A Step-wise Development Method with Progress Concerns

Thai Son Hoang
(joint work with Simon Hudon)

Institute of Information Security, Department of Computer Science
Swiss Federal Institute of Technology Zürich (ETH Zürich)

InfSec Group Seminar, ETH Zurich
26th March 2013

Refinement
The UNITY way vs. the Event-B way

- UNITY: Refines the formulae.

\[ \phi \leftarrow \phi_1 \leftarrow \ldots \leftarrow \phi_n \leftarrow M \]

Translation

- Cons: Hard to understand the choice of refinement.

- Event-B: Refines transition systems.

\[ \phi \models M_0 \subseteq M_1 \ldots \subseteq M \]

Verification

- Cons: No support for liveness properties.

To develop a system \( M \) satisfying property \( \phi \), i.e., \( M \models \phi \).

- \( M \): some transition system
- \( \phi \): some logical formula

The main challenge: the complexity of the system.

Refinement allows the step-by-step design of the system.

Inspired by UNITY and Event-B.

- Support the reasoning of liveness properties (UNITY).
- Refinement of transition systems (Event-B style).

Developments using Unit-B are guided by both safety and liveness requirements.
Running Example. A Signal Control System

SAF 1 There is at most one train on each block

LIVE 2 Each train in the network eventually leaves

Traces and the Language of Temporal Logic

A trace $\sigma$ is a (finite or infinite sequence of states)

$\sigma = s_0, s_1, s_2, s_3, \ldots$

- A (basic) state formula $P$ is any first-order logic formula,
- The basic formulas can be extended by combining the Boolean operators ($\neg$, $\land$, $\lor$, $\Rightarrow$) with temporal operators:
  
  always: $\square \phi$

  eventually: $\Diamond \phi$

Guarded events

$e$ any $t$ where $G.t.v$
then $S.t.v.v'$
end

- $t$: parameters
- $G.t.v$: guard
- $S.t.v.v'$: action

- $e.t$ is enabled when $G.t.v$ holds.
- Execution of $e.t$: $v$ is updated according to the action $S.t.v.v'$.
- $e.t$ corresponds to a formula $act.(e.t)$. 
Scheduled events (1/2)

- \( C.t.v \): coarse-schedule.
- \( F.t.v \): fine-schedule.
- Healthiness condition:
  \[ C.t.v \land F.t.v \Rightarrow G.t.v \]

**Liveness (Scheduling) Assumption**

If \( C.t.v \) holds infinitely long and \( F.t.v \) holds infinitely often then eventually \( e.t \) is executed.

\[
sched(e.t) = \Box(\Box C \land \Box F \Rightarrow \Box(F \land act(e.t)))
\]

Scheduled events (2/2)

Conventions

- \( e \equiv \text{any } t \text{ where } \ldots \text{ during } C.t.v \text{ upon } F.t.v \text{ then } \ldots \text{ end} \)
- Unscheduled events (without during and upon): \( C \) is \( \bot \)
- When only during is present (no upon), \( F \) is \( T \).
- When only upon is present (no during), \( C \) is \( T \).

Schedules vs. Fairness

- \( e \equiv \text{any } t \text{ where } G.t.v \text{ during } C.t.v \text{ upon } F.t.v \text{ then } \ldots \text{ end} \)

- Schedules are a generalisation of weak- and strong-fairness.

- Weak-fairness:
  - If \( e \) is enabled infinitely long then \( e \) eventually occurs.
    - Let \( C \) be \( G \) and \( F \) be \( T \).

- Strong-fairness:
  - If \( e \) is enabled infinitely often then \( e \) eventually occurs.
    - Let \( F \) be \( G \) and \( C \) be \( T \).

Execution of Unit-B Models

\[
\begin{align*}
\text{ex.} M &= \text{saf.} M \land \text{live.} M \\
\text{saf.} M &= \text{init} \land \Box \text{step.} M \\
\text{step.} M &= (\exists e.t \in M \cdot \text{act.}(e.t)) \lor \text{SKIP} \\
\text{live.} M &= \forall e.t \in M \cdot \text{sched.}(e.t) \\
\text{sched.}(e.t) &= \Box(\Box C \land \Box F \Rightarrow \Box(F \land \text{act.}(e.t)))
\end{align*}
\]
A Signal Control System (Recall)

- **SAF 1**: There is at most one train on each block.
- **LIVE 2**: Each train in the network eventually leaves.

Refinement strategy: Prioritise **LIVE 2** first.

**Execution and Properties**

- **M** satisfies $\phi$ if and only if $\text{ex.} \ M \Rightarrow \phi$.

