Actions in SVA

Doron Bustan and John Havlicek

Freescale Semiconductor, Inc.

January 23, 2006

1 Actions

Given an SVA property P, an instance of an action α that is placed in P, and a word w. We need to define in which cycles α should be executed, and if it is executed in a given cycle, how many copies of the execution there should be. This documents considers only the first question: In which cycles the action should be executed?

In this section we present some examples, which demonstrate the different issues. In the reset of this document we will consider a single attempt, which starts at the begining of the computation. For simplicity we use sequences in most of the examples. First, we look at some simple examples. For a cover property `cover property (@ (posedge c)` R `##0 (1,` α`) )`, where R is a sequence. Let w be a word and L_0 be a local variable context. Then, we expect α to be executed after the first prefix w' of w such that $w', L_0, L_1 \models R$ for some L_1. Similarly for a property of the form `cover property (@ (posedge c)` R_1 `##0 (1,` α`) ##1` R_2`)`, we expect α to be executed after every prefix w' of w such that $w', L_0, L_1 \models R_1$ for some L_1 as long as the evaluation of the property is ongoing. Note that R_2 does not describes when α should be executed, but does effect the termination of the evaluation of the overall property.

Thus, when we have a strict partition of the property where one part describe the computation before the action execution and the other describes the computation after the execution, we can ignore the second part in the decision when α should be executed. Other operators that keep the partition of before/after simple are the `first match` sequence operator and, the `|->`, `not`, and `disable iff` property operators.

In some cases α should be execute in more then one cycle, for example in `assert property (@(posedge clk) (a ##1 a ##0 (1,` α`)) [*1:$] ##1 b ##1 b)`. α may be executed after multiple matches of `a ##1 a [*1:$]` as long as they are not followed by a match to `b ##1 b`.

Thus, a condition for an execution of an action α after a prefix w' is that the overall assertion/cover property is still being evaluated at w'. More precisely, an assertion is being evaluated at a cycle i as long as it has not failed and has not succeed before the end of w'. We discuss the definition of

"has not failed yet" and "has not succeed yet" in details later.

1.0.1 Actions inside repetition

Consider the property `cover property (@ (posedge c) (a ##1 a ##0 (1,` α`) ##0 b ##1 b) [*1:$] ##1 d)` . We expect α to be executed after the prefix w' of w such that $w', L_0, L_1 \models$ `a##1a`. However, we also expect α to be executed after prefixes such that $w', L_0, L_1 \models$ `(a ##1 a ##0 (1,` α`) ##0 b ##1 b) [*1:$] ##1 a ##1 a` as long as the property is being evaluated. In general, for every property of the form `cover property (@(posedge c)` R_1`##1(`R_2 `##0 (1,`α`) ##0` R_3`) [*1:$] ##1` R_4`)`, α is executed after matches of the sequence R_1 `##1 (`R_2 `##0 (1) ##0` R_3`) [*0:$]##1` R_2.

1.0.2 Parallel sequences

Until now we consider only "single thread" sequences, which do not include the `intersect`, `and`, `or` operators. For subsequences that are being evaluated in parallel to the action execution, the definition of "has not failed yet" and "has not succeed yet become more complicated. Consider the property `assert property (@(posedge clk)(a ##1 (1,` α`) ##1 b) intersect c[*3])`, we expect α to be executed after the second cycle, provided that $w^0 \models a$. However, what if $w^0 \not\models c$? Should the subsequence `c[*3]`, terminate the evaluation of the sequence `(`a `##1 (1,` α`) ##1` b`)`? One approach the *tight approach* is saying that once the two sequences are put together with the `intersect` operator, they are no longer independent entities. Thus, the evaluation of R should be done in one piece and a failure of `c[*3]` at cycle 1 should terminate the evaluation of `(`a `##1 (1,` α`) ##1` b`)` without executing α.

The other approach, the *loose approach* is saying that the sequences are independent, and the result of their evaluations should be conjoined only after they end.

There are examples that support the tight approach, and others that support the loose approach. For example, consider the a sequence R_{send}, which identifies that a message was sent, and a sequence R_{ack}, which identifies an acknowledgment. The property `assert property (@(posedge clk)` R_{send} `|-> (`R_{ack} `##0 (1,` α`) intersect 1[*0:10]))`, requires that for every message there should be acknowledge that ends after at most 10 cycles. In this case we would not expect α to be executed if the acknowledgment ends after more than 10 cycles. For this example, we prefer the tight approach.

In another example consider a sequence R_{send}, which identifies that a message was sent to two addresses, and the sequences R_{ack1}, and R_{ack2}, which identify acknowledgments from address 1 and address 2 respectively. Then, the property `assert property (@(posedge clk)`R_{send} `|-> ((`R_{ack1} `##0`

(1, α)) and (R_{ack2} ##0 (1, β)))) requires that for every message, both addresses will acknowledge. In this example, we would like the two sub sequences "R_{ack1} ##0 (1, α)" and "R_{ack2} ##0 (1, β)", to be evaluated independently, including an independent execution of α and β.

The tight approach requires a more complicated definition. For properties of the form P_1 and P_2 where α is placed in P_1, we need to decide at every cycle, whether P_2 has failed by that cycle. For a property of the form assert property (@(posedge clk) P_1 or P_2)) where α is placed in P_1, α should be executed according to its place in P_1, provided that P_2 has not **succeed** by that time. For sequences, a success of a match, does not terminates other success, because they may have a different length, or a different local variable context flowing out. Note, that we cannot simply check that the overall assertion has not failed/succeeded yet. For example consider a property of the form assert property (@(posedge) ((P_1 and P_2) or P_3)) where α is placed in P_1. Then, for every prefix we need to check that P_2 has not failed yet, even if the overall property has not failed yet because of P_3.

To summarize the examples above, the decision of when an action should be execute depends on two definitions:

1. The strict prefix before the action. This part describes the computation before the action execution. We need to define prefixes after which the action may be executed.

2. The parts of the properties that describe when the action's thread is still being evaluated. This definition is different for the tight and loose approaches.

We need to define the relations *has not failed yet* and *has not succeed yet* for properties, this definition is used in the second part. In this document we define both the tight approach and the loose approach. A number of combinations should be considered:

1. A key word which determine for each assertion which approach should be taken.

2. The loose approach, with the exception that when the top level property succeeds/fails all evaluations are being terminated.

3. Since properties cannot be joined, it may make sense to use the tight approach for sequences, and the loose approach for properties.

The rest of this document continues as follows: First, we introduce the pre_action function. This function maps sequences and properties in which an instance of an action is placed into new sequences, such that the resulted

sequence matches the prefixes after which the action may be executed. This function ignores the evaluation termination of the action's tread. Next, we define the relations $\models^{\leftarrow S}$ that defines which prefix has not succeeded yet, and the relation $\models^{\leftarrow F}$ that defines which prefixes has not failed yet. Then, we define in the functions live_tight and live_loose which map sequences and properties in which an instance of an action is placed to sequences, such that the resulted sequence matches exactly the prefixes after which the action's thread is still being evaluated according to the tight/loose appraoch respectively. finally, we present two proposals, for the tight and loose approaches.

1.1 The pre_action function

We first define the pre_action function for sequences. The pre_action function maps a sequence R and an instance α of an action placed in R, to a new sequence R' such that when R is evaluated with respect to a word w, α may executed at the end of the prefixes w' that tightly satisfy R'. Since an action that is placed in a sequence of the form R`[*0]` is not executed, we assume that in the non-derived form, the action is not placed in this form of a sequence.

Definition 1.1. *Let R be a sequence and let α be an action placed inside R. we define an* SVA *sequence* pre_action(R, α) *as follows:*

- pre_action(`(1,` α), α)=`1`.
- pre_action(b, α) *is not defined because no action is placed inside b.*
- pre_action(`(` R `)`, α)=pre_action(R, α).
- pre_action(`(` R_1 `##1` R_2 `)`, α)= pre_action(R_1, α) *if α is placed in R_1 and* `(` R_1 `##1` pre_action(R_2, α) `)` *otherwise.*
- pre_action(`(` R_1 `##0` R_2 `)`, α) = pre_action(R_1, α) *if α is placed in R_1 and* `(` R_1 `##0` pre_action(R_2, α) `)` *otherwise.*
- pre_action(`(` R_1 `or` R_2 `)`, α) = pre_action(R_1, α) *if α is placed in R_1 and* pre_action(R_2, α) *otherwise.*
- pre_action(`(` R_1 `intersect` R_2 `)`, α) = pre_action(R_1, α) *if α is placed in R_1 and* pre_action(R_2, α) *otherwise.*
- pre_action(`first_match (` R `)`, α) = `first_match (` pre_action(R, α) `)`.
- pre_action(R `[*1:$]`, α)= pre_action(R, α) `or (` R `[*1:$] ##1` pre_action(R, α) `)`.

