6. Inductive Definitions and Fixed Points 6.3 Specifying and verifying transition systems

Specifying and verifying transition systems

Section 6.3

Motivation

Modeling

Behavior of software-controlled systems can be modeled
- by using a modeling language (UML, B, Z, ASM, ABS, Maude, ...)
- by formalizing the operational behavior as transition system

Transition systems

Transition systems are also a fundamental means for specifying
- the operational semantics of programming and modeling language (cf. Chap. 7)
- process calculi and concurrency
- computing architectures and hardware

Verification of transition systems cannot exploit program structure, but need other techniques.

Transition systems (2)

Remark
- The action labels express input, output, or an “explanation” of an internal state change.
- Finite automata are LTS.
- Often, transitions systems are equipped with a set of initial states or sets of initial and final states.
- Traces are sequences \(\langle q_i \rangle\) of states with \((q_i, q_{i+1}) \in T\) or sequences of labels
- Behaviors are sets of traces (beginning at initial states)
- Properties are often expressed in appropriate logics (PDL, CTL ...)

Definition (Transition system)
A transition system (TS) is a pair \((Q, T)\) consisting of
- a set \(Q\) of states;
- a binary relation \(T \subseteq Q \times Q\), usually called the transition relation.
Notation: \(q \rightarrow q'\)

(Other names: state transition system, unlabeled transition system)

Definition (Labeled transition system)
A labeled transition system (LTS) over \(Act\) is a pair \((Q, T)\) consisting of
- a set \(Q\) of states;
- a ternary relation \(T \subseteq Q \times Act \times Q\), usually called the transition relation.
Notation: \(q \xrightarrow{lab} q', \quad lab \in Act\)

Act is called the set of actions or labels.
Lemma

Every LTS \((Q, T)\) over \(\text{Act}\) can be expressed by a TS \((Q', T')\) such that there is a mapping

\[ rep : Q \times \text{Act} \Rightarrow Q' \]

with

\[ q_1 \xrightarrow{\text{lab}} q_2 \in T \iff \exists \text{lab}. \ rep(q_1, \text{lab}) \xrightarrow{\text{lab}} \rep(q_2, \text{lab}) \in T' \]

(Proof is left as an exercise)

Modeling: Case study Elevator control system

Requirements

Design the control for an elevator serving 3 floors such that:

- Model:
  - Elevator has for each floor one button which, if pressed, causes it to visit that floor. Button is cancelled when the elevator visits the floor.
  - Each floor has a button to request the elevator. Button is cancelled when elevator visits the floor.
  - The elevator remains in the middle floor if no requests are pending.

- Properties:
  - All requests for floors from the elevator must be serviced eventually.
  - All requests from floors must be serviced eventually.

Datatypes for facts and actions

\[
\begin{align*}
\text{datatype floor} &= \text{F0} | \text{F1} | \text{F2} & (* \text{three floors} *) \\
\text{datatype action} &= \text{Call floor} | \text{GoTo floor} | \text{Open} | \text{Move} & (* \text{input message} *) \\
& & (* \text{output message} *) \\
\text{datatype direction} &= \text{UP} | \text{DW} & (* \text{up} | \text{down} *) \\
\text{datatype door} &= \text{CL} | \text{OP} & (* \text{closed} | \text{open} *) \\
\text{type_synonym} & \text{state} = \text{action} \times \text{direction} \times \text{door} \times (\text{floor set}) & (* \text{what} \times \text{floor} \times \text{where to} \times \text{door state} \times \text{requests} *)
\end{align*}
\]
6. Inductive Definitions and Fixed Points

6.3 Specifying and verifying transition systems

Datatypes and actions: Transition relation

```plaintext
inductive_set tr :: (state × state) set where
  g /nelement T; ¬ (f = g ∧ d = OP) ⟹
  (a,f,r,d,T), (Call g,f,r,d,T∪{g})) ∈ tr |
  g /nelement T; ¬ (f = g ∧ d = OP) ⟹
  (a,f,r,d,T), (GoTo g,f,r,d,T∪{g})) ∈ tr |
  f ∈ T ⟹
  (a,F1,r,d,{F0}), (Move,F0,DW,CL,{F0})) ∈ tr |
  F0 /nelement T ⟹
  (a,F1,UP,d,T), (Move,F1,UP,CL,T)) ∈ tr |
  F2 /nelement T ⟹
  (a,F1,UP,d,T), (Move,F2,UP,CL,T)) ∈ tr |
  F1 /nelement T; F2 /nelement T ⟹
  (a,F1,UP,d,T), (Move,F2,UP,CL,T)) ∈ tr |
```