**Safety Properties**

- **Invariance** properties: (in LTL $\Box I$
  - $I$ holds for every reachable state.
  - Proved using the standard induction technique.

- **Unless** properties: $P \un Q$
  - if $P$ holds at some point then it continues to hold unless $Q$ holds.
  - Prove: If for every event
    
    $$e \cong \text{any } t \text{ where } G.t.v \text{ during } \ldots \text{ upon } \ldots \text{ then } S.t.v'.v' \text{ end}$$
    
    in $M$, we have
    
    $$P.v \land \neg Q.v \land G.t.v \land S.t.v'.v' \Rightarrow P.v' \lor Q.v'$$  \hspace{1cm} (UN)
    
    then $M$ satisfies $P \un Q$. 

**A Signal Control System. The Initial Model**

- Focus on trains in the network.
- Set $TRN$ denotes the set of possible trains.
- Variable $trns$ denotes the set of trains in the network.
- Event **arrive** models a train entering the network.
- Event **depart** models a train leaving the network.

**Variables**

- $TRN$: Set of possible trains.
- $trns$: Set of trains in the network.

**Events**

- **arrive** $\text{any } t \text{ where } t \in TRN$ then $trns := trns \cup \{t\}$

- **depart** $\text{any } t \text{ where } t \in trns$ during $t \in trns$ then $trns := trns \setminus \{t\}$
Liveness Properties

- Progress properties $P \rightarrow Q$.
- In LTL: $\Box(P \Rightarrow \Diamond Q)$
- Some important rules

\[
(P \Rightarrow Q) \Rightarrow (P \rightarrow Q) \quad \text{(Implication)}
\]
\[
(P \rightarrow Q) \land (Q \rightarrow R) \Rightarrow (P \rightarrow R) \quad \text{(Transitivity)}
\]
\[
(P \rightarrow Q) \Leftrightarrow (P \land \neg Q \rightarrow Q) \quad \text{(Split-Off-Skip)}
\]

Transient Properties (1/3)

Definition

- Borrowed from UNITY.
- The basic tool for reasoning about progress properties.
- $\text{tr } P$ states that always $P$ is eventually falsified.
- In LTL: $\Box \Diamond \neg P$.
- Important properties:

\[
\text{tr } P = T \rightarrow \neg P = P \rightarrow \neg P
\]

Transient Properties (2/3)

Definition

- Borrowed from UNITY.
- The basic tool for reasoning about progress properties.
- $\text{tr } P$ states that always $P$ is eventually falsified.
- In LTL: $\Box \Diamond \neg P$.
- Important properties:

\[
\text{tr } P = T \rightarrow \neg P = P \rightarrow \neg P
\]
Consider \( \text{tr} P = P \leadsto \neg P = \Box (P \Rightarrow \neg P) \).

Proof (Sketch).
Assume \( P \) holds in some state, we prove \( \Diamond \neg P \) by contradiction.

- Assume \( \Box P \).
- From (SCH), we have \( \Box C \).
- Together with (PRG), we have \( \Box \Diamond F \).
- Scheduling assumption ensures that \( e \) will eventually occur.
- (NEG) guarantees that when \( e \) occurs, \( P \) is falsified.
- We have a contradiction with the assumption from Step 1.

---

Abstract systems can simulate behaviours of concrete systems.

\[ \text{ex.} \text{cncM} \Rightarrow \text{ex.} \text{absM} \]

- Event-based reasoning.
  - \((\text{abs}_e)e \triangleq \text{any } t \text{ where } G \text{ during } C \text{ upon } F \text{ then } S \text{ end}\)
  - \((\text{cnc}_f)f \triangleq \text{any } t \text{ where } H \text{ during } D \text{ upon } E \text{ then } R \text{ end}\)

- Safety:
  - Guard strengthening: \( H \Rightarrow G \)
  - Action strengthening: \( R \Rightarrow S \)

- Liveness:
  - Liveness assumption strengthening.
  - Schedules weakening:
    \[ (\Box C \land \Diamond F) \Rightarrow (\Box D \land \Diamond E) \]

---

\( \text{prg}_0 \_1 \): \( t \in \text{trns} \leadsto t \notin \text{trns} \)

- \( \text{prg}_0 \_1 \) is the same as \( \text{tr} t \in \text{trns} \)
- (SCH) is trivial.
- No fine-schedule (\( F \) is \( \top \)) hence (PRG) is trivial.
- The event falsifies \( t \in \text{trns} \) (NEG)

---

\[ (\Box C \land \Box \Diamond F) \Rightarrow (\Box D \land \Box \Diamond E) \] (REF_LIVE)