We extend the pre_action function for properties. Thus, the pre_action function maps a property P and an instance α of an action placed in P to a sequence R' such that when P is evaluated with respect to a word w, α is executed exactly at the end of the prefixes w' that tightly satisfy R'.

- pre_action(`disable iff ( b )` P, α) = pre_action(P, α).
- pre_action(`not` P, α) = pre_action(P, α).
- pre_action(`(` R `|->` P `)`, α) = pre_action(R, α) if α is placed in R, and pre_action(`(` R `|->` P `)`, α) = R `##0` pre_action(P, α), if α is placed in P.
- pre_action(`(` P_1 `or` P_2 `)`, α) = pre_action(P_1, α) if α is placed in P_1 and pre_action(P_2, α) otherwise.
- pre_action(`(` P_1 `and` P_2 `)`, α) = pre_action(P_1, α) if α is placed in P_1 and pre_action(P_2, α) otherwise.

1.2 The $\models^{\leftarrow S}$ and $\models^{\leftarrow F}$ relations

The $\models^{\leftarrow S}$ and $\models^{\leftarrow F}$ relations capture the notion of *has not succeed yet* and *has not failed yet* respectively. For a property P, a word w and a local variable context L_0, we have that $w^{0..k}, L_0 \models^{\leftarrow F} P$ iff the evaluation of P over the word w and local variable context L_0 has not fail at the first $k+1$ cycles. Similarly, $w^{0..k}, L_0 \models^{\leftarrow S} P$ iff the evaluation of P over the word w and local variable context L_0 has not succeed at the first $k+1$ cycles.

We start with an auxiliary relation $\models^{\leftarrow}$ such that $w^{0..k}, L_0 \models^{\leftarrow} R$ iff for some L_1 the evaluation of R with respect to w, L_0, L_1 has not fail at the first $k+1$ cycles.

Definition 1.2. *Let R be a sequence and L_0 be a local variable context, we define a relation $\models^{\leftarrow}$ as follows:*

- w, $L_0 \models^{\leftarrow}$ `(` `1,` α `)` *iff* $w = \epsilon$ *or* $w, L_0, L_1 \models$ `1` *for some* L_1.
- w,$L_0 \models^{\leftarrow} b$ *iff* $w = \epsilon$ *or* $w, L_0, L_1 \models b$ *for some* L_1.
- w, $L_0 \models^{\leftarrow}$ `(` R `)` *iff* w, $L_0 \models^{\leftarrow} R$.
- w, $L_0 \models^{\leftarrow}$ `(` R_1 `##1` R_2 `)` *iff* w, $L_0 \models^{\leftarrow} R_1$ *or for some* $xy = w$ *and local variable context* L' *we have,* $x, L_0, L' \models R_1$ *and* y, $L' \models^{\leftarrow} R_2$.
- w, $L_0 \models^{\leftarrow}$ `(` R_1 `##0` R_2 `)` *iff either* $w = \epsilon$, *or* $w\top$, $L_0 \models^{\leftarrow} R_1$ *or for some* $xyz = w$ *such that* $|y| = 1$ *and a local variable context* L', *we have that* $xy, L_0, L' \models R_1$ *and* $yz, L' \models^{\leftarrow} R_2$.
- w, $L_0 \models^{\leftarrow}$ `(` R_1 `or` R_2 `)` *iff* w, $L_0 \models^{\leftarrow} R_1$ *or* w, $L_0 \models^{\leftarrow} R_2$.

- $w, L_0 \models^{\leftarrow} \texttt{(}\ R_1\ \texttt{intersect}\ R_2\ \texttt{)}$ *iff* $w, L_0 \models^{\leftarrow} R_1$ *and* $w, L_0 \models^{\leftarrow} R_2$.
- $w, L_0 \models^{\leftarrow} \texttt{first\_match (}\ R\ \texttt{)}$ *iff* $w, L_0 \models^{\leftarrow} R$ *and for every* $xy = w$ *and* L_1, *such that* $|y| > 0$, *we have* $\bar{x}, L_0, L_1 \not\models R$
- $w, L_0 \models^{\leftarrow} R\ \texttt{[*0]}$ *iff* $w, L_0, L_0 \models R\ \texttt{[*0]}$
- $w, L_0 \models^{\leftarrow} R\ \texttt{[*1:\$]}$ *iff* $w, L_0 \models^{\leftarrow} R$ *or for some* $xy = w$ *and local variable context* L' *we have* $x, L_0, L' \models R\ \texttt{[*1:\$]}$ *and* $y, L' \models^{\leftarrow} R$.

Properties: If $w, L_0 \models^{\leftarrow S} P$ *then the property has not succeed yet. If* $w, L_0 \models^{\leftarrow F} P$ *then the property has not failed yet. We define the* $\models^{\leftarrow P}$ *relation such that* $w, L_0 \models^{\leftarrow P} P$ *iff* $w, L_0 \models^{\leftarrow S} P$ *and* $w, L_0 \models^{\leftarrow F} P$.

- $w, L_0 \models^{\leftarrow S} R$ *iff for every non empty prefix* x *of* w *and local variable context* L_1 *we have* $x, L_0, L_1 \not\models R$

  $w, L_0 \models^{\leftarrow F} R$ *iff either* $w, L_0 \models^{\leftarrow} R$ *or there exists a non empty prefix* x *of* w *and* L_1 *such that* $w, L_0, L_1 \models R$.
- $w, L_0 \models^{\leftarrow F} R\ \texttt{|->}\ P$ *iff for every* $xyz = w$, $|y| = 1$, *and* L' *if* $\bar{xy}, L_0, L' \models R$ *then* $yz, L' \models^{\leftarrow F} P$.

  $w, L_0 \models^{\leftarrow S} R\ \texttt{|->}\ P$ *iff* $w\top, L_0 \models^{\leftarrow} R$ *or for some* $xyz = w$, $|y| = 1$ *and* L', *we have that* $\bar{xy}, L_0, L' \models R$, *and* $yz, L_0 \models^{\leftarrow S} P$
- $w, L_0 \models^{\leftarrow S} P_1\ \texttt{and}\ P_2$ *iff* $w, L_0 \models^{\leftarrow S} P_1$ *or* $w, L_0 \models^{\leftarrow S} P_2$.

  $w, L_0 \models^{\leftarrow F} P_1\ \texttt{and}\ P_2$ *iff* $w, L_0 \models^{\leftarrow F} P_1$ *and* $w, L_0 \models^{\leftarrow F} P_2$.
- $w, L_0 \models^{\leftarrow S} P_1\ \texttt{or}\ P_2$ *iff* $w, L_0 \models^{\leftarrow S} P_1$ *and* $w, L_0 \models^{\leftarrow S} P_2$.

  $w, L_0 \models^{\leftarrow F} P_1\ \texttt{or}\ P_2$ *iff* $w, L_0 \models^{\leftarrow F} P_1$ *or* $w, L_0 \models^{\leftarrow F} P_2$.
- $w, L_0 \models^{\leftarrow S} \texttt{not}\ P$ *iff* $\bar{w}, L_0 \models^{\leftarrow F} P$.

  $w, L_0 \models^{\leftarrow F} \texttt{not}\ P$ *iff* $\bar{w}, L_0 \models^{\leftarrow S} P$.
- $w, L_0 \models^{\leftarrow F} \texttt{disable iff (}b\texttt{)}\ P$ *iff either* $w, L_0 \models^{\leftarrow F} P$ *or there exists* $0 \leq k \leq |w|$ *such that* $w^{0..k}, L_0 \models^{\leftarrow F} P$ *and* $w^k \models b$.

  $w, L_0 \models^{\leftarrow S} \texttt{disable iff (}b\texttt{)}\ P$ *iff* $w, L_0 \models^{\leftarrow S} P$ *and for all* $0 \leq k \leq |w|$ *s.t.* $w^k \models b$ *we have that* $w^{0..k}, L_0 \not\models^{\leftarrow F} P$.

1.3 Discussion of the $\models^{\leftarrow}$, $\models^{\leftarrow S}$, $\models^{\leftarrow F}$, and the $\models^{\leftarrow P}$ definitions

The notion has not fail yet and has not succeed yet are already defined for PSL and SVA, in the definition of the `disable iff` operator. There, an evaluation of a property P with respect to local variable context L_0 and a word w has not fail in the first $k+1$ cycles iff $w^{0..k}\top^{\omega}, L_0 \models P$. We find

this definition un intuitive when the `intersect` operator is involved. For example for the property P = (1 `##1` 1 `##1` 1) `intersect` (1 `##1` 1), the sub sequences do not fail until cycle 3 (according to both definitions), but according to the definition that based on $\top^\omega$, the property P fail before the first cycle. The new definition fail P only at cycle 3.

