UNITY Logic

(Based on the Modified Version in [Misra 1995])

Yih-Kuen Tsay

Department of Information Management
National Taiwan University

Introduction

UNITY was once quite popular. Its logic has been modified and improved in a subsequent work.

> *J. Misra. A logic for concurrent programming. Journal of Computer and Software Engineering, 3(2): 239-272, 1995.*

- A program consists of (1) an initial condition and (2) a set of actions (or conditional multiple-assignments), which always includes *skip*.
- Properties are defined in terms of
  - initially p,
  - p **co** q, and
  - p transient.

Program Model: Action System

- Syntax: An action system consists of
  - a set of *variables* and
  - a set of *actions*, always including *skip* (which does not change the system's state).

  A particular valuation of the variables is called a system or program *state*. An action is essentially a *guarded multiple assignment* to the variables.
- Semantics:
  - A system execution starts from some initial state and goes on forever.
  - In each step of an execution, some action is selected (under some fairness constraint) and executed, resulting in a possible change of the program state.

The "Contrains" Operator

- The safety properties of a system are stated using the "*contrains*" (**co**) operator.
- "p **co** q" (p constrains q) states that whenever p holds, q holds after the execution of any single action.
- Formally, $p \textbf{ co } q \text{ in } F \triangleq \langle \forall t : t \text{ in } F : \{p\}\ t\ \{q\} \rangle$.
- As *skip* may be applied in any state, from p **co** q it follows that $p \Rightarrow q$.
- It also follows that once p holds, q continues to hold upto (and including) the point where p ceases to hold (if it ever does).

Usages of the **co**

- "$x = 0$ **co** $x \geq 0$": once x becomes 0 it remains 0 until it becomes positive.
- "$\forall m :: x = m$ **co** $x \geq m$": x never decreases.
  This is equivalent to "$\forall m :: x \geq m$ **co** $x \geq m$".
- "$\forall m, n :: x, y = m, n$ **co** $x = m \vee y = n$": x and y never change simultaneously.

The *unless* Operator

- "p *unless* q" was introduced in the original UNITY logic as a basic safety property:

$$p \textit{ unless } q \text{ in } F \triangleq \forall t : t \text{ in } F : \{p \wedge \neg q\}\ t\ \{p \vee q\}$$

  If p is *true* at some point of computation, then it will continue to hold as long as q does not (q may never hold and p continues to hold forever).
- Example: "$x \geq k$ *unless* $x > k$" says that x is non-decreasing.
- p *unless* $q \equiv p \wedge \neg q$ **co** $p \vee q$.
- p **co** $q \equiv p$ *unless* $\neg p \wedge q$.

Special Cases of co

- p stable $\triangleq$ p **co** p
- p invariant $\triangleq$ (initially p) and (p stable)

Some Rules of Hoare Logic

$$\frac{}{\{p\}\ s\ \{true\}} \qquad \frac{}{\{false\}\ s\ \{q\}}$$

$$\frac{\{p\}\ s\ \{false\}}{\neg p}$$

$$\frac{\langle \forall j :: \{p_j\}\ s\ \{q_j\}\rangle}{\{\langle \forall j :: p_j\rangle\}\ s\ \{\langle \forall j :: q_j\rangle\}} \qquad \frac{\langle \forall j :: \{p_j\}\ s\ \{q_j\}\rangle}{\{\langle \exists j :: p_j\rangle\}\ s\ \{\langle \exists j :: q_j\rangle\}}$$

$$\frac{p \Rightarrow p',\ \{p'\}\ s\ \{q'\},\ q' \Rightarrow q}{\{p\}\ s\ \{q\}}$$

Derived Rules (Theorems)

A theorem in the form of

$$\frac{\Delta_1}{\Delta_2}$$

means that properties in Δ_2 can be deduced from properties in the premise Δ_1.

Some Derived Rules

- *false* **co** p.
- p **co** *true*.
- Conjunction and Disjunction

$$\frac{p \ \textbf{co} \ q, \ p' \ \textbf{co} \ q'}{\begin{array}{c} p \vee p' \ \textbf{co} \ q \vee q' \\ p \wedge p' \ \textbf{co} \ q \wedge q' \end{array}}$$

- Stable Conjunction and Disjunction

$$\frac{p \ \textbf{co} \ q, \ r \ \mathrm{stable}}{\begin{array}{c} p \wedge r \ \textbf{co} \ q \wedge r \\ p \vee r \ \textbf{co} \ q \vee r \end{array}}$$

The Substitution Axiom

An invariant may be replaced by *true*, and vice versa, in any property of a program.

- Example 1: given $p \;\; \mathbf{co} \;\; q$ and $J \;\; \mathrm{invariant}$, we conclude

$$p \wedge J \;\; \mathbf{co} \;\; q, \;\; p \;\; \mathbf{co} \;\; q \wedge J, \;\; p \wedge J \;\; \mathbf{co} \;\; q \wedge J, \;\; \text{etc.}$$

- Example 2:

$$\frac{p \;\; unless \;\; q, \; \neg q \; \mathrm{invariant}}{p \; \mathrm{stable}}$$

Note: there is a distinction between an invariant (of a particular program) and a valid formula (in any context). However, as part of a program property, they can be safely interchanged.

