Abstract State Machine

The first presentation in the course 760

By Bingzhou Zheng

Outline

1. The central idea of Abstract State Machine (ASM)
2. The example Set Extension in pseudo code program and its semantics
3. Sequential Small-Step ASM Program and its semantics
4. Formalizing Set Extension in Sequential Small-Step ASM Program

The Central Idea of ASM

- *Abstract State Machine* (ASM) is a technique to describe algorithm or, more generally, discrete system.
- *Symbols* occurring in a program of ASM are related to the real-world objects and functions of a *state*.
- *ASM* relate symbols to their interpretation by using *structures* (*models*), which including functions and predicates over real world items.
- *ASM* use first order logic to define and analyze such structures.

Terms: Sequential Small-Step ASM

- A discrete system can be represented as a "sequential small-step ASM", if
  - the system exhibits global states
  - the system proceeds in steps form state to state
  - for each step $S \rightarrow S'$, to derive S' from S, it suffices to explore a bounded amount of information about S.

Terms: Transition System (classical models of discrete system)

- A transition system A=(states, **init**, F)
  - a set states of "states"
  - **Init** $\subseteq$ *states* of "initial states"
  - a "next state function" $F: \ states \rightarrow states$
  - a *run* of a transition system is a sequence $S_0 S_1 S_2 \ldots$ of states S_i with S_0 an initial state and $S_i = F(S_{i-1})$.
- A transition system is called an algorithm if each run of the system reach a terminal state
- An effective transition system is an effective algorithm.

Example: Set Extension

- augment is a binary function to extend a set by a item.
  - the first argument is a set.
  - the second argument is any item.

$$augment(M, m) =_{def} M \bigcup \{m\}$$

- To write this idea down in pseudo code, introduce
  - three variable (symbols) *X, x, y*
  - initial state *S* in which *X* is evaluated as *M* (a set), *x* as *m* (an element), *y* as *n* (an element), and augment as the function defined above
  - The pseudo code program as following

Example: Set Extension cont.

```
P: begin
        X := augment(X, x);
        X := augment(X, y);
    end.
```

This program applied to S terminates in a state S'' in which X is evaluated as $M \bigcup \{m, n\}$

Example: Set Extension cont.

- The pseudo code program *P* has a finite set $\Sigma = \{X, x, y\}$
- The finite set ∑ is interpreted in the initial state

  $S = \{X_S = M, x_S = m, y_S = n, augment_S = augment\}$

- The program *P* is applicable to the state *S*. The first assignment statement *X := augment(X, x)* updates *S* and yield *S'*:

  $S' = \{X_{S'} = X_S \bigcup \{m\}, x_{S'} = m, y_{S'} = n, augment_{S'} = augment\}$

- The second assignment *X := augment(X, y)* update *S'* to *S"*: $S'' = \{X_{S''} = X_{S'} \bigcup \{n\}, x_{S''} = m, y_{S''} = n, augment_{S''} = augment\}$

Pseudo code Algorithm

- Every pseudo code program P has a finite set ∑ of symbols to be interpreted in a state.
- A state of P is an interpretation of all symbols in ∑.
- There is an infinite set of states of P. An algorithm is not intended to run on all states.
- The designer of algorithm is free to choose the states, including initial states, which the algorithm is intended for.

Pseudo code Algorithm cont.

- A pseudo code algorithm M is a triple M=(states, **init**, P)
  - P is a pseudo code program, applicable to each state $S \in states$, and return a state $P(S) \in states$
  - **Init** is a set and $Init \subseteq states$

- The algorithm M of the example Set Extension defines a transition system tr(M)

(Remember? a transition system is a triple A=(states, **init**, F))

- A class of pseudo code algorithms is called *sequential small-step algorithms*.

Sequential Small-Step ASM Programs

- Assignment Statements
  - Simple Assignment Statements
  - Updates of Functions

- Consistent Assignment Statements

- Guards and Conditional Assignment Statements

Simple Assignment Statements

- Simple assignment statements of a Sequential Small-Step ASM program over a signature Σ has the form:

$$f := t$$

  - f is a constant symbol in Σ
  - $t \in T_\Sigma$
  - T_Σ is the set of ground terms over Σ
    - each constant symbol in Σ is an element of T_Σ
    - if $f \in T_\Sigma$ with the arity of n and if $t_1, \ldots, t_n \in T_\Sigma$, then $f(t_1, \ldots, t_n) \in T_\Sigma$

SA Statements Cont.

