Proposal for 1381

SV-AC.

May 31, 2006

Section 17.11.2

remove

If there is no match of the antecedent `sequence_expr` from a given start point, then evaluation of the implication from that start point succeeds ~~vacuously~~ and returns true.

Section 17.13.3

remove

~~Vacuity rules are applied only when implication operator is used. A property succeeds non-vacuously only if the consequent of the implication contributes to the success.~~

add

Section 17.13.6 Non-Vacuous Evaluation

An evaluation attempt of a property is either vacuous or non-vacuous. In the following, non-vacuous evaluation is described for the seven kinds of property: sequence, negation, disjunction, conjunction, if...else, implication, and instantiation.

a) An evaluation attempt of a property that is a sequence is always non-vacuous.
b) An evaluation attempt of a property of the form `not property_expr` is non-vacuous iff the underlying evaluation attempt of `property_expr` is non-vacuous.
c) An evaluation attempt of a property of the form `property_expr1 or property_expr2` is non-vacuous iff either the underlying evaluation attempt of `property_expr1` is non-vacuous or the underlying evaluation attempt of `property_expr2` is non-vacuous.
d) An evaluation attempt of a property of the form `property_expr1 and property_expr2` is non-vacuous iff either the underlying evaluation attempt of `property_expr1` is non-vacuous or the underlying evaluation attempt of `property_expr2` is non-vacuous.
e) An evaluation attempt of a property of the form `if ( expression_or_dist ) property_expr1` is non-vacuous iff expression_or_dist evaluates to `true` and the underlying evaluation attempt of `property_expr1` is non-vacuous.

   An evaluation attempt of a property of the form `if ( expression_or_dist ) property_expr1 else property_expr2` is non-vacuous iff either expression_or_dist evaluates to `true` and the underlying evaluation attempt of `property_expr1` is non-vacuous, or expression_or_dist evaluates to `false` and the underlying evaluation attempt of `property_expr2` is non-vacuous.
f) An evaluation attempt of a property of the form `sequence_expression |-> property_expr` is non-vacuous iff there is a successful match of the antecedent `sequence_expression` and the evaluation attempt of `property_expr1` that starts at the end point of the match is non-vacuous.

   An evaluation attempt of a property of the form `sequence_expression |=> property_expr` is non-vacuous iff there is a successful match of the antecedent `sequence_expression` and the evaluation attempt of `property_expr1` that starts at the clock event following the end point of the match is non-vacuous.
g) An evaluation attempt of an instance of a property is non-vacuous iff the underlying evaluation attempt of the `property_expr` that results from substituting actual arguments for formal arguments is non-vacuous.

An evaluation attempt of a property of the form `disable iff (expression_or_dist) property_expr` is non-vacuous iff the underlying evaluation attempt of `property_expr` is non-vacuous and expression_or_dist does not become true during this evaluation.

An evaluation attempt of a property succeeds non-vacuously iff the property evaluates to true and the evaluation attempt is non-vacuous.

Section 17.13.4

---

add

A concurrent assertion statement can be used outside of a procedural context. It can be used within a module, an interface, or a program. A concurrent assertion statement is an **assert**, an **assume**, or a **cover** statement. Such a concurrent assertion statement uses the always semantics, meaning that it specifies

that a new evaluation attempt of the underlying *property_spec* begins at every occurrence of its leading clock event.

Section 17.13.5

---

modify

Another inference made from the context is the enabling condition for a property. Such derivation takes place when a property is placed in an **if...else** block or a **case** block. ~~The enabling condition assumed from the context is used as the antecedent of the property.~~ A concurrent assertion embedded in procedural code specifies that a new evaluation attempt of the underlying *property_spec* begin at every occurrence of the inferred clocking event.

Appendix E

---

add

Appendix E.3.3.3 Vacuity

This subclause defines the relation of non-vacuity, denoted $\models^{non}$, between a word w and a property P. An evaluation of P on w is non-vacuous provided $w \models^{non} P$.

- Base:
  - Let R be a sequence, then $w \models^{non} R$.
- Induction:
  - $w \models^{non} (P_1)$ iff $w \models^{non} P_1$.
  - $w \models^{non} R$`|->`P_1 iff there exists $i \geq 0$, such that $w^{0..i} \models R$ and $w^{i..} \models^{non} P_1$.
  - $w \models^{non} P_1$ `and` P_2 iff $w \models^{non} P_1$ or $w \models^{non} P_2$.
  - $w \models^{non} P_1$ `or` P_2 iff $w \models^{non} P_1$ or $w \models^{non} P_2$.
  - $w \models^{non}$ `not` P_1 iff $w \models^{non} P_1$.
  - $w, L \models^{non}$ `disable iff`$(b)P_1$ iff $w \models^{non} P_1$ and one of the following holds:

1. for every $0 \leq i < |w|$, $w^i \not\models b$.
2. For some prefix $x \leq w$, we have that for every $0 \leq i < |x|$, $x^i \not\models b$, and either $x \cdot \bot^\omega \models P$ or $x \cdot \top^\omega \not\models P$.

A word w satisfies property P non-vacuously iff $w \models P$ and $w \models^{\text{non}} P$.

---

add

Appendix E.3.6.3 Vacuity

The definition is identical to that without local variables, but with the understanding that the underlying properties can have local variables.