

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 satisfy S :

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

- If it is not the case that $P \text{ sat } S(tr)$ 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:

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

weakening: $\frac{P \text{ sat } S(tr)}{P \text{ sat } T(tr)} [\forall tr \in TRACE : S(tr) \Rightarrow T(tr)]$

- Strongest specification for **STOP**:

$$\overline{STOP \text{ sat } tr = \langle \rangle}$$

Can be **weakened** using the following rule:

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

PREFIX, SKIP, RUN



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



- **External choice:**
$$\frac{P \text{ sat } S(tr) \quad Q \text{ sat } T(tr)}{P \sqcap Q \text{ sat } S(tr) \vee T(tr)} \quad \frac{\forall i \in I: P_i \text{ sat } S_i(tr)}{\sqcap_{i \in I} P_i \text{ sat } \exists i \in I: S_i(tr)}$$
- **Internal choice:**
$$\frac{P \text{ sat } S(tr) \quad Q \text{ sat } T(tr)}{P \sqcap Q \text{ sat } S(tr) \vee T(tr)} \quad \frac{\forall i \in I: P_i \text{ sat } S_i(tr)}{\sqcap_{i \in I} P_i \text{ sat } \exists i \in I: S_i(tr)}$$
- **Alphabetized parallel:**
$$\frac{P \text{ sat } S(tr) \quad Q \text{ sat } T(tr)}{P \parallel_B Q \text{ sat } S(tr \upharpoonright A) \wedge T(tr \upharpoonright B) \wedge \sigma(tr) \subseteq (A \cup B)^\vee}$$
- **Indexed parallel:** trivial generalization
- **Interface parallel:** see textbook
 - Also see the weakening rule similar to the one for interleaving



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



- **Sequential Composition:**
$$\frac{P \text{ sat } S(tr) \quad Q \text{ sat } T(tr)}{P; Q \text{ sat } \neg \text{term}(tr) \wedge S(tr) \vee \exists tr_1, tr_2: tr = tr_1 \frown tr_2 \wedge S(tr_1 \langle \checkmark \rangle) \wedge T(tr_2)}$$
- **Interrupt:**
$$\frac{P \text{ sat } S(tr) \quad Q \text{ sat } T(tr)}{P \triangle Q \text{ sat } S(tr) \vee \exists tr_1, tr_2: tr = tr_1 \frown tr_2 \wedge \neg \text{term}(tr_1) \wedge S(tr_1) \wedge T(tr_2)}$$
- **Recursion induction:** For a process recursively defined as $N = F(N)$, $\forall Y: Y \text{ sat } S(tr) \Rightarrow F(Y) \text{ sat } S(tr)$ $[S(\langle \rangle)]$
 - System of equations for mutual recursion



- 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 of \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:

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



- We called earlier the testing framework model-based testing because tests can be algorithmically generated from 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 $a_1 \rightarrow a_2 \rightarrow \dots \rightarrow a_n \rightarrow \text{SUCCESS}$ such that $P \text{ may } T$ iff $\exists t \in TS : P \text{ may } t$

- 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

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