**Background**

There are many verification & validation techniques of varying coverage, expressivity, level of automation, ..., for example:

- **Automation**
  - Typing
  - Testing
  - Model checking
  - Program logics
  - Operational reasoning

- **Coverage**
  - Complete
  - Bounded
  - Sparse

- **Expressivity** (of safety properties)

---

**Model checking application examples**

- Software, e.g. programs with complex control flow
- Distributed systems
- Protocols
- Asynchronous circuits and hardware
- ...
Model checking

Suppose we are given an algorithm that is supposed to transfer, from one bank of the Cam to the other, using only a punt with seat for one, a wolf, a goat, and a cabbage.

The success criteria are:
- safety: the cabbage and the goat, and the wolf and the goat, cannot be left alone on a bank;
- liveness: all three items are moved to the other bank.

1 It is a large cabbage, so it takes up the whole seat

About finding good models

“All models are wrong, some are useful” applies. The designer must ensure the model captures the significant aspects of the real system. Achieving it is a special skill, the acquisition of which requires thoughtful practice.

— How Amazon Web Services Uses Formal Methods
Temporal models

A temporal model over atomic propositions \( AP \) is a left-total transition system where states are labelled with some of \( AP \), and where some states are distinguished as initial:

\[
\begin{align*}
&M, \ldots \in \text{TModel} \triangleq \\
&(S \in \text{Set}) \times \text{states} \\
&(S_0 \in \text{sub } S) \times \text{initial states} \\
&(\omega T \in \text{relation } S \times S) \times \text{transition} \\
&(\ell \in (S \rightarrow \text{sub } AP)) \times \text{state labelling}
\end{align*}
\]

such that \( T \) is left-total:

\[
\forall s \in S. \exists s' \in S. s \xrightarrow{T} s'
\]
Corner case: the initial temporal model

0 ∈ TModel

\[ 0 \equiv \left( \begin{array}{c}
0, \\
0, \\
0, \\
s \mapsto \emptyset
\end{array} \right) \quad (it \ is \ empty) \]

Corner case: the terminal temporal model

1 ∈ TModel

\[ 1 \equiv \left( \begin{array}{c}
AP \rightarrow B, \\
\{ s \mid \top \}, \\
\{ s_0, s_1 \mid \top \}, \\
s \mapsto \{ p \mid s \ p \}
\end{array} \right) \]

Temporal model of a terrible punter

A punter with no concern for goat welfare or cabbage welfare:

\[
\begin{align*}
\text{Side} & ::= \text{Left} \mid \text{Right} \\
\text{Item} & ::= \text{l} \mid \text{d} \mid \text{c} \mid \text{Y} \\
\text{State} & \equiv \text{Item} \rightarrow \text{Side} \\
\text{AP} & = \text{State} \\
\text{M} & = \left\{ s, s' \mid \begin{array}{l}
\text{moveone } s \ s' \equiv \text{(moveone } s \ s' \text{ item)} \\
\text{movezero } s \ s' \equiv \text{stay } s \ s' \text{ item}
\end{array} \right\}, \\
\text{M} & = \left\{ s \mapsto \{ s \} \right\}
\end{align*}
\]

Temporal model of a terrible punter

\[
\begin{align*}
\text{flip Left} & \equiv \text{Right} \quad \text{flip Right} \equiv \text{Left} \\
\text{moveone } s \ s' & \equiv \text{(moveone } s \ s' \text{ item)} \\
\text{movezero } s \ s' & \equiv \text{stay } s \ s' \text{ item}
\end{align*}
\]
Safety: we never go through a red state.
Liveness: we eventually reach the blue state.
Both are pretty clearly false! :-(

Let us try to describe how an elevator for a building with 3 levels works:
• it starts at the ground floor, with the door closed, and goes back there when it is not called;
• if going through a level where it is called, it stops there and opens its door;
• ...

Textual descriptions do not scale very well.
Temporal model of an elevator: complete (?) dynamics

Definitions

{\textbf{(Infinite) Paths}}

stream \in \text{Set} \rightarrow \text{Set}
stream A \triangleq \mathbb{N} \rightarrow A

\text{IsPath} \in (M \in \text{TModel}) \rightarrow \text{stream } M.S \rightarrow \text{Prop}
\text{IsPath } M \pi \triangleq \forall n \in \mathbb{N}. (\pi \ n) \ M.T (\pi \ (n + 1))

\text{Path} \in \text{TModel} \rightarrow \text{Set}
\text{Path } M \triangleq \{ \pi \in \text{stream } M.S \mid \text{IsPath } M \pi \}

\text{Reachable states & the tail operation}

Because the transition relation is left-total, these infinite paths are “complete”, in the sense that they capture reachability:

\text{Reachable} \in (M \in \text{TModel}) \rightarrow M.S \rightarrow \text{Prop}
\text{Reachable } M s \triangleq \exists \pi \in \text{stream } M.S, n \in \mathbb{N}.
\quad \text{IsPath } M \pi \land M.S_0 (\pi \ 0) \land s = \pi \ n

