Verification
Finite-state process modeling and reachability analysis

Topics:
- Finite-state process modeling
- Verification through interactive simulation
- Concurrent composition of processes
- Verification through (exhaustive) reachability analysis

Recall
Difficult to verify non-deterministic process
- New kinds of errors, which may not manifest when exercising traces
- E.g., concurrent system with multiple threads accessing shared data
  - Non-deterministic execution ordering can mean a test that passes during one trial will fail during another

Need to verify process behavior of a system often dominates other design concerns

Terminology
Labeled Transition System
Finite State Process
Parallel Composition (or just composition)

Verifying operational specs
Several techniques available:
- Interactive simulation
  - Tools that allow designer to "execute" the spec
  - Good for checking that specific traces are supported
  - May demonstrate failures (e.g., deadlocks, refusals)
- Reachability analysis
  - Use graph-theoretic algorithms to answer questions about a spec
    - E.g., is a "bad" state reachable from start state?
- Temporal-logic model checking

FSP and LTSA
Finite State Processes (FSP)
- Textual language for expressing behavior of processes in terms of actions, synchronization, and algebraic combining operators
- Based on CSP
- Variables limited to finite domains

Labeled Transition System Analyzer
- Performs efficient reachability analysis
- Specifications written in the FSP language

Quick introduction to FSP
Two kinds of entities:
- processes: correspond to states in a state diagram
  - always written using capital letters
- actions: correspond to events in a state diagram
  - always written beginning with a lower-case letter

Predefined process STOP:
- a.k.a., "deadlock"
- the process that can engage in no action whatsoever
Action prefixing

Programmers may define a new FSP process by prefixing an existing process with some action.

Formally:

If $x$ is an action and $P$ a process then $(x \rightarrow P)$ describes a process that initially engages in the action $x$ and then behaves exactly as described by $P$.

Examples:
- ONESHOT = once $\rightarrow$ STOP.
- is the process that can engage in the action once and then continue to behave as the process STOP.

Prefixing and recursion

To specify repetitive behavior, use recursion:

$\text{SWITCH} = \text{OFF},
\text{OFF} = (\text{on} \rightarrow \text{ON}),
\text{ON} = (\text{off} \rightarrow \text{OFF}).$

Alternatively:

$\text{SWITCH} = (\text{on} \rightarrow \text{off} \rightarrow \text{SWITCH}).$

Choice in FSP

FSP also allows the specification of choice.

More formally:

If $x$ and $y$ are actions then $(x \rightarrow P | y \rightarrow Q)$ describes a process which initially engages in either of the actions $x$ or $y$. After the first action has occurred, the subsequent behavior is described by $P$ if the first action was $x$ and $Q$ if the first action was $y$.

Example:

$\text{SLOT} = (\text{nickel} \rightarrow \text{SLOT} | \text{dime} \rightarrow \text{SLOT}).$

FSP specification of vending machine

$\text{VM} = \text{START},
\text{START} = (\text{quarter} \rightarrow \text{TWENTYFIVECENTS}),
\text{TWENTYFIVECENTS} = (\text{quarter} \rightarrow \text{FIFTYCENTS} | \text{candy} \rightarrow \text{START}),
\text{FIFTYCENTS} = (\text{coffee} \rightarrow \text{START} | \text{candy} \rightarrow \text{TWENTYFIVECENTS}).$

Using LTSA to simulate spec

The LTSA tool provides a graphical interface for:

- interactively simulating an FSP spec
- displaying the LTS denoted by the spec

<< Example [VM.FSP] >>

Guarded actions

Actions may also be guarded with conditions.

More formally:

The choice (when $B$ $x \rightarrow P | y \rightarrow Q$) means that when the guard $B$ is true then the actions $x$ and $y$ are both eligible to be chosen, otherwise if $B$ is false then the action $x$ cannot be chosen.