The following lemmas describe the relationships between the two definitions, the lemmas are proved in Appendix A.

Lemma 1.3. *Let R be an* SVA *sequence, let L_0 be a local variable context, and let w be a word, then if there are w', L_1 such that $ww', L_0, L_1 \models R$, then w, $L_0 \models^{\leftarrow} R$.*

Lemma 1.4. *Let P be an* SVA *property without the* `disable iff` *operator, let L_0 be a local variable context, and let w be a word, then if there w', L_1 such that $ww', L_0, L_1 \not\models P$, then w, $L_0 \models^{\leftarrow S} P$, and if there w', L_1 such that $ww', L_0, L_1 \models P$, then w, $L_0 \models^{\leftarrow F} R$.*

To understand the problem with the `disable iff` operator, let us look at the property `disable iff` (b) 1 `intersect` (1 `##1` 1) and a word w such that $|w| = 1$ and $w^0 \models b$. Then, for every w' and L_0 in particular $w' = \epsilon$, we have that $ww' \not\models$ `disable iff` (b) 1 `intersect` (1 `##1` 1). However, $w^0 \models b$ and $w, L_0 \models^{\leftarrow F}$ 1 `intersect` (1 `##1` 1). Thus, $w, L_0 \not\models^{\leftarrow S}$ `disable iff` (b) 1 `intersect` (1 `##1` 1).

The source of the problem is that in the `disable iff` operator, the notion of "P has not failed yet on w" is captured by $w\top^\omega \models P$. This definition fails the property 1 `intersect` (1 `##1` 1) on the first letter of w although non of the sub sequences fails there. We see this as unintuitive, and that why according to the definition of $\models^{\leftarrow F}$ the sequence 1 `intersect` (1 `##1` 1) fails only in the second letter.

The lemma holds when we use the following definition for the `disable iff` operator.: $w, L_0 \models$ `disable iff (`b`)` P iff either $w, L_0 \models P$ or there exists $0 \leq k < |w|$ such that $w^k \models b[L_0]$ and $w^{0,k-1}, L_0 \models^{\leftarrow F} P$.

Lemma 1.5. *Let R be an* SVA *sequence without the* `intersect` *and* `##0` *operators, let L_0 be a local variable context, and let w be a word. Then w, $L_0 \models^{\leftarrow} R$ implies that there are $k \geq 0$, L_1 such that $w\top^k, L_0, L_1 \models R$.*

The problem with the `intersect` operator was already discussed. To understand the problem with the the `##0` operator consider the property P = `[*0] ##0 [*0]`. Let $w = \epsilon$ and L_0 a local variable context. Then, since $w, L_0 \models^{\leftarrow}$ `[*0]`, we have that $w, L_0 \models^{\leftarrow} P$. However, for every $k \geq 0$ and L_1 we have that $w\top^k, L_0, L_1 \not\models P$.

Lemma 1.6. *Let P be a non degenerate* SVA *property without the* `intersect` *and* `##0` *operators, let L_0 be a local variable context, and let w be a word. If w, $L_0 \models^{\leftarrow S} P$, then $w\bot^\omega, L_0, \not\models P$. If w, $L_0 \models^{\leftarrow F} R$ then $w\top^\omega, L_0, \models P$.*

1.4 The live_tight function

The live_tight and the live_tight$_P$ functions characterizes the prefixes in which the "action's thread" is still under evaluation according to the tight approach.

Definition 1.7. *Let w be a word such that $|w| > 0$ then* pre$(w) = w^{0..(|w|-2)}$.

In other words pre(w) is w minus the last letter.

We define a new SVA sequence unary operator `LV` such that $w, L_0, L_1 \models$ `LV`R iff pre(w), $L_0 \models^{\leftarrow} R$. Similarly, we define $w, L_0, L_1 \models$ `LVS`P iff pre(w), $L_0 \models^{\leftarrow S} P$, $w, L_0, L_1 \models$ `LVF`P iff pre(w), $L_0 \models^{\leftarrow F} P$, $w, L_0, L_1 \models$ `LVP`P iff pre(w), $L_0 \models^{\leftarrow S} P$ and pre(w), $L_0 \models^{\leftarrow F} P$.

Definition 1.8. *Let R be a sequence and let α be an action placed inside R. we define an* SVA *sequence* live_tight(R, α) *as follows:*

- live_tight(`(1,` α`)`, α)=`1`.
- live_tight(b, α) *is not defined because no action is placed inside b.*
- live_tight(`(` R `)`, α)=live_tight(R, α).
- live_tight(`(` R_1 `##1` R_2 `)`, α)= live_tight(R_1, α)`##1` R_2 *if α is placed in R_1 and* `(` R_1 `##1` live_tight(R_2, α) `)` *otherwise.*
- live_tight(`(` R_1 `##0` R_2 `)`, α) = `(` live_tight(R_1, α) `##0` R_2 `)` *if α is placed in R_1 and* `(` R_1 `##0` live_tight(R_2, α) `)` *otherwise.*
- live_tight(`(` R_1 `or` R_2 `)`, α) = live_tight(R_1, α) *if α is placed in R_1 and* live_tight(R_2, α) *otherwise.*
- live_tight(`(` R_1 `intersect` R_2 `)`, α) = `(` live_tight(R_1, α) `intersect LV` R_2 `)` *if α is placed in R_1 and* `(LV` R_1 `intersect` live_tight(R_2, α) *otherwise.*
- live_tight(`first_match (` R `)`, α) = `first_match (` live_tight(R, α) `)`.
- live_tight(R `[*1:$]`, α)= live_tight(R, α) `or (` R `[*1:$] ##1` live_tight(R, α) `)`.

Definition 1.9. *Let P be a property and let α be an action placed inside P. we define an* SVA *sequence* live_tight$_P(P, \alpha)$ *as follows:*

- live_tight$_P$(R , α) = `(LV` live_tight(R, α) `) intersect LVP` R.
- live_tight$_P$(`(` R `|->` P `)`, α) = live_tight(R, α) `intersect LVP (`R `|->` P `)` *if α is placed in R, and* live_tight$_P$(`(` R `|->` P `)`, α) = *(*R `##0` live_tight$_P(P, \alpha)$*)* `intersect LVP(`R `|->` P `)`, *if α is placed in P.*

- live_tight$_P$((P_1 `or` P_2), α) = (live_tight$_P$(P_1, α) `intersect LVS` P_2), *if* α *is placed in* P_1 *and* live_tight$_P$((P_1 `or` P_2), α) = (live_tight$_P$(P_2, α) `intersect LVS` P_1), *if* α *is placed in* P_2.

- live_tight$_P$((P_1 `intersect` P_2), α) = (live_tight$_P$(P_1, α) `intersect LVF` P_2), *if* α *is placed in* P_1 *and* live_tight$_P$((P_1 `intersect` P_2), α) = (live_tight$_P$(P_2, α) `intersect LVF` P_1), *if* α *is placed in* P_2.

- live_tight$_P$(`not` P, α) =live_tight$_P$(P, α).

- live_tight$_P$(`disable iff (` b `)` P, α) = live_tight$_P$(P, α) `intersect SVP disable iff (` b `)` P.

1.5 The live_loose function

The live_loose and the live_loose$_P$ functions characterizes the prefixes in which the "action's thread" is still under evaluation according to the loose approach.

Definition 1.10. *Let R be a sequence and let α be an action placed inside R. we define an* SVA *sequence* live_loose(R, α) *as follows:*

- live_loose(`(1,` α`)`, α)=`1`.

- live_loose(b, α) *is not defined because no action is placed inside b.*

- live_loose(`(` R `)`, α)=live_loose(R, α).

- live_loose(`(` R_1 `##1` R_2 `)`, α)= live_loose(R_1, α)`##1` R_2 *if* α *is placed in* R_1 *and* `(` R_1 `##1` live_loose(R_2, α) `)` *otherwise.*

- live_loose(`(` R_1 `##0` R_2 `)`, α) = `(` live_loose(R_1, α) `##0` R_2 `)` *if* α *is placed in* R_1 *and* `(` R_1 `##0` live_loose(R_2, α) `)` *otherwise.*

- live_loose(`(` R_1 `or` R_2 `)`, α) = live_loose(R_1, α) *if* α *is placed in* R_1 *and* live_loose(R_2, α) *otherwise.*

- live_loose(`(` R_1 `intersect` R_2 `)`, α) = `(` live_loose(R_1, α) *if* α *is placed in* R_1 *and* live_loose(R_2, α) *otherwise.*

- live_loose(`first_match (` R `)`, α) = `first_match (` live_loose(R, α) `)`.