\text{tailn} \in (A \in \text{Set}) \rightarrow \mathbb{N} \rightarrow \text{stream } A \rightarrow \text{stream } A
\text{tailn } A n \pi \triangleq i \mapsto \pi (i + n)
Stuttering

A temporal model is **stuttering** when all states loop back to themselves:

\[
\text{stuttering} \in \text{TModel} \rightarrow \text{Prop} \\
\text{stuttering} M \overset{\text{def}}{=} \forall s \in M. S \cdot S. T \cdot s
\]

⚠️ If the temporal model is not stuttering, then we can count transitions. This is only sound if they exactly match those of the system being analysed.

See "What good is temporal logic" §2.3, by Leslie Lamport
https://lamport.azurewebsites.net/pubs/what-good.pdf

Applications of model checking

- Hardware:
  - circuits (with memory) directly translate to temporal models
  - lots of protocols
    - cache protocols
    - bus protocols
    - …
    - their specification involves lots of temporal “liveness” ("eventually something good") properties
- Software: often not finite a priori, but “proper modelling”, or bounded model-checking
- Security protocols
- Distributed systems
- …

The common denominator of many of these is the "killer app" of model checking: concurrency.

Examples

In the rest of this lecture, we will sketch how some of these are approached.

The point is not the details of any individual temporal model, but the overall approach.
Temporal model from operational semantics

An initial configuration for a small-step operational semantics naturally leads to a temporal model: take
- configurations as states,
- the initial configuration as the (only) initial state,
- steps as transitions, and
- some interesting properties as atomic propositions, for example
  \[ X, Y, Z, \ldots \in \text{Var} \]
  \[ v \in \mathbb{Z} \]
  \[ AP ::= X = v | X = Y | X > Y | X + Y < Z | X \times Y < Z | \ldots \]

Dealing with the size of temporal model from operational semantics

These temporal models are very often infinite or intractably large!

Many approaches:
- bounded model checking:
  - assume (and possibly check whether) loops execute no more than \( n \) times
  - consider executions of length smaller than \( n \)
  - \ldots
- use a model checking DSL to write an idealised version of the program
- use abstraction
Example circuit

Synchronous (the clock is left implicit) counter that goes 0, 1, 2, 0, 1, 2, ... (assuming all registers are initially 0):

```
Example circuit
Synchronous (the clock is left implicit) counter that goes 0, 1, 2, 0, 1, 2, ... (assuming all registers are initially 0):
```

Registers make the circuit not be a simple function, which motivates using a temporal model.

Example circuit temporal model

The states of the temporal model are the state of the registers, and the labels are which registers are set to 1:

```
Example circuit temporal model
The states of the temporal model are the state of the registers, and the labels are which registers are set to 1:
```

Safety: The state \{r_0, r_1\} should never be reached. Liveness: all other states should be visited infinitely often.

Difference circuit

Given two circuits \(C_1, C_2 \in \text{SCircuit} \ i \ 1\), we can define their difference circuit \(C_1 \ominus C_2\):

```
Difference circuit
Given two circuits \(C_1, C_2 \in \text{SCircuit} \ i \ 1\), we can define their difference circuit \(C_1 \ominus C_2\):
```

If the answer is always 0, then they are equivalent. The typical use case is to have a simple, clearly correct \(C_1\), and a complex \(C_2\) to verify.
Temporal models of distributed algorithms

Nodes in distributed algorithms are often specified in terms of interacting automata; the temporal model directly results from their interaction.

See IB Concurrent and Distributed Systems, Distributed Algorithms, by Nancy Lynch.

Models of cache algorithms

Cache algorithms are also often specified in terms of interacting automata (they are distributed algorithms too).

See Section 21.5.2.1 German’s Protocol in the Handbook of Model Checking.

Models of security protocols

Given a security protocol, define a temporal model where a state contains:

- the state of each agent
- the set of messages sent
- the set of all the messages that can be deduced from the messages sent; this includes taking messages apart, and reassembling them, including via hashing or encrypting using known keys

and where there is a transition from one state to another when

- an agent sends a message
- an adversary sends a deducible message to an agent

See Chapter 22 Model Checking Security Protocols, in the Handbook of Model Checking.

Remark on examples

As illustrated, interesting programs are big, often too big to work on by hand. This is why we use model checkers.

We cannot easily work with such examples here. Instead, we will mostly look at toy examples like the cabbage-goat-wolf puzzle here.

Summary

Temporal models make it possible to describe systems that evolve in time. Model checking allows checking temporal properties of such models.

Temporal models have to capture the relevant parts of an artefact. They can sometimes be extracted directly, for example from circuits, or are hand-crafted to do that.

In the next lecture, we will see how to use temporal logic(s) to specify the behaviour of temporal models.