- Practical rules:
  - Coarse-schedule following
    \[ C \land F \leadsto D \] (C_FLW)
  - Coarse-schedule stabilising
    \[ D \triangleright C \] (C_STB)
  - Fine-schedule following
    \[ C \land F \leadsto E \] (F_FLW)
A Signal Control System. The First Refinement

The State

- Introduce the network topology: **BLK**, **Entry**, **PLF**, **Exit**.
- Variable **loc** denotes location of trains in the network.

$$\text{inv1}_1 : \text{loc} \in \text{trns} \rightarrow \text{BLK}$$

- Variable **loc** denotes location of trains in the network.

The Ensure Rule

**Theorem (The ensure-rule)**

For all state predicates **p** and **q**,

$$(\text{P un Q}) \wedge (\text{tr p} \wedge \neg \text{Q}) \Rightarrow (\text{P} \Rightarrow \text{Q})$$  \hspace{1cm} (ENS)

- The 1st condition is implemented by event **movein** (not shown)
- The 2nd condition is implemented by event **moveout**
- We need the ensure rule (next slide).
A Signal Control System. The First Refinement

New Event moveout

\[ t \in \text{trns} \land \text{loc}.t \in \text{PLF} \implies t \in \text{trns} \land \text{loc}.t = \text{Exit} \]

Ensure rule

\[ t \in \text{trns} \land \text{loc}.t \in \text{PLF} \quad \text{un} \quad t \in \text{trns} \land \text{loc}.t = \text{Exit} \land \]

\[ (\text{tr} (t \in \text{trns} \land \text{loc}.t \in \text{PLF}) \land \neg (t \in \text{trns} \land \text{loc}.t = \text{Exit})) \]

\[ \text{moveout} \]

\[ \text{any} \quad t \quad \text{where} \]

\[ t \in \text{trns} \land \text{loc}.t \in \text{PLF} \]

\[ \text{during} \]

\[ t \in \text{trns} \land \text{loc}.t \in \text{PLF} \]

\[ \text{then} \]

\[ \text{loc}.t := \text{Exit} \]

\[ \text{end} \]

---

A Signal Control System. The Second Refinement

The State

SAF 1 There is at most one train on each block

\[ \forall t_1, t_2 \cdot t_1 \in \text{trns} \land t_2 \in \text{trns} \land \text{loc}.t_1 = \text{loc}.t_2 \implies t_1 = t_2 \]

---

A Signal Control System. The Third Refinement

The State

- Introduce the signals \( sgn \)

\[ \text{inv}_3.1 : \quad sgn \in \{ \text{Entry} \} \cup \text{PLF} \rightarrow \text{COLOR} \]

\[ \text{inv}_3.2 : \quad \forall p \cdot p \in \text{PLF} \land sgn.p = \text{GR} \implies \text{Exit} \notin \text{ran}.\text{loc} \]

\[ \text{inv}_3.3 : \quad \forall p, q, p, q \in \text{PLF} \land sgn.p = sgn.q = \text{GR} \implies p = q \]

- Neither weak- nor strong-fairness is satisfactory.
- Weak-fairness requires \( \text{Exit} \) to be free infinitely long.
- Strong-fairness is too strong assumption.
Refinement of moveout

(abs_)moveout

any  t  where
  t ∈ trns ∧ loc.t ∈ PLF ∧
  Exit ̸∈ ran.loc
  during
  t ∈ trns ∧ loc.t ∈ PLF
  upon
  Exit ̸∈ ran.loc
then
loc.t := Exit
end

Refinement of moveout

(cnc_)moveout

any  t  where
  t ∈ trns ∧ loc.t ∈ PLF ∧
  sgn.(loc.t) = GR
  during
  t ∈ trns ∧ loc.train ∈ PLF ∧
  sgn.(loc.t) = GR
then
loc.t := Exit
  sgn.(loc.t) := RD
end

Refinement requires to prove:

\( (\forall t \in trns \land (\forall p \in PLF \land p \notin ran.loc) \land Exit \notin ran.loc) \land sgn.(loc.t) = RD \)

Summary

Guarded and scheduled events.
Reasoning about liveness (progress) properties.
Refinement preserving safety and liveness properties.
Developments are guided by safety and liveness requirements.
Simon Hudon.  
A Progress Preserving Refinement.  
*Master Thesis.*  
Chair of Information Security, ETH Zurich, 2011.

Simon Hudon and Thai Son Hoang.  
Systems Design Guided by Progress Concerns.  
Accepted for *iFM 2013.*