Temporal Specification

[Models and Programs]

Michael Fisher

Department of Computer Science, University of Liverpool, UK

[MFisher@liverpool.ac.uk]

An Introduction to Practical Formal Methods Using Temporal Logic

Plan of Action

1. Look at temporal specifications and visualise the execution sequences they characterise.
2. Examine temporal specification of individual elements:
   (a) syntax of an imperative programming language;
   (b) temporal semantics of the language;
   (c) example program and its temporal semantics.
3. Consider *refinement* of, and *concurrency* within, temporal specifications and semantics.
4. Linking Temporal Specifications via *message passing*.
5. Linking Temporal Specifications *(other approaches and possible difficulties).*

Interpreting Temporal Formulae

When we look at a temporal specification, the possible executions of the program being specified correspond to the possible models for the specification.

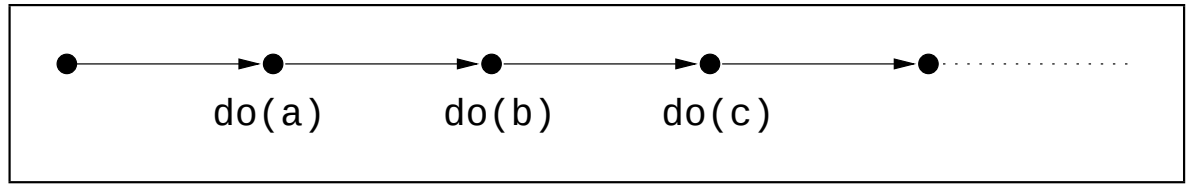
For example, if we were able to write a program such as

```
do(a); do(b); do(c)
```

then we might expect the temporal specification for such a program to be

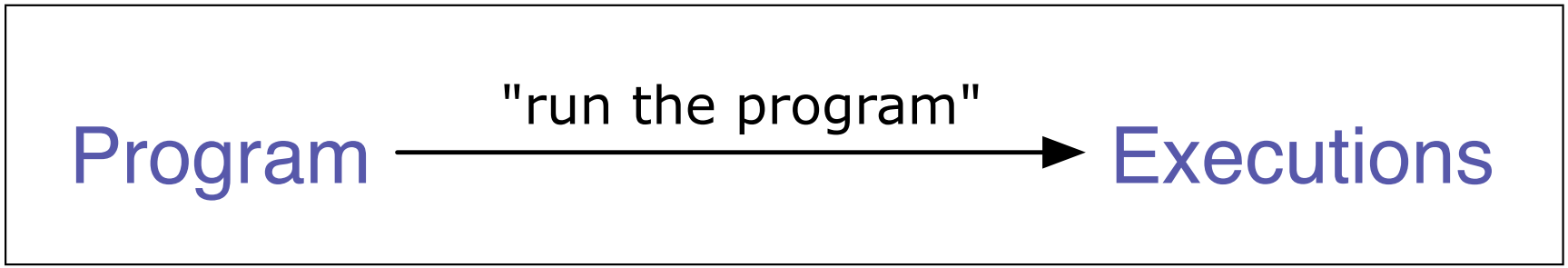
$$\mathbf{start} \Rightarrow \bigcirc do(a) \wedge \bigcirc\bigcirc do(b) \wedge \bigcirc\bigcirc\bigcirc do(c)$$

