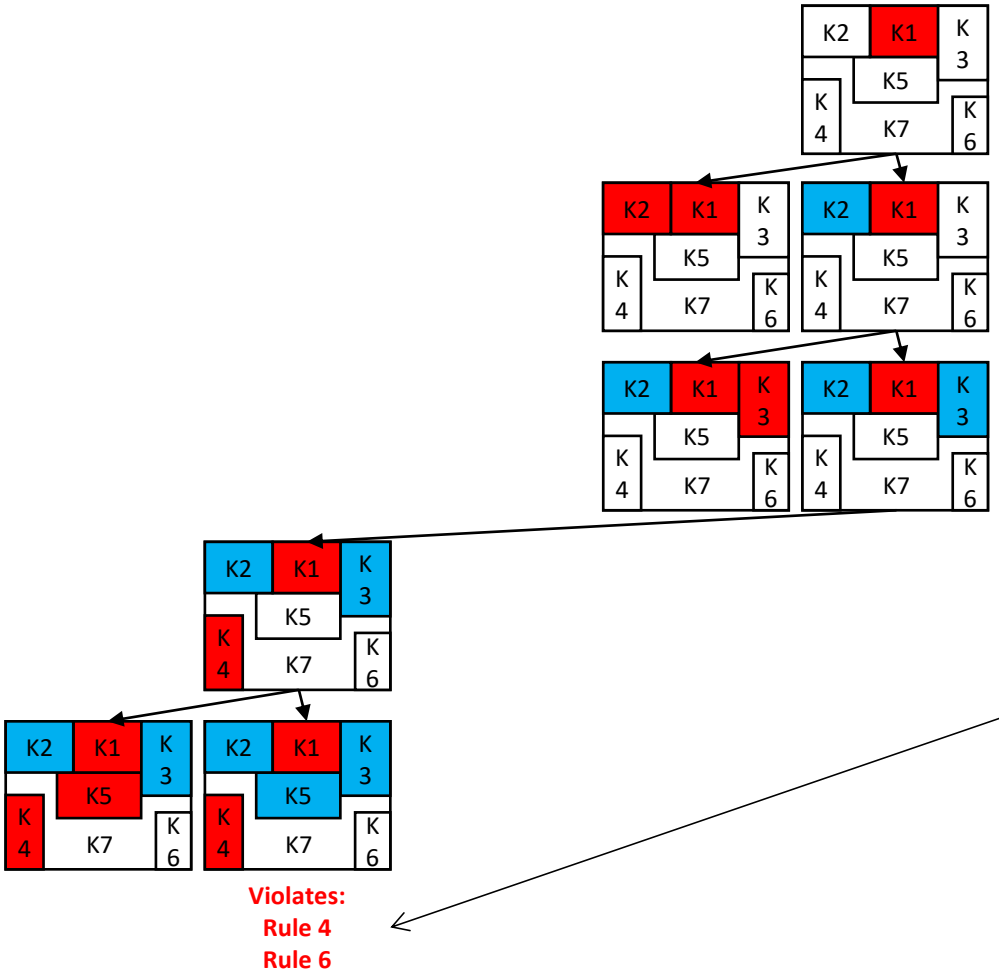


Visited / dead ends:
Complete, but inconsistent assignments

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN



Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN



INCONSISTENCY of this PARTIAL assignment was established at the node level:

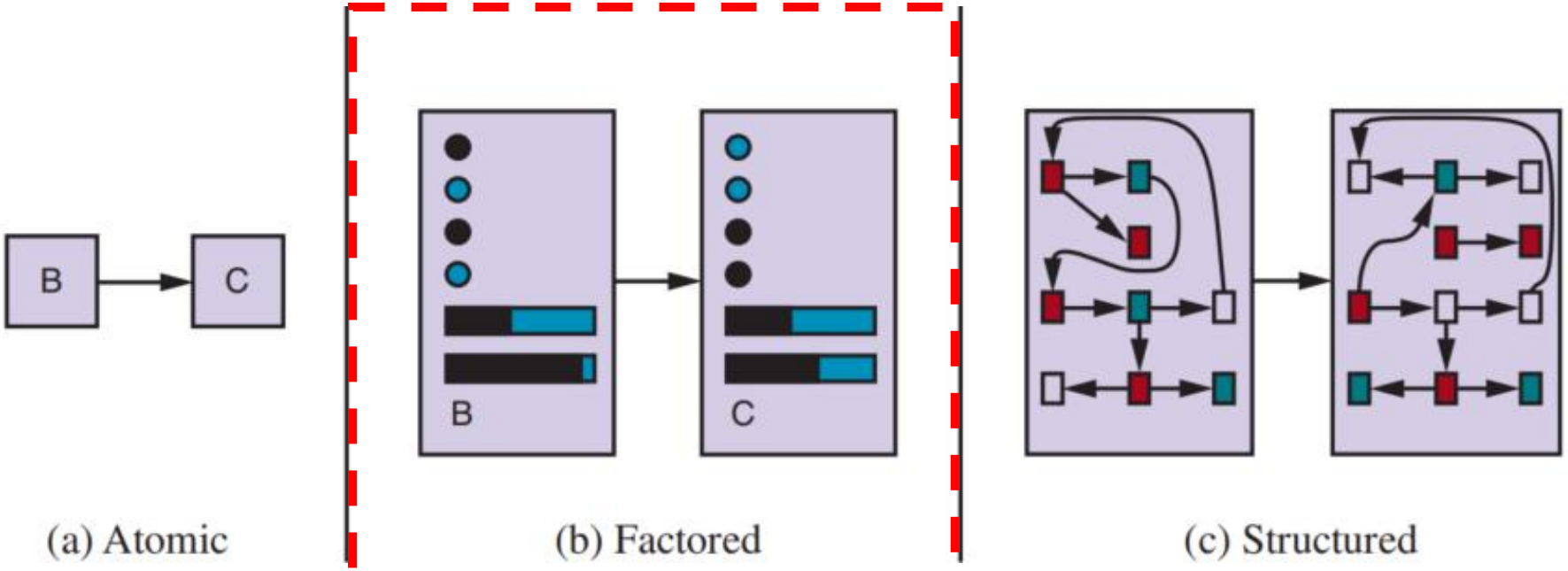
node consistency

Can we detect / "predict" INCONSISTENCIES earlier and prune useless branches?

YES!

Variable assignment order: K1, K2, K3, K4, K5, K6, K7 | **Value assignment order:** RED, BLUE, GREEN

How CSP Can Reduce Work



Next move?

- **Expand the node and visit succesors**

Next move?

- **Expand the node (assign value to a variable) and visit successors**
- **Infer where to go from current assignment and constraints (constraint propagation)**