- live_loose(R `[*1:$]`, α)= live_loose(R, α) `or (` R `[*1:$] ##1` live_loose(R, α) `)`.

Definition 1.11. *Let P be a property and let α be an action placed inside P. we define an* SVA *sequence* live_loose$_P$(P, α) *as follows:*

- live_loose$_P$(R , α) = `(` `LV` live_loose(R, α) `)` `intersect LVP` R.

- live_loose$_P$((R |-> P), α) = live_loose(R, α) intersect LVP (R |-> P) *if* α *is placed in* R, *and* live_loose$_P$((R |-> P), α) = (R ##0 live_loose$_P$(P, α)) intersect LVP(R |-> P), *if* α *is placed in* P.

- live_loose$_P$((P_1 or P_2), α) = (live_loose$_P$(P_1, α), *if* α *is placed in* P_1 *and* live_loose$_P$((P_1 or P_2), α) = (live_loose$_P$(P_2, α) *if* α *is placed in* P_2.

- live_loose$_P$((P_1 intersect P_2), α) = (live_loose$_P$(P_1, α), *if* α *is placed in* P_1 *and* live_loose$_P$((P_1 intersect P_2), α) = live_loose$_P$(P_2, α), *if* α *is placed in* P_2.

- live_loose$_P$(not P, α) =live_loose$_P$(P, α).

- live_loose$_P$(disable iff (b) P, α) = live_loose$_P$(P, α) intersect SVP disable iff (b) P.

Proposal 1 (tight): Let P be a property, α be an action, w be a word, and L_0 be a local variable context to the local variables flowing into P. Then, the action should be execute iff there exists L_1 such that $w, L_0, L_1 \equiv$ (pre_action(P, α) and live_tight$_P$(P, α) and LVP P).

Proposal 2 (loose): Let P be a property, α be an action, w be a word, and L_0 be a local variable context to the local variables flowing into P. Then, the action should be execute iff there exists L_1 such that $w, L_0, L_1 \equiv$ (pre_action(P, α) and live_loose$_P$(P, α) and LVP P).

2 examples

1. pre_action(((a ##1 ((($1, \alpha$)##1 c)[*1:$])##1 d), α)=
   pre_action((a ##1 ((($1, \alpha$)##1 c)[*1:$]), α)=
   (a ##1 pre_action((((1, α)##1 c)[*1:$]), α))=
   (a ##1 ((1##1 c)[*0:$]) ##1 pre_action(($(1, \alpha)$ ##1 c), α)))=
   (a ##1 ((1##1 c)[*0:$]) ##1 pre_action($(1, \alpha)$, α)))=
   (a ##1 (((1##1 c)[*0:$]) ##1 1))

   For this example, we have live_tight((a ##1 ((($1, \alpha$)##1 c)[*1:$])##1 d), α) = live_loose$_P$((a ##1 ((($1, \alpha$)##1 c)[*1:$])##1 d), α) = LVP (a ##1 ((1##1 c)[*1:$])##1 d) = $\{w \mid w$ match (a ##1 ((1##1 c)[*1:$])##1 d), and for every $xy = w$, we have that x does not match (a ##1 ((1##1 c)[*1:$])##1 d)$\}$

2. $R_1 = (1$ ##1 $1)$[*1:$]. $R_2 = 1$ ##1 $(1$ ##1 $1)$[*1:$]. $R_3 = R_1$ intersect R_2. $P = R_3$ |-> (1,α).

   For every w, L_0 we have that $w, L_0 \equiv^{\leftarrow} R_1$.

For every w, L_0 we have that $w, L_0 |\equiv^{\leftarrow} R_2$.

For every w, L_0 we have that $w, L_0 |\equiv^{\leftarrow} R_3$.

For every w, L_0 we have that $w, L_0 \models^{\leftarrow S} R_3$ `|->1`.

For every w, L_0 we have that $w, L_0 \models^{\leftarrow F} R_3$ `|->1`.

pre_action(P, α) = R_3 `##0` 1.

live_tight$_P$(P, α) = live_loose$_P$(P, α) = R_3 `##0` 1.

For every w, L_0 we have that $w, L_0 \not\equiv R_3$`##0`1.

Thus α is never executed.

3. R_1 = (1 `##1` (1, α))`[*1:$]`. R_2 = 1 `##1` (1 `##1` 1)`[*1:$]`. R_3 = R_1 `intersect` R_2. P = R_3 `|->` 1.

   For every w, L_0 we have that $w, L_0 |\equiv^{\leftarrow} R_1$.

   For every w, L_0 we have that $w, L_0 |\equiv^{\leftarrow} R_2$.

   For every w, L_0 we have that $w, L_0 |\equiv^{\leftarrow} R_3$.

   For every w, L_0 we have that $w, L_0 \models^{\leftarrow S} R_3$ `|->1`.

   For every w, L_0 we have that $w, L_0 \models^{\leftarrow F} R_3$ `|->1`.

   pre_action(P, α) = (1 `##1` 1)`[*1:$]`

   live_tight$_P$(P, α) = `(` live_tight(R_3, α) `##0` 1) `intersect` `LVP` P. Since for every w and L_0 we have that $w, L_0 \models$ `LVP` P, this is equal to live_tight(R_3, α) `##0` 1 = `(` live_tight(R_1, α) `intersect` `LV` R_2 `)` . Since for every w and L_0 we have that $w, L_0 |\equiv$ `LV` R_2, this is equal to live_tight(R_1, =) (1 `##1` 1)`[*1:$]`.

   Thus, α should be execute at all even cycles.

4. P = 1 `or` (1 `##1` (1, α));

   pre_action(P, α) = (1 `##1` 1).

   For every word w with and L_0 we have that $w, L_0 |\equiv$ pre_action(P, α) iff $|w| = 2$.

   For every word w and local variable context L_0 we have that $w, L_0 \models P$ iff $|w| = 1$.

   For every word w with and L_0 we have that $w, L_0 \models^{\leftarrow P} P$ iff $|w| \leq 1$.

   Thus, α is never executed.

5. P = (1,α)`[*1:$]` `##1` 0.

   pre_action(P, α) = (1,α)`[*1:$]`.

   live_tight$_P$(P, α) = 1 `[*1:$]` `##1` 0.

Since for every word w and assignment L_0 we have that $w, L_0| \equiv^{\leftarrow} 1$`[*1:$]##10`, α should be execute at every cycle.

6. $P = (a$ `[*1:$]` $)$ `|->` $(1$ `##1` $(1,\alpha)$ `##1` $1)$.

   $\mathsf{pre\_action}(P, \alpha) = (a$ `[*1:$]` $)$ `##0` 1 `##1` 1.

   $\mathsf{live\_tight}_P(P, \alpha) = \mathsf{live\_loose}_P(P, \alpha) = (a$ `[*1:$]` $)$ `##0` 1 `##1` 1 `##1` 1.

   Let w be a word of length 8 such that a holds exactly at the first 3 letters. Then for every L_0 we have that only the first three nonempty prefixes w' of w have $w', L_0| \equiv^{\leftarrow} a$`[*1:$]`.

   Every prefix w' and L_0 satisfies that $w', L_0 \models^{\leftarrow F} P$.

   The first 7 prefixes w' and all L_0 satisfy $w', L_0 \models^{\leftarrow S} P$ and the last 2 prefixes satisfy $w', L_0 \not\models^{\leftarrow S} P$

   Thus α is executed at cycles 2,3,4.

7. $P = (a$ `[*1:$]` $)$ `|->` $((1$ `##1` (b,α) `##1` $1)$ `and` $\mathrm{c})$.

   $\mathsf{pre\_action}(P, \alpha) = (a$ `[*1:$]` $)$ `##0` 1 `##1` b.

   $\mathsf{live\_tight}_P(P, \alpha) = (a$ `[*1:$]` $)$ `##0` $((1$ `##1` b `##1` $1)$ `and LVP` $\mathrm{c})$.

   $\mathsf{live\_loose}_P(P, \alpha) = (a$ `[*1:$]` $)$ `##0` 1 `##1` b `##1` 1.

8. $P = (1$ `|->` $1)$ `or` $(1$ `##1` $(1, \alpha))$.

   $\mathsf{pre\_action}(P, \alpha) = (1$ `##1` $(1, \alpha))$.

   For every word w and local variable context L_0, we have $w, L_0 \models^{\leftarrow F} 1$ `|->` 1. For every local variable context L_0 we have that only $w = \epsilon$ satisfies $w, L_0 \models^{\leftarrow S} 1$ `|->` 1.

   Thus, α is never executed.

A proofs

Lemma 1.3 Let R be an SVA sequence, let L_0 be a local variable context, and let w be a word, then if there are w', L_1 such that $ww', L_0, L_1| \equiv R$, then w, $L_0 \models^{\leftarrow} R$.