The models for this formula should look like

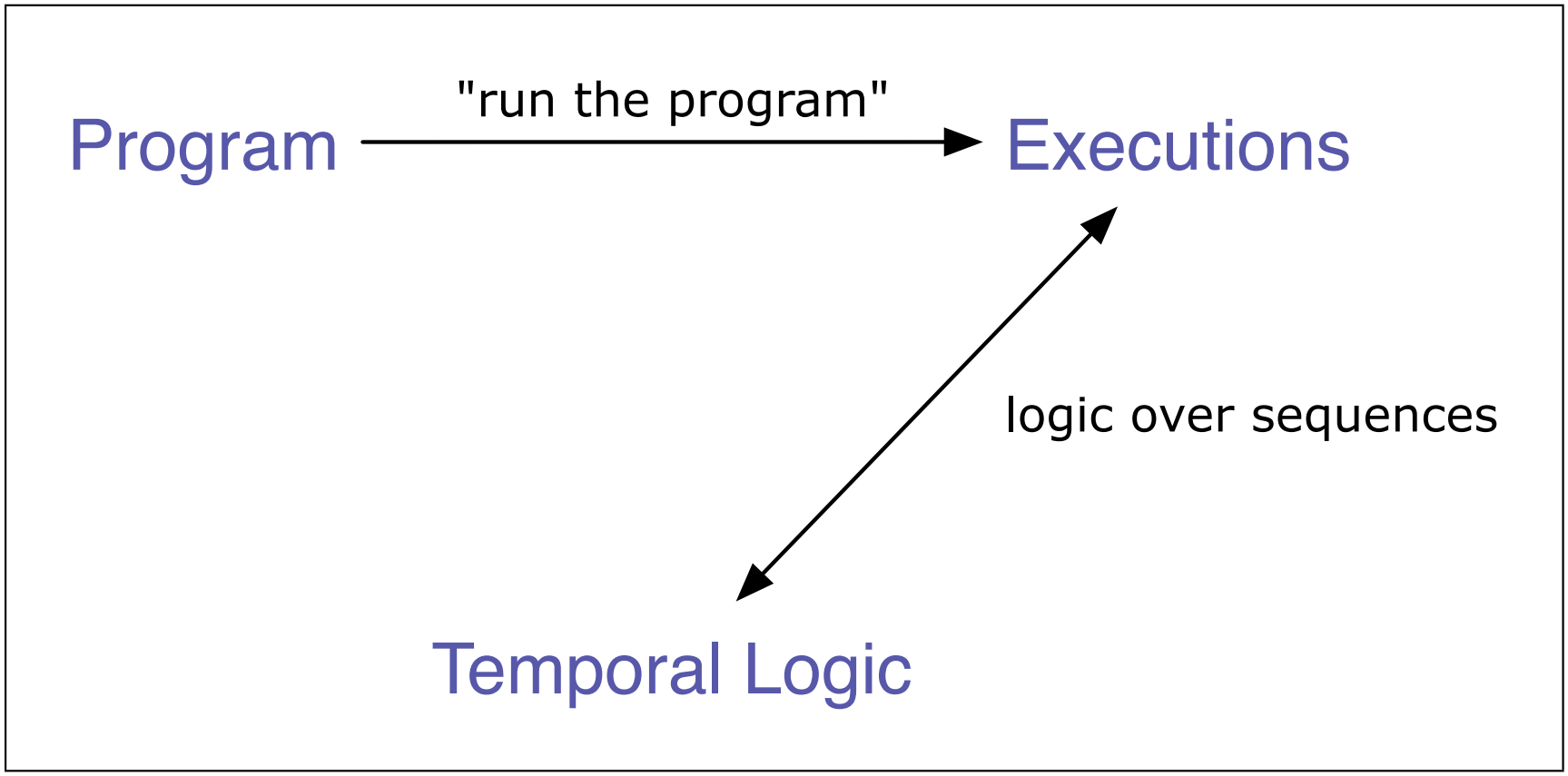


corresponding to potential executions of the program.