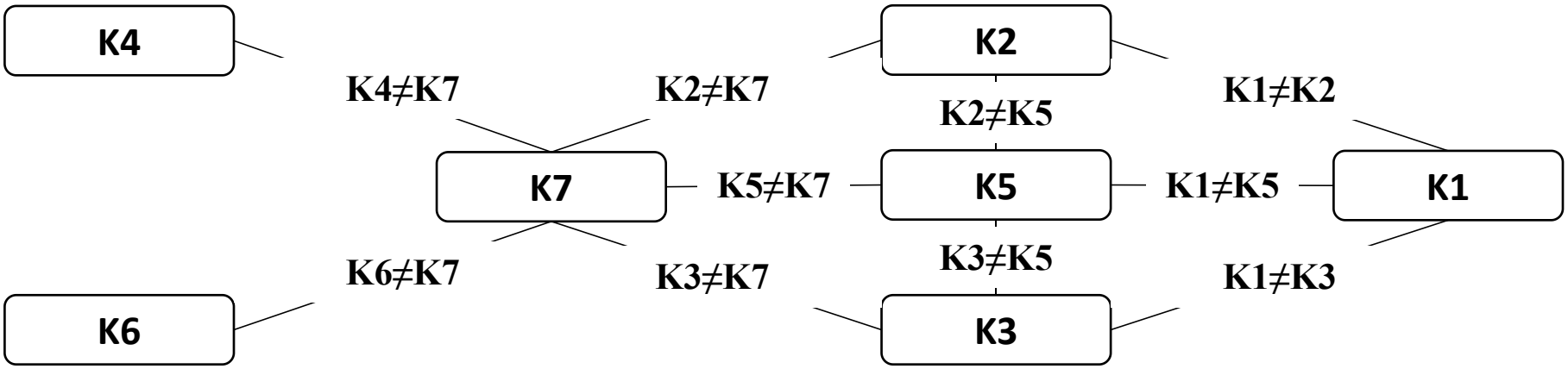
CSP: More Pruning with Inference

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, assignment)
      if inferences ≠ failure then
        add inferences to csp
        result ← BACKTRACK(csp, assignment)
        if result ≠ failure then return result
        remove inferences from csp
      remove {var = value} from assignment
  return failure
```

With the information available to you, you can INFER that a particular branch is going to be INCONSISTENT

Inference in CSP

- **Simplifying the problem:**
  - **preprocessing / pre-check or part of the search**
  - **it can reduce the problem OR even solve it**
- **Inference with Constraint Propagation:**
  - **use constraint graph to enforce consistency locally**

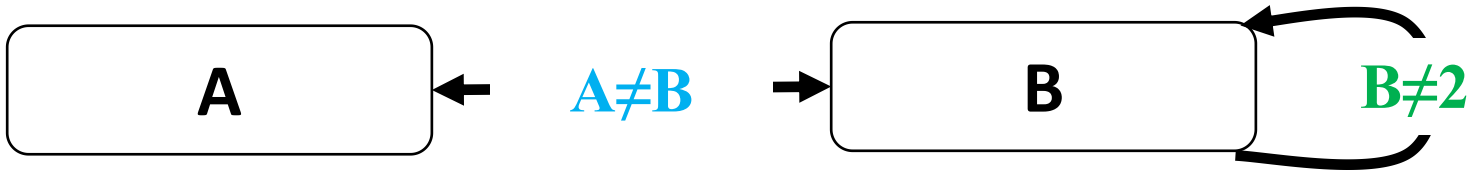


Local Consistency

- The idea:
  - remove inconsistent values from variable domains as we go as they would make certain assignments inconsistent later anyway
- Types:
  - Node consistency
  - Arc consistency (or edge consistency)
  - Path consistency

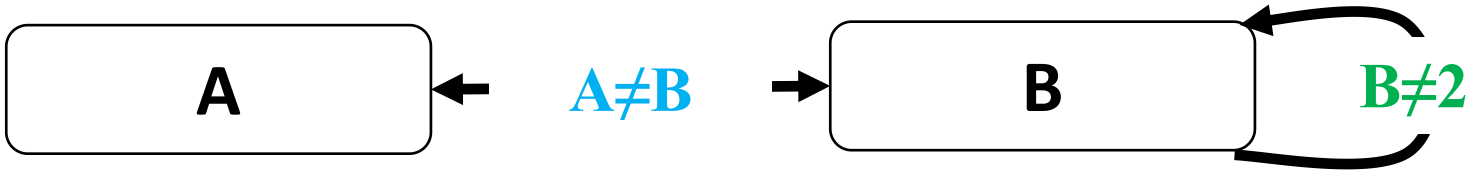
Node Consistency

- **Consider the following CSP example:**
  - **variables:** $X = \{A, B\}$
  - **domains:**
    - $D_A = \{0, 1, 3\}$
    - $D_B = \{2, 3, 4\}$
  - **constraints:** $C = \{A \neq B, B \neq 2\}$
    - **one binary and one unary constraint**
  - **constraint graph:**



Node Consistency

- The idea:
  - a single variable is node-consistent (in a constraint graph) if all the values in its domain satisfy variable unary constraints
- (Constraint) graph is node-consistent if every variable in the graph is node-consistent

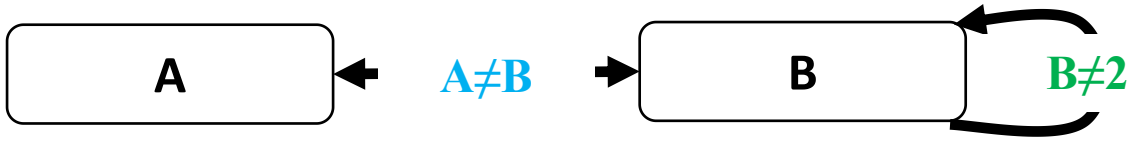


Variable B is NOT node-consistent because in $D_B = \{2,3,4\}$ value 2 does not satisfy unary B≠2

- Approach: remove unary constraints by reducing variable domain

Node Consistency

- **Unary constraints can easily be removed to reduce the problem:**
  - **BEFORE (unary constraint removal) domains:**
    - $D_A = \{0, 1, 3\}$
    - $D_B = \{2, 3, 4\}$



Constraint graph is NOT node-consistent because of variable B

  - **AFTER (unary constraint removal) domains:**
    - $D_A = \{0, 1, 3\}$
    - $D_B = \{3, 4\}$

Constraint graph is node-consistent

Arc (Edge) Consistency

- The idea:
  - a single variable is arc-consistent (in a constraint graph) if all the values in its domains satisfy ALL its binary constraints