An Elimination Theorem

- Free variables may be eliminated by taking conjunctions or disjunctions.
- Suppose p a property that does not name any program variable other than x.
- Then, $p[x := m]$ does not contain any variable and is a constant (and hence stable).
- Observe that $p = \langle \exists m : p[x := m] : x = m \rangle$.
- An elimination theorem:

$$\frac{\begin{array}{c} x = m \;\; \textbf{co} \;\; q, \text{ where } m \text{ is free} \\ p \text{ does not name } m \text{ nor any program variable other than } x \end{array}}{p \;\; \textbf{co} \;\; \langle \exists m :: p[x := m] \wedge q \rangle}$$

An Elimination Theorem (cont.)

$$\frac{\begin{array}{c} x = m \;\; \textbf{co} \;\; q, \text{ where } m \text{ is free} \\ p \text{ does not name } m \text{ nor any program variable other than } x \end{array}}{p \;\; \textbf{co} \;\; \langle \exists m :: p[x := m] \wedge q \rangle}$$

Proof:

$x = m \;\; \textbf{co} \;\; q$, premise

$p[x := m] \wedge x = m \;\; \textbf{co} \;\; p[x := m] \wedge q$

, stable disjunction with $p[x := m]$

$\langle \exists m :: p[x := m] \wedge x = m \rangle \;\; \textbf{co} \;\; \langle \exists m :: p[x := m] \wedge q \rangle$

, disjuction over all m

$p \;\; \textbf{co} \;\; \langle \exists m :: p[x := m] \wedge q \rangle$, simplifying the lhs

Transient Predicate (under Weak Fairness)

- Under weak fairness, it is sufficient to have a single action falsify a transient predicate.
- p transient $\triangleq \langle \exists s :: \{p\}\ s\ \{\neg p\} \rangle$
- Some derived rules:

$$(p \;\; \text{stable} \;\; \wedge p \;\; \text{transient}) \equiv \neg p$$

(The only predicate that is both stable and transient is *false*.)

$$\frac{p \;\; \text{transient}}{p \wedge q \;\; \text{transient}}$$

Progress Properties

- $p \text{ ensures } q \triangleq (p \wedge \neg q \ \ \mathbf{co} \ \ p \vee q)$ and $p \wedge \neg q$ transient.
  If p holds at any point, it will continue to hold as long as q does not hold; eventually q holds.
- "$p \mapsto q$" specifies that if p holds at any point then q holds or will eventually hold. Inductive definition:

$$\frac{p \text{ ensures } q}{p \mapsto q}$$

$$\text{(transitivity)} \quad \frac{p \mapsto q, q \mapsto r}{p \mapsto r}$$

$$\text{(disjunction)} \quad \frac{\langle \forall m : m \in W : p(m) \mapsto q \rangle}{\langle \exists m : m \in W : p(m) \rangle \mapsto q}$$

Example: "$x \geq k \mapsto x > k$" says that x will eventually increase.

Some Derived Rules for Progress

- (Progress-Safety-Progress, PSP)

$$\frac{p \mapsto q,\ r \ \ \mathbf{co} \ \ s}{(p \wedge r) \mapsto (q \wedge s) \vee (\neg r \wedge s)}$$

- (well-founded induction)

$$\frac{\langle \forall m :: p \wedge M = m \mapsto (p \wedge M < m) \vee q \rangle}{p \mapsto q}$$

Asynchronous Composition

- Notation: $F \mathbin{[\!]} G$ (the *union* of F and G)
- Semantics:
  - The set of variables is the *union* of the two sets of variables.
  - The set of actions is the *union* of the two sets of actions.
  - The composed system is executed as a single system.

UNITY Logic vs. Lamport's 'Hoare Logic'

- "**co**" enjoys the complete rule of consequence.
- Rules of conjunction and disjunction also hold.
- Stronger rule of parallel composition:

$$\frac{p \ \mathbf{co} \ q \text{ in } F, \ \ p \ \mathbf{co} \ q \text{ in } G}{p \ \mathbf{co} \ q \text{ in } F \parallel G}$$

- But, "**co**" is much less convenient for sequential composition.

Union Theorems

- $$\frac{p \textit{ unless } q \text{ in } F, \;\; p \;\text{ stable }\; \text{in } G}{p \textit{ unless } q \text{ in } F \parallel G}$$
- $$\frac{p \;\text{ invariant }\; \text{in } F, \;\; p \;\text{ stable }\; \text{in } G}{p \;\text{ invariant }\; \text{in } F \parallel G}$$
- $$\frac{p \textit{ ensures } q \text{ in } F, \;\; p \;\text{ stable }\; \text{in } G}{p \textit{ ensures } q \text{ in } F \parallel G}$$
- If any of the following properties holds in F, where p is a local predicate of F, then it also holds in $F \parallel G$ for any G: p *unless* q, p *ensures* q, p invariant.

Note: Any invariant used in applying the substitution axiom to deduce a property of one module should be proved an invariant in the other module.

References

- J. Misra. *A Discipline of Multiprogramming*, Springer, 2001.
- J. Misra. A logic for concurrent programming. *Journal of Computer and Software Engineering*, 3(2): 239-272, 1995.
- M. Chandy and J. Misra. *Parallel Program Design: A Foundation*, Addison-Wesley, 1988.