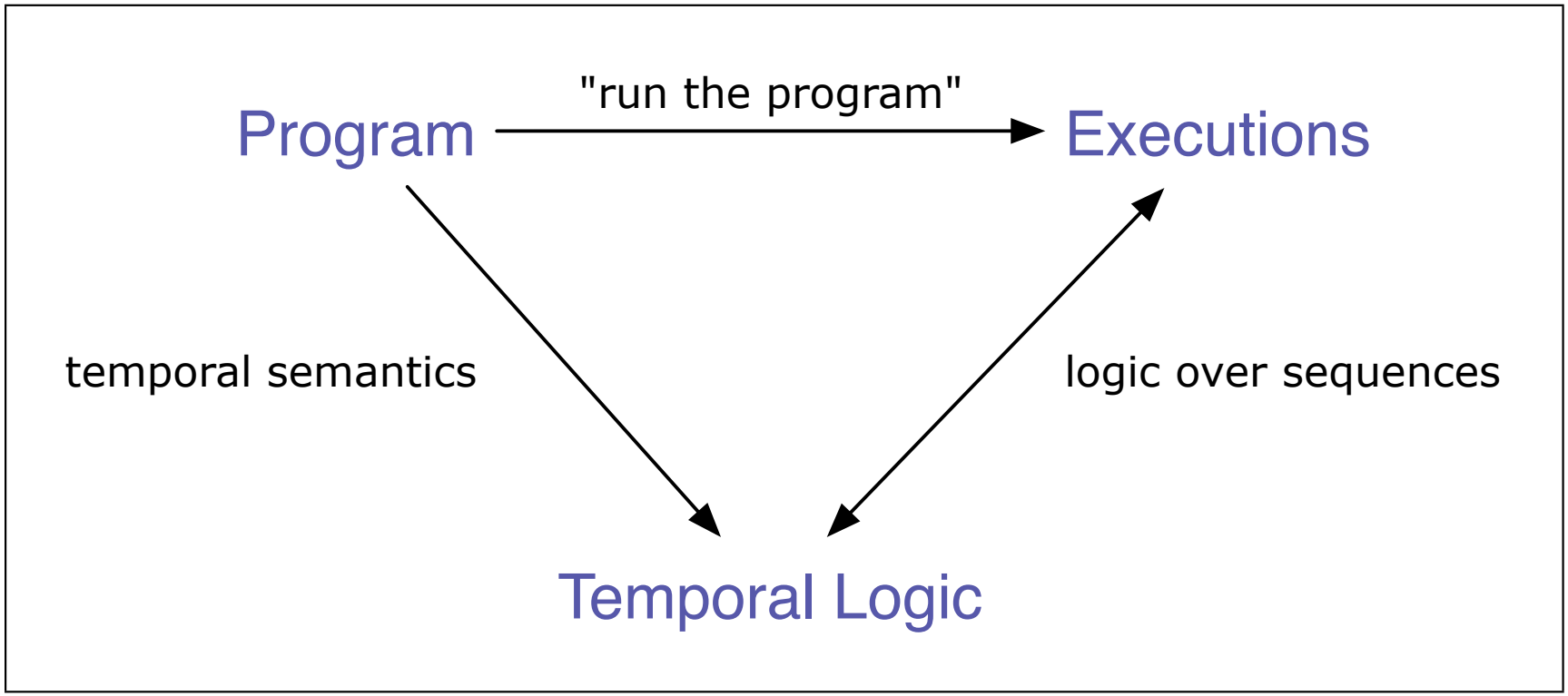
Intuition (1)