- (Constraint) graph is arc-consistent if every variable in the graph is arc-consistent

Variables A and B are NOT arc-consistent because in $D_A = \{1,2,3\}$ and $D_B = \{3,4\}$ value 3 clashes

- Approach: reducing variable domains to remove clashes

Arc (Edge) Consistency

- **Values that clash can be removed from variable domains to reduce the problem:**
  - **BEFORE (clashing value(s) removal) domains:**
    - $D_A = \{0, 1, 3\}$
    - $D_B = \{3, 4\}$

Constraint graph is NOT arc-consistent because of value 3 clashing in both domains

  - **AFTER (clashing value(s) removal) domains:**
    - $D_A = \{0, 1, 3\}$
    - $D_B = \{4\}$ or
    - $D_A = \{0, 1\}$
    - $D_B = \{3, 4\}$ (depends on: which variable we start with)

Constraint graph is arc-consistent

AC-3 Algorithm: Pseudocode

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise

  queue ← a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**

    $(X_i, X_j) \leftarrow$ POP(*queue*)

    if REVISE(*csp*, X_i, X_j) **then**

      if size of $D_i = 0$ **then return** *false*

      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**

        add (X_k, X_i) to *queue*

  return *true*

Note: treat a constraint graph edge as two directional edges: constraint $X_i \neq X_j$ corresponds to edges (X_i, X_j) and (X_j, X_i)

function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i

  revised ← *false*

  for each x **in** D_i **do**

    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**

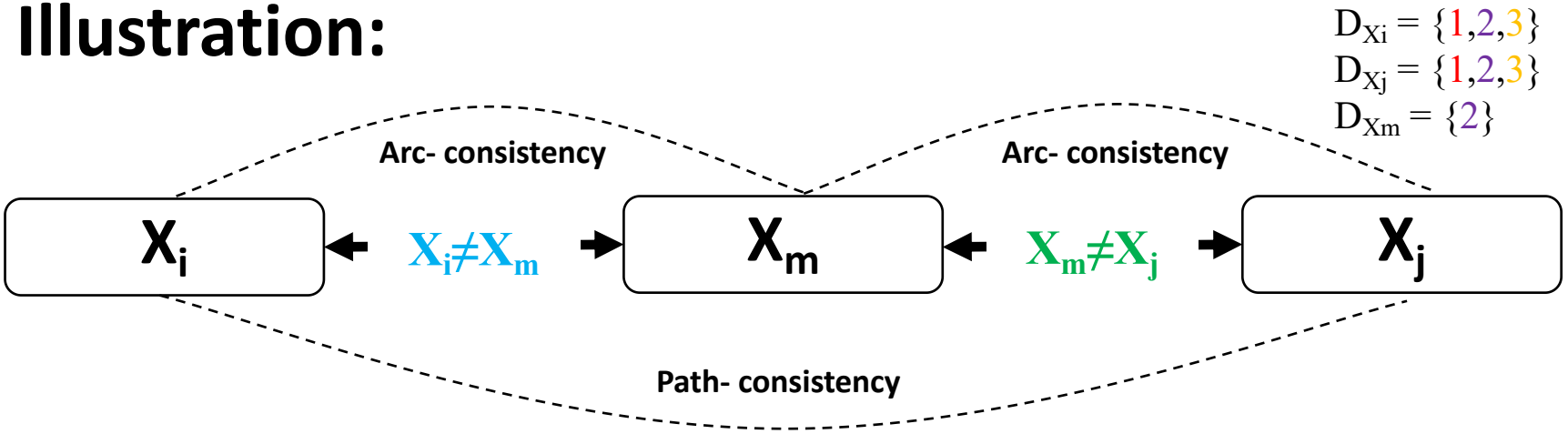
      delete x from D_i

      revised ← *true*

  return *revised*

Path Consistency

- **The idea:**
  - **two variable set $\{X_i, X_j\}$ is path-consistent (in a constraint graph) with respect to a third variable X_m if for EVERY assignment $\{X_i = a, X_j = b\}$ there is an assignment to X_m (between X_i and X_j) that satisfies constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$.**
- **Illustration:**



Searching with Inference

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
    return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
            remove {var = value} from assignment
    return failure
```

Apply local consistency checks and report failure if you know that following given path is going to dead end

Searching with Inference

Two key ideas:

- **Forward checking**
- **Maintaining Arc Consistency**