Proof: By induction on R:

- Base:
  - $ww', L_0, L_1 \models$ `( 1, v = e )`. Then, $|ww'| = 1$. Thus, either $w = \epsilon$ or $|w| = 1$ and $w, L_0, L_1| \equiv R$ by the definition of $|\equiv^{\leftarrow}$, in both cases w, $L_0 \models^{\leftarrow} R$.
  - $ww', L_0, L_1 \models b$ Then, $|ww'| = 1$. Thus, either $w = \epsilon$ or $|w| = 1$ and $w, L_0, L_1| \equiv R$ by the definition of $|\equiv^{\leftarrow}$, in both cases w, $L_0 \models^{\leftarrow} R$.

- Induction:
  - $ww', L_0, L_1 \models$ `(` R `)` iff $ww', L_0, L_1 \models R$. By induction w, $L_0 \models^{\leftarrow} R$, and thus w, $L_0 \models^{\leftarrow}$ `(`R`)`
  - $ww', L_0, L_1 \models$ `(` R_1 `##1` R_2 `)` iff there exist x, y, L' such that $ww' = xy$ and $x, L_0, L' \models R_1$ and $y, L', L_1 \models R_2$. we distinguish between two cases:
    1. If w is a prefix of x, then let w'' be such that $ww'' = x$. By induction w, $L_0 \models^{\leftarrow} R_1$ and thus w, $L_0 \models^{\leftarrow}$ `(`R_1 `##1` R_2`)`.
    2. Otherwise, x is a prefix of w. Let $w = xw''$, then $w''w' = y$. By induction, w'', $L' \models^{\leftarrow} R_2$, thus w, $L_0 \models^{\leftarrow}$ `(`R_1 `##1` R_2`)`.
  - $ww', L_0, L_1 \models$ `(` R_1 `##0` R_2 `)` iff there exist x, y, z, L' such that $ww' = xyz$ and $|y| = 1$, and $xy, L_0, L' \models R_1$ and $yz, L', L_1 \models R_2$.
    we distinguish between two cases:
    1. If w is a prefix of xy, then let w'' be such that $ww'' = xy$. By induction w, $L_0 \models^{\leftarrow} R_1$ and thus w, $L_0 \models^{\leftarrow}$ `(`R_1 `##0` R_2`)`.
    2. Otherwise, xy is a prefix of w. Let $w = xw''$, then $w''w' = yz$. By induction, w'', $L' \models^{\leftarrow} R_2$, thus w, $L_0 \models^{\leftarrow}$ `(`R_1 `##0` R_2`)`.
  - $ww', L_0, L_1 \models$ `(` R_1 `or` R_2 `)`. Then, there exists L' such that either $ww', L_0, L' \models R_1$ or $ww', L_0, L' \models R_2$. By induction either w, $L_0 \models^{\leftarrow} R_1$ or w, $L_0 \models^{\leftarrow} R_2$, thus, w, $L_0 \models^{\leftarrow}$ `(`R_1 `or` R_2`)`.
  - $ww', L_0, L_1 \models$ `(` R_1 `intersect` R_2 `)`. Then, there exist L', L'' such that $ww', L_0, L' \models R_1$ and $ww', L_0, L'' \models R_2$. By induction w, $L_0 \models^{\leftarrow} R_1$ and w, $L_0 \models^{\leftarrow} R_2$, thus, w, $L_0 \models^{\leftarrow}$ `(`R_1 `intersect` R_2`)`.
  - $ww', L_0, L_1 \models$ `first_match (` R `)`. Then,
    - $ww', L_0, L_1 \models R$ and thus w, $L_0 \models^{\leftarrow} R$.
    - if there exist x, y, L' such that $ww' = xy$ and $\bar{x}, L_0, L' \models R$, then y is empty. Thus for every strict prefix x of w we have that $x, L_0, L_1 \not\models R$.

    and thus w, L_0, $\models^{\leftarrow}$`first_match(`R`)`.
  - $ww', L_0, L_1 \models R$ `[*0]`. Then $w = \epsilon$ and thus w, $L_0 \models^{\leftarrow} R$ `[*0]`.
  - $ww', L_0, L_1 \models R$ `[*1:$]` iff there exist $L_{(0)} = L_0, w_1, L_{(1)}, w_2, L_{(2)} \ldots, w_j, L_{(j)} = L_1$ $(j \geq 1)$ such that $ww' = w_1 w_2 \cdots w_j$ and for every i such that $1 \leq i \leq j$, $w_i, L_{(i-1)}, L_{(i)} \models R$. We distinguish between two cases:
    1. If w is a prefix of w_1, then let w'' be such that $ww'' = w_1$. By induction w, $L_0 \models^{\leftarrow} R$ and thus w, $L_0 \models^{\leftarrow} R$ `[*1:$]`.
    2. Otherwise, let i be the maximal integer such that $w_1 w_2 \ldots w_i$ is a strict prefix of w. Let $ww'' = w_1 w_2 \ldots w_i w_{i+1}$, and let w''' be such that $w_1 w_2 \ldots w_i w''' = w$. Then $w'''w'' = w_{i+1}$. By induction, w''', $L_{(i)} \models^{\leftarrow} R$, thus w, $L_0 \models^{\leftarrow} R_1$ `[*1:$]`.

Lemma A.1. *Let $w, L_0, L_1 \models\equiv R$. If w' results from w by changing zero or more letters to $\top$, then $w', L_0, L_1 \models\equiv R$.*

Proof: By induction.

- Base:
  - b. $w, L_0, L_1 \models\equiv b$ iff $|w| = 1$ and $w^0 \models b[L_0]$ and $L_1 = L_0$. This implies that $|w'| = 1$ and $(w')^0 \models b[L_0]$ and $L_1 = L_0$ iff $w', L_0, L_1 \models\equiv b$
  - $(1, v = e)$. $w, L_0, L_1 \models\equiv (1, v = e)$ iff $|w| = 1$ and $w^0 \models 1$ and $L_1 = L_0|_{\mathrm{dom}(L_0)-v} \cup (v, e[L_0, w^0])$. [if $(w')^0 = \top$, then by our convention $L_1 = L_0|_{\mathrm{dom}(L_0)-v} \cup (v, e[L_0, (w')^0])$] $|w'| = 1$ and $(w')^0 \models 1$ and $L_1 = L_0|_{\mathrm{dom}(L_0)-v} \cup (v, e[L_0, (w')^0])$ iff $w', L_0, L_1 \models\equiv (1, v = e)$.
- Induction:
  - (R_1). $w, L_0, L_1 \models\equiv (R_1)$ iff $w, L_0, L_1 \models\equiv R_1$ By induction $w', L_0, L_1 \models\equiv R_1$ iff $w', L_0, L_1 \models\equiv (R_1)$.
  - $(R_1 \texttt{\#\#1} R_2)$. $w, L_0, L_1 \models\equiv (R_1 \texttt{\#\#1} R_2)$ iff there exist x, y, L s.t. $w = xy$ and $x, L_0, L \models\equiv R_1$ and $y, L, L_1 \models\equiv R_2$. By induction there exist x', y', L s.t. $w' = x'y'$ and $x', L_0, L \models\equiv R_1$ and $y', L, L_1 \models\equiv R_2$ iff $w', L_0, L_1 \models\equiv (R_1 \texttt{\#\#1} R_2)$
  - $(R_1 \texttt{\#\#0} R_2)$. $w, L_0, L_1 \models\equiv (R_1 \texttt{\#\#0} R_2)$ iff there exist x, y, z, L s.t. $w = xyz$ and $|y| = 1$ and $xy, L_0, L \models\equiv R_1$ and $yz, L, L_1 \models\equiv R_2$. By induction there exist x', y', z', L s.t. $w' = x'y'z'$ and $|y'| = 1$ and $x'y', L_0, L \models\equiv R_1$ and $y'z', L, L_1 \models\equiv R_2$ iff $w', L_0, L_1 \models\equiv (R_1 \texttt{\#\#0} R_2)$
  - $(R_1 \texttt{or} R_2)$. $w, L_0, L_1 \models\equiv (R_1 \texttt{or} R_2)$ iff there exists L s.t. both

    1. Either $w, L_0, L \models\equiv R_1$ or $w, L_0, L \models\equiv R_2$, and
    2. $L_1 = L|_{flow(\mathrm{dom}(L_0), (R_1 \texttt{or} R_2))}$

    By induction both

    1. Either $w', L_0, L \models\equiv R_1$ or $w', L_0, L \models\equiv R_2$, and
    2. $L_1 = L|_{flow(\mathrm{dom}(L_0), (R_1 \texttt{or} R_2))}$

    iff $w', L_0, L_1 \models\equiv (R_1 \texttt{or} R_2)$
  - $(R_1 \texttt{intersect} R_2)$. $w, L_0, L_1 \models\equiv (R_1 \texttt{intersect} R_2)$ iff there exist L', L'' s.t. $w, L_0, L' \models\equiv R_1$ and $w, L_0, L'' \models\equiv R_2$ and $L_1 = L'|_{D'} \cup L''|_{D''}$, where $D' = flow(\mathrm{dom}(L_0), R_1) - (block((R_1 \texttt{intersect} R_2)) \cup sample(R_2))$ $D'' = flow(\mathrm{dom}(L_0), R_2) - (block((R_1 \texttt{intersect} R_2)) \cup sample(R_1))$. By Induction, $w', L_0, L' \models\equiv R_1$ and $w', L_0, L'' \models\equiv R_2$ iff $w', L_0, L_1 \models\equiv (R_1 \texttt{intersect} R_2)$
  - `first_match` (R_1). $w, L_0, L_1 \models\equiv \texttt{first\_match}(R_1)$ iff both

    1. $w, L_0, L_1 \models\equiv R_1$, and
    2. If there exist x, y, L s.t. $w = xy$ and $\bar{x}, L_0, L \models\equiv R_1$, then y is empty.