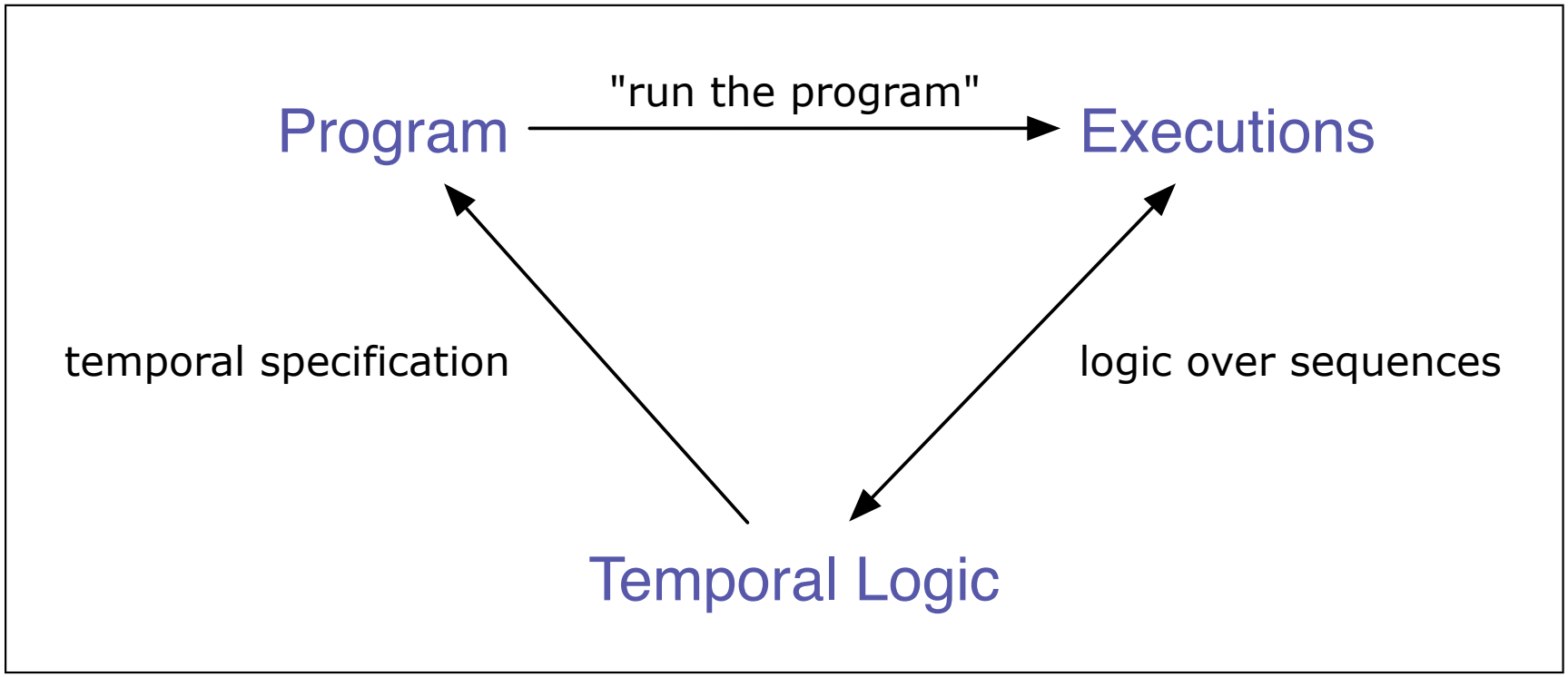
Intuition (2)


Program
"run the program"
Executions
logic over sequences
Temporal Logic

Intuition (3)


Program
"run the program"
Executions
temporal semantics
logic over sequences
Temporal Logic

Intuition (4)



So: it is our intention that temporal models correspond to potential executions of the program.

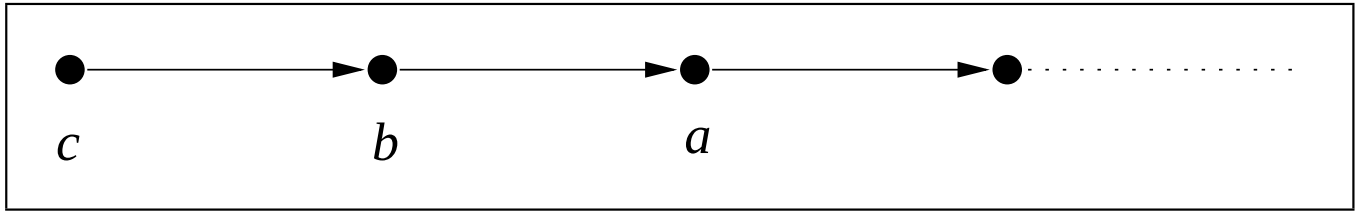
But: can temporal specifications capture all *reasonable* patterns of program behaviour?

Temporal Specifications (1)

We will look as a specific programming language later, but now consider an abstract specification such as

$$\Box \left[\begin{array}{rrl} & \mathbf{start} \Rightarrow & c \\ \wedge & (c \Rightarrow & \bigcirc b) \\ \wedge & (b \Rightarrow & \bigcirc a) \end{array} \right]$$

Typical models for this formula look like

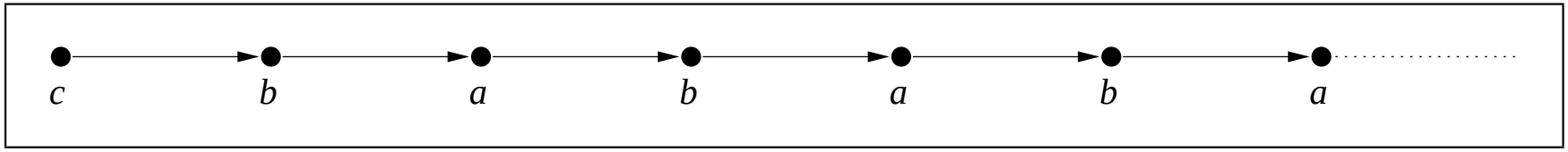


Temporal Specifications (2)

