CS 515: Specification and Verification with Traces

Stefan D. Bruda

Winter 2019

Property-Oriented Specifications

- The specification $S(tr)$ consists of a predicate on traces
- We then say that a process P satisfies a specification $S(tr)$ iff all the traces of P sasisfy S:

$$P \textbf{ sat } S(tr) = \forall\, tr \in \text{traces}(P) : S(tr)$$

- If it is not the case that P **sat** $S(tr)$ then this must be because S fails to hold on a particular, finite trace
  - There must be a point where a particular event violates the specification
  - Conversely, for a specification to hold, it must be the case that no violations happen at any point in the execution of the process
  - Thus trace specifications are all safety specifications (nothing bad will even happen)
  - Other kind of specifications (such as liveness specifications) are both possible and useful, but they cannot be specified within the trace semantics
- Note that $S(\langle\rangle)$ must always hold, else S is vacuous because no process can ever satisfy it
  - Conversely, *STOP* satisfies all possible specifications
  - That is, in terms of trace specifications *STOP* is the safest process

Property-Oriented Verification

- Compositional process
- Three general rules:

$$\frac{}{P \textbf{ sat } true(tr)} \qquad \frac{P \textbf{ sat } S(tr) \quad P \textbf{ sat } T(tr)}{P \textbf{ sat } (S \wedge T)(tr)}$$

weakening: $\dfrac{P \textbf{ sat } S(tr)}{P \textbf{ sat } T(tr)}$ $\left[\forall\, tr \in TRACE : S(tr) \Rightarrow T(tr)\right]$

Property-Oriented Verification

- Compositional process
- Three general rules:

$$\frac{}{P \textbf{ sat } true(tr)} \qquad \frac{P \textbf{ sat } S(tr) \quad P \textbf{ sat } T(tr)}{P \textbf{ sat } (S \wedge T)(tr)}$$

weakening: $\dfrac{P \textbf{ sat } S(tr)}{P \textbf{ sat } T(tr)} \; [\forall\, tr \in TRACE : S(tr) \Rightarrow T(tr)]$

- Strongest specification for *STOP*:

$$\frac{}{STOP \textbf{ sat } tr = \langle\rangle}$$

Can be weakened using the following rule:

$$\frac{STOP \textbf{ sat } tr = \langle\rangle}{STOP \textbf{ sat } S(tr)} \; [\forall\, tr \in TRACE : tr = \langle\rangle \Rightarrow S(tr)]$$

PREFIX, *SKIP*, *RUN*

- Prefix: $$\frac{P \textbf{ sat } S(tr)}{a \to P \textbf{ sat } tr = \langle\rangle \vee \text{head}(tr) = a \wedge S(\text{tail}(tr))}$$
- Prefix choice: $$\frac{\forall\, a \in A : P(a) \textbf{ sat } S_a(tr)}{x : A \to P(x) \textbf{ sat } tr = \langle\rangle \vee \exists\, a \in A : \text{head}(tr) = a \wedge S_a\text{tail}(tr)}$$
- Output: $$\frac{P \textbf{ sat } S(tr)}{c!v \to P \textbf{ sat } tr = \langle\rangle \vee \text{head}(tr) = c.v \wedge S(\text{tail}(tr))}$$
- Input: $$\frac{\forall\, c \in T : P(v) \textbf{ sat } S_v(tr)}{c?x : T \to P(x) \textbf{ sat } tr = \langle\rangle \vee \exists\, v \in T : \text{head}(tr) = c.v \wedge S_v(\text{tail}(tr))}$$

