Booleans/Signature

Let us start this section about search algorithms by presenting how to specify data structures without relying on a specific programming language. The method we introduce here is usually called **algebraic specification**.

Consider one of the simplest data structure you can imagine: the booleans. Let us call the following specification Bool. Here is its **signature**:

- **Defined types**
  - The type of the booleans is t.
- **Constructors**
  - True : t
  - False : t

Booleans/Signature (continued)

$\textsc{True}$ and $\textsc{False}$ are called constructors because they allow the construction of **values** of type t, i.e. booleans. This is why we write ": t" after these constructors.

For more excitement, let us add some usual **functions** whose arguments are booleans:

- $\textsc{Not} : \mathsf{t} \to \mathsf{t}$
  Expression $\textsc{Not}(b)$ is the negation of b;
- $\textsc{And} : \mathsf{t} \times \mathsf{t} \to \mathsf{t}$
  Expression $\textsc{And}(b_1, b_2)$ is the conjunction of b_1 and b_2;
- $\textsc{Or} : \mathsf{t} \times \mathsf{t} \to \mathsf{t}$
  Expression $\textsc{Or}(b_1, b_2)$ is the disjunction of b_1 and b_2.

Booleans/Equations

We can give mode information about the previous functions by means of **equations** (or **axioms**). A possible set of equations matching the signature is

$$
\begin{aligned}
\textsc{Not}(\textsc{True}) &= \textsc{False} \\
\textsc{Not}(\textsc{False}) &= \textsc{True} \\
\textsc{And}(\textsc{True}, \textsc{True}) &= \textsc{True} \\
\textsc{And}(\textsc{True}, \textsc{False}) &= \textsc{False} \\
\textsc{And}(\textsc{False}, \textsc{True}) &= \textsc{False} \\
\textsc{And}(\textsc{False}, \textsc{False}) &= \textsc{False} \\
\forall\, b_1, b_2 \quad \textsc{Or}(b_1, b_2) &= \textsc{Not}(\textsc{And}(\textsc{Not}(b_1), \textsc{Not}(b_2)))
\end{aligned}
$$

The signature and the equations make a **specification**.

Booleans/Equations (cont)

We note something interesting in the last equation: it contains **variables**, here b_1 and b_2. These variables must be of type t because they are arguments of function $\textsc{Or}$, whose type, as given by the signature, is $\mathrm{t} \times \mathrm{t} \to \mathrm{t}$.

It is very important to notice also that these variables are bound to a **universal quantifier**, $\forall$, at the beginning of the equation. This means, in particular, that we can rename these variables because they are just names which are local to the equation. For instance

$$\forall\, u, v \quad \textsc{Or}(u, v) = \textsc{Not}(\textsc{And}(\textsc{Not}(u), \textsc{Not}(v)))$$

would be equivalent.

Booleans/Simplifying the equations

We can simplify these equations before going on.

First we can omit the quantifiers (but remember they are implicitly present).

Second we remark that it suffices that one argument of $\textsc{And}$ is $\textsc{False}$ to make the call equal to $\textsc{False}$. In other words

$$\begin{aligned}
\textsc{Not}(\textsc{True}) &= \textsc{False} \\
\textsc{Not}(\textsc{False}) &= \textsc{True} \\
\textsc{And}(\textsc{True}, \textsc{True}) &= \textsc{True} \\
\textsc{And}(x, \textsc{False}) &= \textsc{False} \\
\textsc{And}(\textsc{False}, x) &= \textsc{False} \\
\textsc{Or}(b_1, b_2) &= \textsc{Not}(\textsc{And}(\textsc{Not}(b_1), \textsc{Not}(b_2)))
\end{aligned}$$

Booleans/Simplifying the equations (cont)

In order to be 100% confident, we must check whether the set of new equations is **equivalent** to the first set.

$$\begin{cases} \textsc{And}(\textsc{True}, \textsc{False}) = \textsc{False} \\ \textsc{And}(\textsc{False}, \textsc{True}) = \textsc{False} \\ \textsc{And}(\textsc{False}, \textsc{False}) = \textsc{False} \end{cases} \overset{?}{\iff} \begin{cases} \forall\, x & \textsc{And}(x, \textsc{False}) = \textsc{False} \\ \forall\, x & \textsc{And}(\textsc{False}, x) = \textsc{False} \end{cases}$$

The new system is equivalent to $\begin{cases} \textsc{And}(\textsc{True}, \textsc{False}) = \textsc{False} \\ \textsc{And}(\textsc{False}, \textsc{False}) = \textsc{False} \\ \textsc{And}(\textsc{False}, \textsc{True}) = \textsc{False} \\ \textsc{And}(\textsc{False}, \textsc{False}) = \textsc{False} \end{cases}$

So the answer is yes.