Now, if we extend the specification (omitting '$\wedge$') to

$$\Box \left[\begin{array}{rcl} \mathbf{start} & \Rightarrow & c \\ c & \Rightarrow & \bigcirc b \\ b & \Rightarrow & \bigcirc a \\ a & \Rightarrow & \bigcirc b \end{array} \right]$$

we get the infinite cyclic behaviour:



Temporal Specifications (3)

Consider a slightly more complex example:

$$\Box \left[\begin{array}{rcl} \mathbf{start} & \Rightarrow & a \\ \mathbf{start} & \Rightarrow & b \\ (a \wedge b) & \Rightarrow & \bigcirc (c \vee d) \\ c & \Rightarrow & \bigcirc c \\ d & \Rightarrow & \bigcirc e \end{array} \right]$$

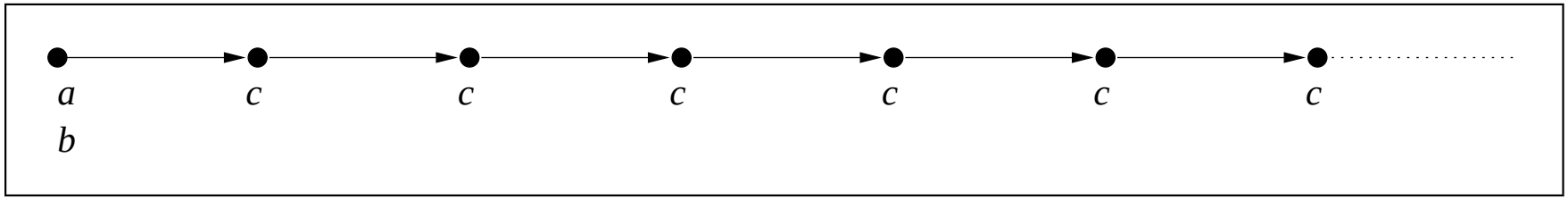
Here, the key formula is: $(a \wedge b) \Rightarrow \bigcirc (c \vee d)$

which says that if a and b are both true then, in the next moment, there is a *choice* of making at least c or d true.

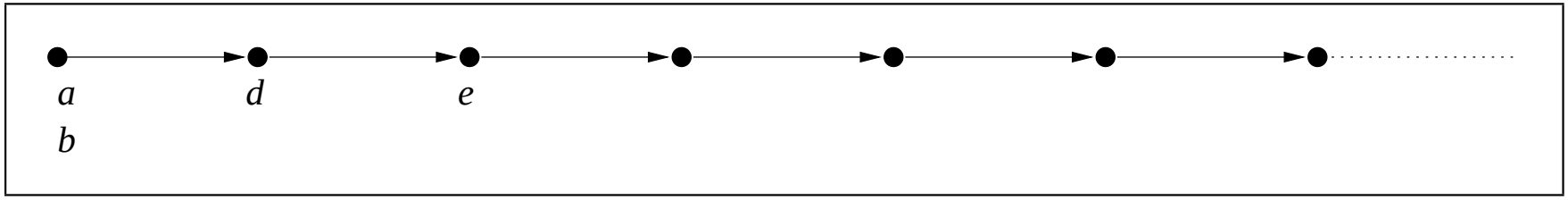
Note that this is a *non-deterministic* choice.