PREFIX, *SKIP*, *RUN*

- Prefix: $$\frac{P \textbf{ sat } S(tr)}{a \to P \textbf{ sat } tr = \langle\rangle \vee \text{head}(tr) = a \wedge S(\text{tail}(tr))}$$
- Prefix choice: $$\frac{\forall\, a \in A : P(a) \textbf{ sat } S_a(tr)}{x : A \to P(x) \textbf{ sat } tr = \langle\rangle \vee \exists\, a \in A : \text{head}(tr) = a \wedge S_a\text{tail}(tr)}$$
- Output: $$\frac{P \textbf{ sat } S(tr)}{c!v \to P \textbf{ sat } tr = \langle\rangle \vee \text{head}(tr) = c.v \wedge S(\text{tail}(tr))}$$
- Input: $$\frac{\forall\, c \in T : P(v) \textbf{ sat } S_v(tr)}{c?x : T \to P(x) \textbf{ sat } tr = \langle\rangle \vee \exists\, v \in T : \text{head}(tr) = c.v \wedge S_v(\text{tail}(tr))}$$
- *SKIP*: $$\frac{}{SKIP \textbf{ sat } tr = \langle\rangle \vee tr = \langle\checkmark\rangle}$$
- *RUN*: $$\frac{}{RUN \textbf{ sat } true(tr)}$$
  - Superfluous rule (already covered earlier)

Choice, Parallel

- External choice: $\dfrac{P \textbf{ sat } S(tr) \quad Q \textbf{ sat } T(tr)}{P \,\Box\, Q \textbf{ sat } S(tr) \vee T(tr)}$ $\dfrac{\forall\, i \in I : P_i \textbf{ sat } S_i(tr)}{\Box_{\,i \in I} P_i \textbf{ sat } \exists\, i \in I : S_i(tr)}$

- Internal choice:

Choice, Parallel

- External choice: $\dfrac{P \textbf{ sat } S(tr) \quad Q \textbf{ sat } T(tr)}{P \,\Box\, Q \textbf{ sat } S(tr) \vee T(tr)}$ $\quad \dfrac{\forall i \in I : P_i \textbf{ sat } S_i(tr)}{\Box_{i \in I} P_i \textbf{ sat } \exists i \in I : S_i(tr)}$
- Internal choice: $\dfrac{P \textbf{ sat } S(tr) \quad Q \textbf{ sat } T(tr)}{P \,\sqcap\, Q \textbf{ sat } S(tr) \vee T(tr)}$ $\quad \dfrac{\forall i \in I : P_i \textbf{ sat } S_i(tr)}{\sqcap_{i \in I} P_i \textbf{ sat } \exists i \in I : S_i(tr)}$
- Alphabetized parallel:

$$\frac{P \textbf{ sat } S(tr) \quad Q \textbf{ sat } T(tr)}{P \;{}_A\|_B\; Q \textbf{ sat } S(tr \upharpoonright A) \wedge T(tr \upharpoonright B) \wedge \sigma(tr) \subseteq (A \cup B)^{\checkmark}}$$

- Indexed parallel: trivial generalization
- Interface parallel: see textbook
  - Also see the weakening rule similar to the one for interleaving

Interleaving, Hiding, Renaming

- Interleaving: $$\frac{P \textbf{ sat } S(tr) \quad Q \textbf{ sat } T(tr)}{P \,|||\, Q \textbf{ sat } \exists\, tr_1, tr_2 : S(tr_1) \wedge T(tr_2) \wedge tr \text{ interleaves } tr_1, tr_2}$$
- The following weakening rule (common property) also useful in practice:
$$\frac{\begin{array}{l} P \textbf{ sat } S(tr) \\ Q \textbf{ sat } T(tr) \\ \forall\, tr, tr_1, tr_2 \in TRACE : S(tr_1) \wedge T(tr_2) \wedge tr \text{ interleaves } tr_1, tr_2 \Rightarrow R(tr) \end{array}}{P \,|||\, Q \textbf{ sat } R(tr)}$$
- Hiding: $$\frac{P \textbf{ sat } S(tr)}{P \setminus A \textbf{ sat } \exists\, tr_1 : tr_1 \setminus A = tr \wedge S(tr_1)}$$
  - Often useful: $$\frac{P \textbf{ sat } S(tr)}{P \setminus A \textbf{ sat } S(tr)} \quad [\forall\, tr : S(tr) \text{ iff } S(tr \setminus A)]$$
- Renaming: $$\frac{P \textbf{ sat } S(tr)}{f(P) \textbf{ sat } \exists\, tr_1 : S(tr_1) \wedge f(tr_1) = tr} \qquad \frac{P \textbf{ sat } S(tr)}{f^{-1}(P) \textbf{ sat } S(f(tr))}$$