By induction $w., L_0, L_1 | \equiv R_1$. For every $x'y' = w'$, if there exist x', y', L s.t. $x'y' = w'$ and $\bar{x'}, L_0, L | \equiv R_1$, then, since a word with $\perp$ as letter cannot tightly satisfy, $w' = w$, hence $|y'| = |y| = 0$ both

1. $w', L_0, L_1 | \equiv R_1$, and

2. If there exist x', y', L s.t. $w' = x'y'$ and $\bar{x'}, L_0, L | \equiv R_1$, then y' is empty iff $w', L_0, L_1 | \equiv \texttt{first\_match}(R_1)$

- $R\texttt{[*0]}$ $w, L_0, L_1 | \equiv R\texttt{[*0]}$ iff $|w| = 0$ and $L_1 = L_0$. Thus, $|w'| = 0$ and $L_1 = L_0$ iff $w', L_0, L_1 | \equiv R\texttt{[*0]}$
- $R\texttt{[*1:\$]}$. $w, L_0, L_1 | \equiv R_1\texttt{[*1:\$]}$ iff there exist $L_{(0)} = L_0, w_1, L_{(1)}, w_2, ..., w_j, L_{(j)} = L_1 (j \geq 1)$ s.t. $w = w_1 w_2 ... w_j$ and for every i s.t. $1 \leq i \leq j$, $w_i, L_{(i-1)}, L_{(i)} | \equiv R_1$. By induction there exist $L_{(0)} = L_0, w'_1, L_{(1)}, w'_2, ..., w'_j, L_{(j)} = L_1 (j \geq 1)$ s.t. $w' = w'_1 w'_2 ... w'_j$ and for every i s.t. $1 \leq i \leq j$, $w'_i, L_{(i-1)}, L_{(i)} | \equiv R_1 iff w', L_0, L_1 | \equiv R_1\texttt{[*1:\$]}$.

Lemma A.2. *Let $w, L_0 \models P$. If w' results from w by changing zero or more letters to $\top$, then $w', L_0 \models P$.*

Let $w, L_0 \not\models P$. If w'' results from w by changing zero or more letters to $\perp$, then $w'', L_0 \not\models P$.

Proof: By induction.

- Base:
  - For property R, we have that $w, L_0 \models R$ iff for some $0 < k < |w|$ and L_1, $w^{0..k}, L_0, L_1 | \equiv R$.

    If $w, L_0 \models R$, then Lemma A.1, implies that $(w')^{0..k}, L_0, L_1 | \equiv R$, thus $w', L_0 \models R$.

    If $w, L_0 \not\models R$, then for every $0 < k < |w''|$ we have that either $(w'')^{0..k} = w^{0..k}$ in which case for every L_1 $(w'')^{0..k}, L_0, L_1 \not| \equiv R$, or $(w'')^{0..k}$ contains a $\perp$ in which case Lemma A.4 implies that $(w'')^{0..k}, L_0, L_1 \not| \equiv R$. Thus $w'', L_0 \not\models R$.

- Induction:
  - $w, L_0 \models \texttt{disable iff (}b\texttt{)}\ P$ iff either $w, L_0 \models P$ or there exists $0 \leq k < |w|$ such that $w^k \models b[L_0]$ and $w^{0,k-1}\top^\omega, L_0 \models P$.

    If $w, L_0 \models P$, then by induction $w', L_0 \models P$ and thus, $w', L_0 \models \texttt{disable iff (}b\texttt{)}\ P$.

    If there exists $0 \leq k < |w|$ such that $w^k \models b[L_0]$ and $w^{0,k-1}\top^\omega, L_0 \models P$, then by induction $(w')^k \models b[L_0]$ and $(w')^{0,k-1}\top^\omega, L_0 \models P$ and thus, $w', L_0 \models \texttt{disable iff (}b\texttt{)}\ P$.

    If neither $w, L_0 \models P$ nor there exists $0 \leq k < |w|$ such that $w^k \models b[L_0]$ and $w^{0,k-1}\top^\omega, L_0 \models P$. Then, by induction $w'', L_0 \not\models P$ and for every

$0 \leq k < |w|$ such that $(w'')^k \models b[L_0]$, we have that $w^k \models b[L_0]$, thus $w^{0,k-1}\top^\omega, L_0 \not\models P$ and by induction $(w'')^{0,k-1}\top^\omega, L_0 \not\models P$. Thus, $w'', L_0 \not\models$ `disable iff (`b`)` P.

- $w, L_0 \models$ `not` P iff $\bar{w}, L_0 \not\models P$.

  If $w, L_0 \models$ `not` P, then $\bar{w}, L_0 \not\models P$. By Induction $\bar{w'}, L_0 \not\models P$, and thus $w', L_0 \models$ `not` P.

  If $w, L_0 \not\models$ `not` P, then $\bar{w}, L_0 \models P$. By Induction $\bar{w''}, L_0 \models P$, and thus $w'', L_0 \not\models$ `not` P.

- $w, L_0 \models$ `(` R `|->` P `)` iff for every $0 \leq j < |w|$ and L_1 such that $\bar{w}^{0,j}, L_0, L_1 \models R_1$, $w^{j\cdot\cdot}, L_1 \models P$.

  If $w, L_0 \models$ `(` R `|->` P `)`. Let $0 \leq j < |w|$ and L_1 be such that $(\bar{w'})^{0,j}, L_0, L_1 \models R$, then by Lemma A.4 $\bar{w'}^{0,j} = \bar{w}^{0..j}$, thus $w^{j\cdot\cdot}, L_1 \models P$ and by induction $(w')^{j\cdot\cdot}, L_1 \models P$. This implies that $w', L_0 \models$ `(` R `|->` P `)`.

  If $w, L_0 \not\models$ `(` R `|->` P `)`, then there exist $0 \leq j < |w|$ and L_1 such that $\bar{w}^{0,j}, L_0, L_1 \models R$ and $w^{j\cdot\cdot}, L_1 \not\models P$. Lemma A.1 implies that $(\bar{w''})^{0,j}, L_0, L_1 \models R$, and by induction $(w'')^{j\cdot\cdot}, L_1 \not\models P$. Thus $w'', L_0 \not\models$ `(` R `|->` P `)`.

- $w, L_0 \models P_1$ `and` P_2 iff $w, L_0 \models P_1$ and $w, L_0 \models P_2$.

  If $w, L_0 \models P_1$ `and` P_2, then $w, L_0 \models P_1$ and $w, L_0 \models P_2$. By induction w', $L_0 \models P_1$ and w', $L_0 \models P_2$, and thus w, $L_0 \models P_1$ `and` P_2.

  If $w, L_0 \not\models P_1$ `and` P_2, then for some $i \in \{1, 2\}$, $w, L_0 \not\models P_i$. By induction w'', $L_0 \not\models P_i$, and thus w'', $L_0 \not\models P_1$ `and` P_2.

- $w, L_0 \models P_1$ `or` P_2 iff $w, L_0 \models P_1$ or $w, L_0 \models P_2$.

  If $w, L_0 \models P_1$ `or` P_2, then for some $i \in \{1, 2\}$, $w, L_0 \models P_i$. By induction w', $L_0 \models P_i$, and thus w', $L_0 \models P_1$ `or` P_2.

  If $w, L_0 \not\models P_1$ `or` P_2, then $w, L_0 \not\models P_1$ and $w, L_0 \not\models P_2$. By induction w'', $L_0 \not\models P_1$ and w'', $L_0 \not\models P_2$, and thus w'', $L_0 \not\models^{\leftarrow S} P_1$ `or` P_2.