Models for Last Example

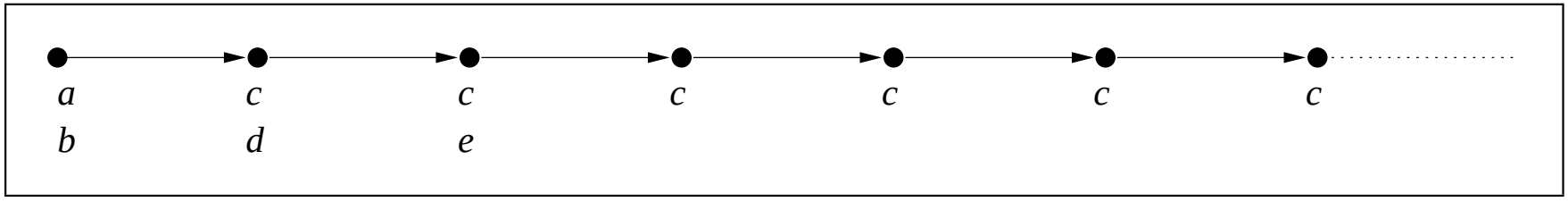
Thus, there are several possibilities, one being that we choose to make c true:



another being that we choose to make d true:



with a third being that we make *both* c and d true:

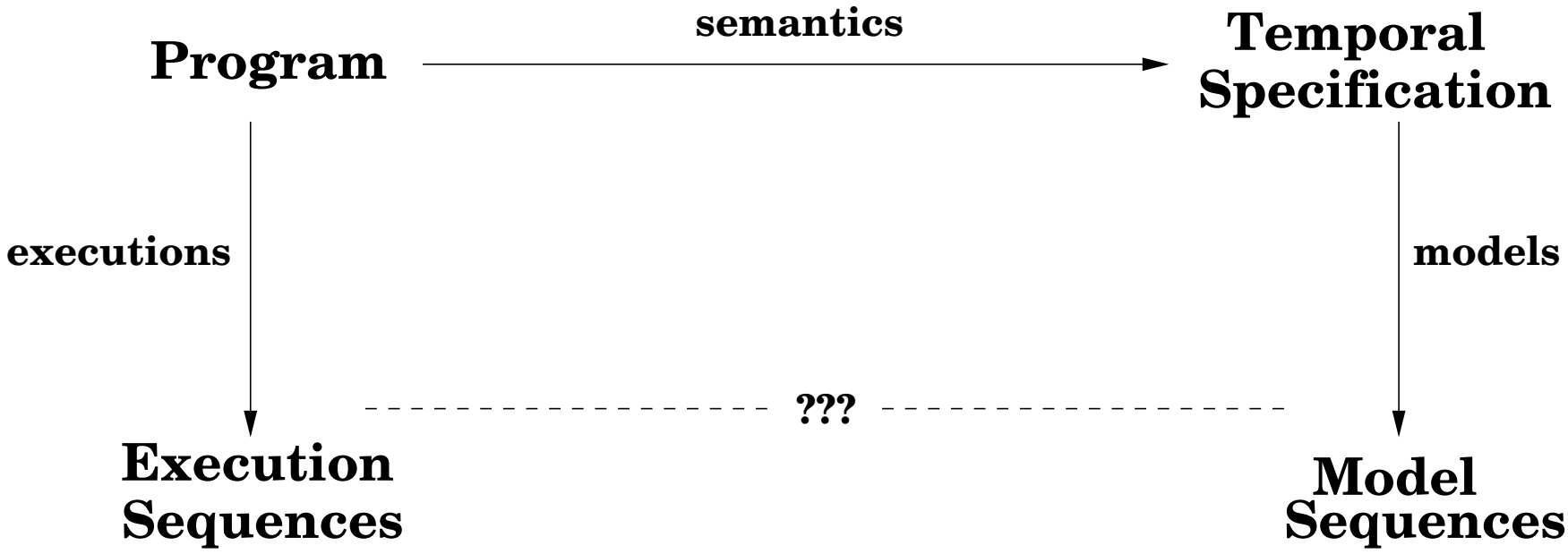


Exercise

What pattern of behaviour does the following specify?

$$\Box \left[\begin{array}{rcl} \mathbf{start} & \Rightarrow & x \\ x & \Rightarrow & \bigcirc (w \wedge y \wedge z) \\ z & \Rightarrow & \bigcirc a \\ (z \wedge y \wedge w) & \Rightarrow & \bigcirc b \\ (a \wedge b) & \Rightarrow & \bigcirc x \end{array} \right]$$

Overview


Program
semantics
Temporal Specification
executions
models
???
Execution Sequences
Model Sequences

Representing Programs

In general, the purpose of a semantics is to assign some 'meaning' to each statement within a particular language.

$$\textit{formal}\ \text{semantics} \longrightarrow \textit{formal}\ \text{meaning}$$

There are many different forms of formal semantics, e.g: *denotational*; *operational*; *temporal*; etc.