- Applied to a Σ-structure *S*, yield the step $S \xrightarrow{f:=t} S'$
  - S' updates the value of *f*, the constant symbol *f* gains t_S as a new value in S', i.e. $f_{S'} = t_S$
  - the semantics of all other symbols remains untouched, i.e. $g_{S'} = g_S$ for each $g \in \Sigma,\ g \neq f$

Updates of Functions

- The general form of the updates over a signature Σ is of the form:

$$f(t_1,\ldots,t_n) := t$$

  - with $f \in \Sigma$
  - and $t_1,\ldots t_n, t \in T_\Sigma$

- A step $S \xrightarrow{f(t_1,\ldots t_n):=t} S'$ updates f_S at $(t_{1_S} \ldots t_{n_S})$ by t_S , yielding $f_{S'}(t_{1_S} \ldots t_{n_S}) := t_S$. The function f remains untouched for all other arguments i.e.

$$f_{S'}(u_1,\ldots,u_n) = f_S(u_1,\ldots,u_n) \text{ for } (u_1,\ldots,u_n) \neq (t_{1_S},\ldots,t_{n_S})$$

Consistent Assignment Statements

- A step $S \to S'$ of an ASM program in general executes more than one assignment statements, provided every two such assignments are consistent.

- Consistent assignments:

  Two assignments $f(t_1, \ldots, t_n) := t$ and $f(u_1, \ldots, u_n) := u$ are consistent at a state S if $(t_{1_S}, \ldots, t_{n_S}) = (u_1, \ldots, u_n)$ implies $t_S = u_S$.

CA Statements cont.

- let Z be a set of assignment statements with terms in T_{Σ}
- let S be a Σ-structure and assume that Z is consistent at S

$$S \xrightarrow{Z} S'$$

  - S' *is a* Σ-structure
  - the universe U of S' is identical to the universe of S.

$$f_{S'}(u) = \begin{cases} v & \text{if Z at S updates } f_S(u) \text{ by } v \\ f_S(u) & \text{otherwise} \end{cases}$$

Guards and Conditional Assignment Statements

- ASM employ *conditional* assignment statements, of the form

  if α then *r*

  - *r* is an assignment statement
  - α is a Boolean expression
  - the term α plays the role of a *guard* of *r*
- The guard over Σ are symbols sequence:
  - for all $t, u \in T_{\Sigma}$, $t = u$ is a guide over Σ
  - if α, β are guards over Σ, so are $\alpha \wedge \beta$, and $\neg \alpha$
  - we assume each Σ is extended by $=, \wedge, \neg,$ true, false

SSS ASM Programs and Semantics

- A Sequential Small-Step ASM program *P* over a signature Σ is a set of conditional assignment statements over Σ .
- For each Σ-Structure *S*, the program *P* defines a *successor structure* *S'*, usually written *P*(*S*), by step

$$S \xrightarrow{P} S'$$

- To define *S'*, let $Z =_{def} \{r \mid \text{"if } \alpha \text{ then r"} \in \text{P and } \alpha_S = true\}$ and construct *S'* by:

$$f_{S'}(u) = \begin{cases} v & \text{if Z at S updates } f_S(u) \text{ by } v \\ f_S(u) & \text{otherwise} \end{cases}$$

SSS ASM Programs and Semantics cont.

- ASM are reactive systems which iterate their computation step.

- For the special case of terminating runs, one can choose among various natural termination criteria
  - No statement is applicable any more
  - Machine yields an empty update set
  - The state does not change any more

SSS ASM Programs and Semantics cont.

- An ASM computation step in a given state consists in executing simultaneously (parallel) all updates of all assignment states whose guard is true in the state.
- If these updates are consistent, the result of their execution yields the next state.

Set Extension in SSS ASM Program

P : **par**

if $l = 0$ then X := g(X, x);

if $l = 0$ then l := 1 ;

if $l = 1$ then X := g(X, y);

if $l = 1$ then l := 2

endpar.

- An ASM program cannot express sequential composition. This deficit is easily overcome by adding a fresh variable, l, and evaluating l as 0 in the initial state *S*.

Bisection Algo in Pseudo code

- Pseudo code Program

```
while |f(a) – f(b)| ≥ ε do
        m := mean(a,b);
        if sign(a) ≠ sign(m) then b := m
                                  else a := m
```

Bisection Algo in SSS ASM Prog

- SSS ASM Program:

P : **par**

if stop(a,b) = true then result := a;

if ¬ (stop(a,b)=true) ∧ f(mean(a,b))=0
then result:=mean(a,b);

if ¬ (stop(a,b)=true) ∧ f(mean(a,b)) ≠ 0
∧ eqsign(f(a),f(mean(a,b)))=true
then a:=mean(a, b);

if ¬(stop(a,b)=true) ∧ f(mean(a,b)) ≠ 0
∧ eqsign(f(b),f(mean(a,b)))=true
then b:=mean(a, b);

endpar.

Reference

- **Wolfgang Reisig**, Abstract State Machines for the Classroom – The Basics
- **Egon Borger, Robert Stark**, Abstract State Machines – A method for High-level System Design and Analysis

The End

Question?