Chapter 7

Application: Inductively Defined Sets

Defining sets inductively: Repetition

SessionSlides6.1 starting slide 23

- Rule induction
- Demo inductively defined sets
- Inductive predicates
- Demo

Transition systems

Definition 7.1. *TS*
A transition system (TS) is a pair (Q,T) consisting of

- *a set Q of states;*
- *a binary relation $T \subset (Q * Q)$, usually called the transition relation*

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

Definition 7.2. *LTS*
A labeled transition system (LTS) over Act is a pair (Q,T) consisting of

- *a set Q of states;*
- *a ternary relation $T \subset (Q * Act * Q)$, usually called the transition relation, transitions written as q1 -l-> q2*

Act is called the set of actions.

Transition systems (cont.)

Remark 7.3.

- 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 (qi) of states with $(qi, qi+1) \in T$
- Behavior:: Set of traces beginning at initial states.
- Properties:: expressed in appropriate logic (PDL, CTL ...)

Lemma 7.4. *Every LTS (Q,T) over Act can be expressed by a TS (Q’,T’) such that there is a mapping*
$rep : Q * Act \Rightarrow Q'$
with $q1 - l- > q2 \in T \iff \exists l' : (rep(q1, l'), rep(q2, l)) \in T'$

Proof: <exercise>

Modeling: Case study Elevator

Model of an elevator control system: Description

- Design the logic to move one lift between 3 floors satisfying:
- The lift has for each floor one button which, if pressed, causes the lift to visit that floor. It is cancelled when the lift visits the floor.
- Each floor has a button to request the lift. It is cancelled when the lift visits the floor.
- The lift remains in middle floor if no requests are pending.
- Properties
- All requests for floors from the lift must be serviced eventually.
- All requests from floors must be serviced eventually.

Modeling: Case study Elevator

Datatypes and actions

```
datatype floor = F0 | F1 | F2
(* actions *)
datatype action = Call floor  (* input message *)
                | GoTo floor  (* input message *)
                | Open        (* output message *)
                | Move       (* internal message *)

(* types for elevator state *)
datatype direction = UP | DW
datatype door      = CL | OP
(* elevator state *)
"action * floor * direction * door * (floor set)"
(* where | last move | open/closed | what to serve *)
```

Datatypes and actions

```
(* transition relation *)
inductive_set tr :: "(state * state) set" where
 "\<lbrakk> g \<notin> T; \<not> (f = g \<and> d = OP)
     \<rbrakk> \<Longrightarrow> ( (a, f, r, d, T), (
    Call g, f, r, d, T \<union> { g }) ) \<in> tr"
| "\<lbrakk> g \<notin> T; \<not> (f = g \<and> d = OP
   ) \<rbrakk> \<Longrightarrow> ( (a, f, r, d, T), (
   GoTo g, f, r, d, T \<union> { g }) ) \<in> tr"
| "f \<in> T  \<Longrightarrow> ( (a, f, r, d, T), (
   Open, f, r, OP, T - { f }) ) \<in> tr"
| "( (a, F1, r, d, { F0 }), (Move, F0, DW, CL, { F0 })
    ) \<in> tr"
| "( (a, F1, r, d, { F2 }), (Move, F2, UP, CL, { F2 })
    ) \<in> tr"
| "F0 \<notin> T \<Longrightarrow> ( (a, F0, r, d, T),
    (Move, F1, UP, CL, T) ) \<in> tr"
```

Datatypes and actions (cont.)

```
| "F2 \<notin> T \<Longrightarrow> ( (a, F2, r, d, T),
    (Move, F1, DW, CL, T) ) \<in> tr"
| "\<lbrakk> F1 \<notin> T; F2 \<in> T \<rbrakk> \<
   Longrightarrow> ( (a, F1, UP, d, T), (Move, F2, UP
   , CL, T) ) \<in> tr"
| "\<lbrakk> F1 \<notin> T; F0 \<in> T \<rbrakk> \<
   Longrightarrow> ( (a, F1, DW, d, T), (Move, F0, DW
   , CL, T) ) \<in> tr"

(* traces *)
types trace = "nat \<Rightarrow> state"

coinductive_set traces :: "trace set" where
"[| t \<in> traces; (s, t 0) \<in> tr |]  ===>
(\<lambda>n. case n of 0 \<Rightarrow> s | Suc x \<
   Rightarrow> t x) \<in> traces"
```

Datatypes and actions (cont.)

```
(* Functions on traces *)

definition head :: "trace \<Rightarrow> state" where
  "head t \<equiv> t 0"

definition drp :: "trace \<Rightarrow> nat \<
    Rightarrow> trace" where
  "drp t n \<equiv> (\<lambda>x. t (n + x))"

lemma [iff]:
  "drp (drp t n) m = drp t (n + m)"

lemma drp_traces:
  "t \<in> traces ===> drp t n \<in> traces"
```

Reasoning about finite transition systems

Logic for expressing properties of traces

- For every floor f: If f is a target floor, the elevator will eventually reach the floor and open the door.
- Always («To f» –> Finally («Op» and «At f»))
- $\rightsquigarrow$ Temporal logic. Here e.g. LTL
- Formulae built with Atoms, $\neg, \wedge, \Box, \Diamond$
- Interpretations: Kripke structures (Q, I, T, L)
- A transition relation $T \subseteq Q * Q$ such that $\forall q \in Q. \exists q' \in Q.(q, q') \in T$
- a labeling (or interpretation) function $L : Q \rightarrow 2^{Atoms}$

Reasoning about finite transition systems

Remark 7.5.

- 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 single outgoing edge back to qd itself.
- Labeling states (elevator)

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

fun L :: "state => atom => bool" where
"L (_, _, UP, _, _) Up = True" |
"L (_, _, DW, _, _) Up = False" |
"L (_, _, _, CL, _) Op = False" |
"L (_, _, _, OP, _) Op = True" |
"L (_, f, _, _, _) (At g) = (f = g)" |
"L (_, _, _, _, fs) (To f) = (f \<in> fs)"
```

Reasoning about finite transition systems (cont.)

- The labeling function L defines for each state q in Q the set L(s) of all atomic propositions that are valid in s.
- Semantics of LTL

```
primrec valid ::
"trace => formula => bool"   ("(_ |= _)" [80, 80]
    80) where
   "t |= Atom a     = ( a \<in> L (head t) )"
 | "t |= Neg f      = ( \<not> (t |= f) )"
 | "t |= And f g    = ( t |= f \<and> t |= g )"
 | "t |= Always f   = ( \<forall>n. drp t n |= f )"
 | "t |= Finally f = ( \<exists>n. drp t n |= f )"
```

- »> Elevator.thy