Lemma 1.4 Let P be an SVA property without the `disable iff` operator, let L_0 be a local variable context, and let w be a word, then if there w', L_1 such that $ww', L_0, L_1 \not\models P$, then w, $L_0 \models^{\leftarrow S} P$, and if there w', L_1 such that $ww', L_0, L_1 \models P$, then w, $L_0 \models^{\leftarrow F} R$.

Proof: By Induction on P:

- Base:

  - $w, L_0 \models R$ iff there exist $0 \leq j < |ww'|$ and L_1 such that $w^{0,j}, L_0, L_1 \models R$.

    $ww', L_0 \not\models R$ implies that for every $xy = w$ we have that $x \not\models R$, thus, w, $L_0 \models^{\leftarrow S} R$.

$ww', L_0 \models R$ implies that for some $xy = ww'$ and some L_1, we have $x, L_0, L_1 | \equiv R$. We distinguish between two cases:

1. If x is a prefix of w then $w, L_0 \models R$ and thus w, $L_0 \models^{\leftarrow F} R$.
2. If w is a prefix of x, then Lemma 1.3 implies that w, $L_0 \mid\equiv^{\leftarrow} R$ and thus w, $L_0 \models^{\leftarrow F} R$.

- Induction:
  - $w, L_0 \models$ `not` P iff $\bar{w}, L_0 \not\models P$.
    If $ww', L_0 \not\models$ `not` P, then $w\bar{w}w', L_0 \models P$. By induction, $\bar{w}$, $L_0 \models^{\leftarrow F} P$ and thus w, $L_0 \models^{\leftarrow S}$ `not` P
    If $ww', L_0 \models$ `not` P, then $w\bar{w}w', L_0 \not\models P$. By induction, $\bar{w}$, $L_0 \models^{\leftarrow S} P$ and thus w, $L_0 \models^{\leftarrow F}$ `not` P.
  - $w, L_0 \models$ `(` R `|->` P `)` iff for every $0 \leq j < |w|$ and L_1 such that $\bar{w}^{0,j}, L_0, L_1 \mid\equiv R$, $w^{j\cdot\cdot}, L_1 \models P$.
    If $ww', L_0 \not\models$ `(` R `|->` P `)`, then there are $xyz = ww'$, $|y| = 1$ and L' such that $x\bar{y}, L_0, L' | \equiv R$ and $yz, L' \not\models P$. We distinguish between two cases:
    1. If w is a strict prefix of xy, then ww'^0, $L_0 \mid\equiv^{\leftarrow} R$. By Lemma A.1 $w\top$, $L_0 \mid\equiv^{\leftarrow} R$. and thus w, $L_0 \models^{\leftarrow S}$ `(` R `|->` P `)`.
    2. If xy is a prefix of w, then for u such that $xyu = w$, we have that $yuw', L' \not\models P$, and thus u, $L' \models^{\leftarrow S} P$. Thus, w, $L_0 \models^{\leftarrow S}$ `(` R `|->` P `)`.

    If $ww', L_0 \models$ `(` R `|->` P `)`, then for every $xyz = w$ and $|y| = 1$ and L_1 such that $x\bar{y}, L_0, L_1 \mid\equiv R$, we have that $yzw', L_1 \models P$. By induction $yz, L_1 \models^{\leftarrow F} P$, thus w, $L_0 \models^{\leftarrow F}$ `(` R `|->` P `)`.
  - $w, L_0 \models P_1$ `and` P_2 iff $w, L_0 \models P_1$ and $w, L_0 \models P_2$.
    If $ww', L_0 \not\models P_1$ `and` P_2, then for some $i \in \{1, 2\}$, $ww', L_0 \not\models P_i$. By induction w, $L_0 \models^{\leftarrow S} P_i$, and thus w, $L_0 \models^{\leftarrow S} P_1$ `and` P_2.
    If $ww', L_0 \models P_1$ `and` P_2, then $ww', L_0 \models P_1$ and $ww', L_0 \models P_1$. By induction w, $L_0 \models^{\leftarrow F} P_1$ and w, $L_0 \models^{\leftarrow F} P_1$, and thus w, $L_0 \models^{\leftarrow F} P_1$ `and` P_2.
  - $w, L_0 \models P_1$ `or` P_2 iff $w, L_0 \models P_1$ or $w, L_0 \models P_2$.
    If $ww', L_0 \not\models P_1$ `or` P_2, then $ww', L_0 \not\models P_1$ and $ww', L_0 \not\models P_1$. By induction w, $L_0 \models^{\leftarrow S} P_1$ and w, $L_0 \models^{\leftarrow S} P_1$, and thus w, $L_0 \models^{\leftarrow S} P_1$ `or` P_2.
    If $ww', L_0 \models P_1$ `or` P_2, then for some $i \in \{1, 2\}$, $ww', L_0 \models P_i$. By induction w, $L_0 \models^{\leftarrow F} P_i$, and thus w, $L_0 \models^{\leftarrow F} P_1$ `or` P_2.

To understand the problem with the `disable iff` operator, let us look at the property `disable iff` (b) 1 `intersect` (1 `##1` 1) and a word w such that $|w| = 1$ and $w^0 \models b$. Then, for every w' and L_0 in particular $w' = \epsilon$, we

have that $ww' \not\models$ `disable iff` (b) 1 `intersect` (1 `##1` 1). However, $w^0 \models b$ and $w, L_0 \models^{\leftarrow F}$ 1 `intersect` (1 `##1` 1). Thus, $w, L_0 \not\models^{\leftarrow S}$ `disable iff` (b) 1 `intersect` (1 `##1` 1).

The source of the problem is that in the `disable iff` operator, the notion of "P has not failed yet on w" is captured by $w\top^\omega \models P$. This definition fails the property 1 `intersect` (1 `##1` 1) on the first letter of w although non of the sub sequences fails there. We see this as unintuitive, and that why according to the definition of $\models^{\leftarrow F}$ the sequence 1 `intersect` (1 `##1` 1) fails only in the second letter.

Proposition A.3. *For every sequence R and local variable context L_0, we have that ϵ, $L_0 \models^{\leftarrow} R$.*

The proposition can be easily proved with induction on R.

Lemma 1.5 Let R be an SVA sequence without the `intersect` and `##0` operators, let L_0 be a local variable context, and let w be a word. Then w, $L_0 \models^{\leftarrow} R$ implies that there are $k \geq 0$, L_1 such that $w\top^k, L_0, L_1 \models R$.

Proof: By induction on R:

- Base:
  - w, $L_0 \models^{\leftarrow}$ `(` 1, v `=` e `)` implies that either $w = \epsilon$ in which case it can be extended with $\top^1$ or $w, L_0, L_1 \models R$ where $L_1 = \{(v, e[L_0, w^0])\} \cup L_0|_D$. In this case it can be extended with $\top^0$. In both cases $w\top^k, L_0, L_1 \models$ `(` 1, v `=` e `)`.
  - w, $L_0 \models^{\leftarrow} b$ implies that either $w = \epsilon$ in which case it can be extended with $\top^1$ or $w, L_0, L_0 \models b$ in which case it can be extended with $\top^0$. In both cases $w\top^k, L_0, L_0 \models b$.
- Induction:
  - w, $L_0 \models^{\leftarrow}$ `(`R`)`implies w, $L_0 \models^{\leftarrow} R$. By induction there exists k, L_1 such that $w\top^k, L_0, L_1 \models R$ thus, $w\top^k, L_0, L_1 \models$ `(` R `)`.
  - w, $L_0 \models^{\leftarrow}$ `(` R_1 `##1` R_2 `)` iff w, $L_0 \models^{\leftarrow} R_1$ or for some $xy = w$ and local variable context L' we have, $x, L_0, L' \models R_1$ and y, $L' \models^{\leftarrow} R_2$.
    If for some $xy = w$ and local variable context L' we have, $x, L_0, L' \models R_1$ and y, $L' \models^{\leftarrow} R_2$ then by induction there are k, L_1 such that $y\top^k, L', L1 \models R_2$. Thus, $w\top^k, L_0, L_1 \models$ `(` R_1 `##1` R_2 `)`.
    If w, $L_0 \models^{\leftarrow} R_1$, then by induction there are k and L' such that $w\top^k, L_0, L' \models R_1$. By Proposition A.3, $\epsilon, L' \models^{\leftarrow} R_2$, and by induction there are k' and L_1 such that $\epsilon\top^{k'}, L', L_1 \models R_2$. Thus, $w\top^k\top^{k'}, L_0, L_1 \models$ `(` R_1 `##1` R_2 `)`.
  - w, $L_0 \models^{\leftarrow}$ `(`R_1 `or` R_2`)` implies w, $L_0 \models^{\leftarrow} R_1$ or w, $L_0 \models^{\leftarrow} R_2$. By induction there are k and L_1 such that either $w\top^k, L_0, L' \models R_1$ or $w\top^k, L_0, L' \models R_2$. Thus, $.w\top^k, L_0, L_1 \models$ `(` R_1 `or` R_2 `)`.