Example:

$\text{COUNT (N=3)} = \text{COUNT}[0],
\text{COUNT}[i:0..N] = (\text{when}(i<N) \text{inc} \rightarrow \text{COUNT}[i+1] | \text{when}(i>0) \text{dec} \rightarrow \text{COUNT}[i-1]).$
**Alternative spec of vending machine**

VM(N=2) = MACHINE[0].
MACHINE[0..N-1] = (when (i < 2) quarter -> MACHINE[i+1] when (i > 0) candy -> MACHINE[i-1] when (i > 1) coffee -> MACHINE[i-2]).

<< Example: [VM2_FSP] >>

---

**Concurrent composition**

New processes may be defined by concurrently executing other processes.

More formally:

If P and Q are processes then (P||Q) represents the concurrent execution of P and Q. The operator || is the parallel composition operator.

Example:

ITCH = (scratch → STOP).
CONVERSE = (think → talk → STOP).

||CONVERSE_ITCH = (ITCH || CONVERSE).

---

**Shared actions**

If processes in a composition have actions in common, these actions are said to be **shared**. Shared actions are the way that process interaction is modeled.

While unshared actions may be arbitrarily interleaved, a shared action must be executed at the same time by all processes that participate in the shared action.

---

**Example of shared actions**

Consider a system that independently models resource production and use

MAKER = (make → ready → used → MAKER).
USER = (ready → use → used → USER).

||MAKER_USER = (MAKER || USER).

What are some traces of MAKER_USER?

---

**Modeling handshaking protocols**

Handshake is an action acknowledged by others

MAKERv2 = (make → ready → used → MAKER).
USERv2 = (ready → use → used → USER).

||MAKER_USERv2 = (MAKERv2 || USERv2).

Notice how interaction constrains the behavior.

What are some traces of MAKER_USERv2?

---

**Composing state graphs**

[start, s1]
Composing state graphs

- (start, s1)
- quarter
- (25, s2)
- quarter
- (50, s3)

CSE 435: Software Engineering
K. Stirewalt
Composition using LTSA

Example [DetComposition.FSP]

Restricting to a set of traces

Composition can also be used to restrict a rich specification to a smaller set of traces

Example
Question:

What would you expect to happen when we compose the non-deterministic vending machine with the coffee scenario?

If this were a more complex specification, how would you know when you have "sufficiently simulated" the spec to guarantee that it would not deadlock?
Composing (non-det) state graphs

- **<start,s1>**

- **start**
  - 50C1
  - coffee
  - quarter
  - candy
  - 25

- **s1**
  - quarter
  - 50C0
  - quarter
  - candy
  - 25

- **s2**
  - quarter

- **s3**

- **<start,s1>** quarter <25,s2>

- **<start,s1>** quarter <25,s2> 50C1,s3

- **<start,s1>** quarter <25,s2> 50C2,s3

- **<start,s1>** quarter <25,s2> 50C1,s3

- **<start,s1>** quarter <25,s2> 50C2,s3

- **Deadlock!!!**
Reachability of ERROR state

The composed state diagram tells us something about how the system supports the given trace.

If ERROR is reachable, then it is possible for the system to deadlock on this trace!

Thus, composing the processes and checking for the possibility of deadlock provides more certainty than does simulation.

Using LTSA to detect deadlock

<< Example [NonDetComposition.FSP] >>
Reachability analysis

Technique for reasoning about operational specifications
- Treats specification as a directed graph
- Analyzes the graph to determine:
  • whether certain desired (or undesired) nodes are reachable from the "start" node
  • whether certain actions or sequences of actions are enabled "infinitely often"—i.e., that the system never gets into a state where it can deny them

Useful for proving properties of an operational spec
- E.g., does spec S support trace t?
- E.g., might a system that implements spec S deadlock?

Multi-party synchronization

MAKE_A  = (makeA->ready->used->MAKE_A).
MAKE_B  = (makeB->ready->used->MAKE_B).
ASSEMBLE = (ready->assemble->used->ASSEMBLE).
||FACTORY = (MAKE_A || MAKE_B || ASSEMBLE).