Sequential Composition, Interrupt

- Sequential Composition:

$$\frac{P \textbf{ sat } S(tr) \qquad Q \textbf{ sat } T(tr)}{P; Q \textbf{ sat } \neg term(tr) \wedge S(tr) \vee \exists\, tr_1, tr_2 : tr = tr_1 \frown tr_2 \wedge S(tr_1 \frown \langle \checkmark \rangle) \wedge T(tr_2)}$$

- Interrupt:

$$\frac{P \textbf{ sat } S(tr) \qquad Q \textbf{ sat } T(tr)}{P \triangle Q \textbf{ sat } S(tr) \vee \exists\, tr_1, tr_2 : tr = tr_1 \frown tr_2 \wedge \neg term(tr_1) \wedge S(tr_1) \wedge T(tr_2)}$$

- Recursion induction: For a process recursively defined as $N = F(N)$,

$$\frac{\forall\, Y : Y \textbf{ sat } S(tr) \Rightarrow F(Y) \textbf{ sat } S(tr)}{N \textbf{ sat } S(tr)} \; [S(\langle\rangle)]$$

  - System of equations for mutual recursion

Process-Oriented Specification

- We provide the specification as a CSP process describing acceptable behaviour (model)
- Acceptable defined in terms of traces: the traces of the specification must all be encountered in the implementation:

$$S \sqsubseteq_T I = \text{traces}(I) \subseteq \text{traces}(S)$$

  - The implementation I is a refinement of the specification S iff any trace of I is "allowed" by S
  - $\sqsubseteq_T$ also called an implementation relation; it is reflexive, transitive, and anti-symmetric
  - $=_T$ definable in terms if $\sqsubseteq_T$ as usual: $P =_T Q$ iff $P \sqsubseteq_T Q \wedge Q \sqsubseteq_T P$
  - $\sqsubseteq_T$ also definable in terms of $=_T$: $P \sqsubseteq_T Q$ iff $P =_T P \sqcap Q$
  - All CSP operators are monotonic with respect to refinement ($P \sqsubseteq_T Q \Rightarrow F(P) \sqsubseteq_T F(Q)$)
  - Further laws for refinement:

$RUN \sqsubseteq_T P$	($\sqsubseteq_T$-bottom)
$P \sqsubseteq_T STOP$	($\sqsubseteq_T$-top)
P_0 **sat** $S(tr) \wedge P_0 \sqsubseteq_T P_1 \Rightarrow P_1$ **sat** $S(tr)$	($\sqsubseteq_T$-spec)

Model-Based May Testing

- We called earlier the testing framework model-based testing because tests can be algorithmically generated form a model (or process-oriented specification)
- In case of may testing this is supported by the following:

Theorem

*For any test T there exists an equivalent set of sequential may tests TS all of form $a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n \rightarrow SUCCESS$ such that P **may** T iff*
$\exists\, t \in TS : P$ ***may*** t

- Proof is based on the traces of T
- Create a sequential test out of each successful trace of T and call the resulting set TS

Model-Based May Testing

- We called earlier the testing framework model-based testing because tests can be algorithmically generated form a model (or process-oriented specification)
- In case of may testing this is supported by the following:

Theorem

*For any test T there exists an equivalent set of sequential may tests TS all of form $a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n \rightarrow SUCCESS$ such that P **may** T iff $\exists\, t \in TS : P$ **may** t*

  - Proof is based on the traces of T
  - Create a sequential test out of each successful trace of T and call the resulting set TS

- Traces and the two refinement relations are all equivalent:

Theorem

$\sqsubseteq_{may}$, $\sqsubseteq_T$ and trace inclusion are all equivalent with each other