- w, $L_0 \models^{\leftarrow}$ `first_match ( R )` iff w, $L_0 \models^{\leftarrow} R$ and for every $xy = w$ and L_1, such that $|y| > 0$, we have $x, L_0, L_1 \not\models R$

  By induction there are k L_1 such that $w\top^k, L_0, L_1 \models R$. Let k be the minimal number such that there exists L_1 such that $w\top^k, L_0, L_1 \models R$, then $w\top^k, L_0, L_1 \models$ `first_match ( R )`.

- w, $L_0 \models^{\leftarrow} R$ `[*0]` iff $w = \epsilon$ iff $w\top^0, L_0, L_0 \models R$ `[*0]`.

- w, $L_0 \models^{\leftarrow} R$ `[*1:$]` iff either w, $L_0 \models^{\leftarrow} R$, or for some $xy = w$ and a local variable context L' we have $x, L_0, L' \models R$ `[*1:$]` and $y, L' \models^{\leftarrow} R$.

  If for some $xy = w$ and L' we have $x, L_0, L' \models R$ `[*1:$]` and $y, L' \models^{\leftarrow} R$, then by induction there are k and L_1 such that $y\top^k, L', L_1 \models R$. Thus $w\top^k, L_0, L_1 \models R$ `[*1:$]`.

  If w, $L_0 \models^{\leftarrow} R$, then by induction there are k and L_1 such that $w\top^k, L_0, L_1 \models R$. Thus $w\top^k, L_0, L_1 \models R$ `[*1:$]`.

The following Lemma is proved in [] (**Lemma 5.1**)

Lemma A.4. *Let w be a word, L_0 and L_1 be local variable contexts, and R be an* SVA *sequence. If $w, L_0, L_1 \models R$, then w does not contain the letter $\perp$.*

Lemma 1.6 Let P be a non degenerate SVA property without the `intersect` and `##0` operators, let L_0 be a local variable context, and let w be a word. If w, $L_0 \models^{\leftarrow S} P$, then $w\perp^\omega, L_0, \models P$. If w, $L_0 \models^{\leftarrow F} R$ then $w\top^\omega, L_0, \models P$.

Proof: By Induction on P:

- Base:

  - w, $L_0 \models^{\leftarrow S} R$ implies that for every $xy = w$ and local variable context L_1, we have $x, L_0, L_1 \not\models R$. Thus we have that $w, L_0 \not\models R$. By Lemma A.4, for every $k > 0$ and L_1 we have $w\perp^k, L_0, L_1 \not\models R$, thus $w\perp^\omega L_0 \not\models R$.

    $w, L_0 \models^{\leftarrow F} R$ implies either $w, L_0 \models R$ in which case, for every extension w' and in particular $w' = \top^\omega$, we have $ww', L_0 \models R$. Or $w, L_0 \models^{\leftarrow} R$, in which case Lemma 1.5, implies that there are $\top^k$ and L_1 such that $w\top^k, L_0, L_1 \models R$, and thus $w\top^\omega, L_0 \models R$,

- Induction:

  - w, $L_0 \models^{\leftarrow F} R$ `|->` P iff for every $xyz = w$, $|y| = 1$, and L', if $x\bar{y}, L_0, L' \models R$ then yz, $L' \models^{\leftarrow F} P$.

    Lemma A.4 implies that every $k > 0$ and L_1 we have $w\bar{\top}^k, L_0, L_1 \not\models R$. Let $xyz = w$ be such that $|y| = 1$ and xy is a prefix of w. Then, for every L_1, if $x\bar{y}, L_0, L_1 \models R$, then yz, $L_1 \models^{\leftarrow F} P$. By induction $yz\top^\omega \models P$. Thus $w\top^\omega, L_0 \models R$`|->`$P$

$w, L_0 \models^{\leftarrow S} R$ `|->` P iff $w\top, L_0 \models^{\leftarrow} R$ or for some $xyz = w$, $|y| = 1$ and L', we have that $x\bar{y}, L_0, L'| \equiv R$, and $yz, L_0 \models^{\leftarrow S} P$

If $w\top, L_0 \models^{\leftarrow} R$, then Lemma 1.5 implies that for some $k > 0$ and L_1 we have $w\top^k, L_0, L_1| \equiv R$, thus $w\bot^k, L_0, L_1| \equiv R$. Since P is non degenerate, $\bot^\omega, L_1 \not\models P$, thus $w\bot^\omega \not\models R$ `|->` P.

If for some $xyz = w$, $|y| = 1$ and L', we have that $x\bar{y}, L_0, L'| \equiv R$, and $yz, L_0 \models^{\leftarrow S} P$, then by induction $yz\bot^\omega, L' \not\models P$ and thus, $w\bot^\omega \not\models R$ `|->` P.

- $w, L_0 \models^{\leftarrow S} P_1$ `and` P_2 iff $w, L_0 \models^{\leftarrow S} P_1$ or $w, L_0 \models^{\leftarrow S} P_2$. By Induction $w\bot^\omega, L_0 \not\models P_1$ or $w\bot^\omega, L_0 \not\models P_2$. Thus $w\bot^\omega, L_0 \not\models P_1$ `and` P_2.

  $w, L_0 \models^{\leftarrow F} P_1$ `and` P_2 iff $w, L_0 \models^{\leftarrow F} P_1$ and $w, L_0 \models^{\leftarrow F} P_2$. By Induction $w\top^\omega, L_0 \models P_1$ and $w\top^\omega, L_0 \models P_2$. Thus $w\top^\omega, L_0 \models P_1$ `and` P_2.
- $w, L_0 \models^{\leftarrow S} P_1$ `or` P_2 iff $w, L_0 \models^{\leftarrow S} P_1$ and $w, L_0 \models^{\leftarrow S} P_2$. By Induction $w\bot^\omega, L_0 \not\models P_1$ and $w\bot^\omega, L_0 \not\models P_2$. Thus $w\bot^\omega, L_0 \not\models P_1$ `or` P_2.

  $w, L_0 \models^{\leftarrow F} P_1$ `or` P_2 iff $w, L_0 \models^{\leftarrow F} P_1$ or $w, L_0 \models^{\leftarrow F} P_2$. By Induction $w\top^\omega, L_0 \models P_1$ or $w\top^\omega, L_0 \models P_2$. Thus $w\top^\omega, L_0 \models P_1$ `or` P_2.
- $w, L_0 \models^{\leftarrow S}$ `not` P iff $\bar{w}, L_0 \models^{\leftarrow F} P$. By induction $\bar{w}\top^\omega \models P$. Thus $w\bot^\omega \not\models$ `not`P

  $w, L_0 \models^{\leftarrow F}$ `not` P iff $\bar{w}, L_0 \models^{\leftarrow S} P$. By induction $\bar{w}\bot^\omega \not\models P$. Thus $w\top^\omega \models$ `not`P
- $w, L_0 \models^{\leftarrow F}$ `disable iff (`b`)` P iff either $w, L_0 \models^{\leftarrow F} P$ or there exists $0 \leq k \leq |w|$ such that $w^{0..k}, L_0 \models^{\leftarrow F} P$ and $w^k \models b$.

  If $w, L_0 \models^{\leftarrow F} P$, then by induction $w\top^\omega, L_0 \models P$ and thus $w\top^\omega, L_0 \models$ `disable iff (`b`)` P.

  If there exists $0 \leq k \leq |w|$ such that $w^{0..k}, L_0 \models^{\leftarrow F} P$ and $w^k \models b$. Then, by induction $w^{0..k}\top^\omega, L_0 \models P$. Thus, $w\top^\omega, L_0 \models$ `disable iff (`b`)` P.

  $w, L_0 \models^{\leftarrow S}$ `disable iff (`b`)` P iff $w, L_0 \models^{\leftarrow S} P$ and for all $0 \leq k \leq |w|$ s.t. $w^k \models b$ we have that $w^{0..k}, L_0 \not\models^{\leftarrow F} P$.

  By Induction $w\bot^\omega, L_0 \not\models P$. For every $0 \leq k < |w|$ we have that if $w^k \models b$, then $w^{0..k}, L_0 \not\models^{\leftarrow F} P$. By Lemma 1.4 $w^{0..k}\top^\omega, L_0 \not\models P$. For every $k \geq |w|$ we have that $(w\bot^\omega)^k, L_0 \not\models b$, Thus,$w\bot^\omega, L_0 \not\models$ `disable iff (`b`)` P.