We wish to use a temporal semantics to produce a temporal logic formula specifying the behaviour of a program.

To show how such a semantics works, we will define a function

$$[\![\_]\!] : \mathit{Prog} \mapsto \mathit{PTL}$$

which provides a temporal formula for each program within our target programming language.

Boolean Assignments

Imagine we have a minimal programming language with just

`x:=v`
. . . *after this statement, Boolean variable* x *has the value* v

`S1; S2` . . . . . *execute statement* S2 *after executing* S1

We can define a temporal semantics through a semantic function '⟦ ⟧' that maps program fragments on to their (temporal) meaning. So, for a program P, then

$$[\![P]\!]$$

is a temporal formula.

Semantics of Boolean Assignment

Let us begin with simple assignment to Boolean variables:

$$[\![\texttt{x:=true}]\!] \quad \equiv \quad \bigcirc x$$

Thus, we model the assignment of '`true`' to the Boolean variable '`x`' by making the corresponding proposition 'x' true in the next state (i.e. after the assignment operation has taken place) within our temporal formula. Obviously:

$$[\![\texttt{x:=false}]\!] \quad \equiv \quad \bigcirc \neg x$$

Rather than considering such isolated program statements, we are more likely to have a program constructed through sequential composition, '`;`'. So, we would have:

$$[\![\texttt{x:=true; S}]\!] \quad \equiv \quad \bigcirc (x \wedge \bigcirc [\![\texttt{S}]\!])$$

Aside: Concurrency

We will see later that we can also tackle more complex program constructs.

However, we will now just mention one appealing feature of a temporal semantics, namely the representation of parallel activities.

So, (synchronous) parallel statements (i.e. two things happening at once; in this case, 'S' and 'T') might simply be modelled by

$$[\![S \parallel T]\!] \quad \equiv \quad [\![S]\!] \wedge [\![T]\!]$$

This simplicity is one of the appealing features of using a temporal logic.

Example: Concurrency (1)

We can use this approach to give a (temporal) description of the meaning of the following program, where the parallel composition operation is synchronous:

```
(x:=true; x:=false) || (y:=true; y:=false)
```

The temporal description of this program is simply:

$$\bigcirc(x \wedge \bigcirc\neg x) \wedge \bigcirc(y \wedge \bigcirc\neg y)$$

which can be rewritten to:

$$\bigcirc(x \wedge y) \wedge \bigcirc\bigcirc(\neg x \wedge \neg y).$$

Example: Concurrency (2)

Recall that models satisfying $\bigcirc(x \wedge y) \wedge \bigcirc\bigcirc(\neg x \wedge \neg y)$ are of the form

$$
\begin{array}{ll}
\textbf{state } 0: & \ldots \\
\textbf{state } 1: & [\, x \mapsto \mathbf{T},\ y \mapsto \mathbf{T} \,] \\
\textbf{state } 2: & [\, x \mapsto \mathbf{F},\ y \mapsto \mathbf{F} \,] \\
\ldots & \ldots
\end{array}
$$

and so such models describe possible execution sequences for the above program:

Example: Concurrency (3)

Note that the following model does *not* satisfy $\bigcirc(x \wedge y) \wedge \bigcirc\bigcirc(\neg x \wedge \neg y)$ and so could not be an execution sequence for the program described above

$$
\begin{array}{ll}
\textbf{state } 0: & \ldots \\
\textbf{state } 1: & [\, x \mapsto \mathbf{T},\ y \mapsto \mathbf{T} \,] \\
\textbf{state } 2: & [\, x \mapsto \mathbf{T},\ y \mapsto \mathbf{F} \,] \\
\textbf{state } 3: & [\, x \mapsto \mathbf{F},\ y \mapsto \mathbf{T} \,] \\
\ldots & \ldots
\end{array}
$$