Traces

Defining sets of infinite traces

```plaintext
types trace = "nat ⇒ state"
coinductive_set traces :: "trace set" where
  "[ t ∈ traces; (s, t 0) ∈ tr ⟹
  (λ n. case n of 0 ⇒ s | Suc x ⇒ t x) ∈ traces"
```

More interesting properties

Expressing temporal properties of traces

- For every floor f: If f is a requested floor, the elevator will eventually reach the floor and open the door in f:

  ```plaintext
  Always (≪To f≫ → Finally (≪Op≫ and ≪At f≫))
  ```

  Could be directly expressed over traces

- Alternative: Temporal logic, e.g., linear TL:
  - Formulas built with Atoms, ¬, ∧, □, ◯
  - Interpretations: Kripke structures (Q, I, T, L)
  - A transition relation T ⊆ Q × Q such that ∀q ∈ Q.3q' ∈ Q.(q,q') ∈ T
  - A labeling (or interpretation) function L : Q → 2Atoms
```

Basic properties of traces

- lemma [iff]: "drp (drp t n) m = drp t (n + m)"
- lemma drp_traces: "t ∈ traces ⟹ drp t n ∈ traces"
### Syntax for LTL

LTL formulas:

```plaintext
datatype formula = Atom atom ("≪ _ ≫")  
                 | Neg formula ("¬")  
                 | And formula formula (infixr ".∧" 80)  
                 | Always formula ("□")  
                 | Finally formula ("⋄")
```

As abbreviation:

```plaintext
definition Imp :: "formula ⇒ formula ⇒ formula"  
                 (infixr ".→" 80)  
where
  "a .→ b = .¬ (a .∧ .¬ b)"
```

### Semantics for LTL

**Definition (Kripke structure)**

Let $AP$ be a set of atomic propositions. A *Kripke structure* is a 4-tuple $M = (Q, I, T, L)$ consisting of:

- a finite set of states $Q$
- a set of initial states $I ⊆ Q$
- a relation $T ⊆ Q × Q$ such that $∀ q ∈ Q \exists q′ ∈ Q$ with $(q, q′) ∈ T$
- a labeling (or interpretation) function $L :: Q ⇒ 2^{AP}$

### Kripke structure of elevator example

- $Q$ as defined by type synonym "state" (*UNIV state*)
- $I$: some suitable set of initial states
- $T$ as defined by $tr$ (why is there always a successor state?), and
- define $AP ≡ atom$ and $L$ as follows:

```plaintext
datatype atom = Up | Op | At floor | To floor

fun L :: "state ⇒ atom ⇒ bool" where  
  "L (_, _, r, _, _) Up = (r = UP)" |  
  "L (_, _, r, d, _) Op = (d = OP)" |  
  "L (_, f, _, _, _) (At g) = (f = g)" |  
  "L (_, _, _, _, fs) (To f) = (f ∈ fs)"
```

### Remarks and example

**Remarks:**

- Since $T$ is left-total, it is always possible to construct an infinite path through the Kripke structure. A *deadlock state* $qd$ can be expressed by a single outgoing edge back to $qd$ itself.
- The labeling function $L$ defines for each state $q$ in $Q$ the set $L(s)$ of all atomic propositions that are valid in $s$.
- Kripke structures are used to define the semantics of LTL (see next slide)

**Example of formalized property:**

```plaintext
definition liveness :: "floor ⇒ formula" where  
  "liveness f = □ (≪To f≫ .→ ⋄ (≪Op≫ .∧ ≪At f≫))"
```
Semantics for LTL

Let $M = (Q, I, T, L)$ be a Kripke structure and `trace` the type of traces defined by $T$:

```haskell
primrec valid_in_trace :: "trace ⇒ formula ⇒ bool" ("(_ ⊨ _)"
where
    "t ⊨ ≪ a≫ = ( a ∈ L (head t) )"
| "t ⊨ .¬ f = ( ¬ (t ⊨ f) )"
| "t ⊨ f ∧ g = ( (t ⊨ f) ∧ (t ⊨ g) )"
| "t ⊨ □ f = ( ∀ n. ((drp t n) ⊨ f ))"
| "t ⊨ ◻ f = ( ∃ n. ((drp t n) ⊨ f ))"

definition valid :: "formula ⇒ bool" ("⊨ _"
where
    "⊨ f ≡ ( ∀ t ∈ traces. t ⊨ f )"
```

Reasoning about finite transition systems

Three options for reasoning:

1. In Isabelle/HOL using the rules obtained from the definitions (semantics-based, formalized mathematical reasoning):
   » Elevator.thy
2. In LTL using rules for temporal reasoning (rules not shown here)
3. Model checking (works for finite state systems)