



FP6-IST-507219

PROSYD:

Property-Based System Design

Instrument: Specific Targeted Research Project

Thematic Priority: Information Society Technologies

Annual Language Study and Recommendation to Accellera (Deliverable 4.3/1c)

Due date of deliverable: December 31, 2006

Actual submission date: December 31, 2006

Start date of project: January 1, 2004

Duration: Three years

Organisation name of lead contractor for this deliverable: IBM

Revision 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	<input checked="" type="checkbox"/>
PP	Restricted to other programme participants (including the Commission Services)	<input type="checkbox"/>
RE	Restricted to a group specified by the consortium (including the Commission Services)	<input type="checkbox"/>
CO	Confidential, only for members of the consortium (including the Commission Services)	<input type="checkbox"/>

Notices

For information, contact eisner@il.ibm.com.

This document is intended to fulfil the contractual obligations of the PROSYD project concerning deliverable 4.3/1c described in contract number 507219.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

© Copyright PROSYD 2006. All rights reserved.

Table of Revisions

Version	Date	Description and Reason	By	Affected Sections
0.1	15.11.06	First draft by authors	Cindy Eisner	All
0.2	26.11.06	Update with comments from partners	Cindy Eisner	2
1.0	25.12.06	Incorporate final set of comments from partners, and minor formatting.	Cindy Eisner	2

Authors

Lyes Benalycherif

Cindy Eisner

Andrea Fedeli

Dana Fisman

Adriana Maggiore

Anthony McIsaac

Avigail Orni

Dmitry Pidan

Marco Roveri

Sitvanit Ruah

Simone Semprini

Klaus Winkelmann

Executive Summary

This document contains the 2004-2006 combined PSL language recommendations of all partners of the PROSYD consortium to Accellera. We record various proposals we have made to Accellera as a result of our work on PROSYD and make a number of new recommendations.

Purpose

The purpose of this document is to distil our experience of the past three years with PSL into concrete recommendations as to language extensions. A second purpose is to document the influence of the PROSYD project on the evolution of the standard. For this purpose, information from the previous versions of this deliverable (D4.3/1a, D4.3/1b) is also included.

Intended Audience

This report is intended for Mr. Harry Foster (a consortium member), in his role as the Chairman of the Accellera Formal Verification Technical Committee, and in

his follow-on role as Chairman of IEEE P1850, which is following Accellera's recommendation to further standardize Accellera PSL as IEEE-1850, for the members of the IEEE P1850 working group, and for the Project Officer and Reviewers of the PROSYD project.

Background

The consortium delivered its first set of language recommendations in deliverable 4.3/1(a), delivered March 29, 2005, and its second set in deliverable 4.3/1(b) delivered December 31, 2005. During the course of the past year, we have gained additional experience with the language. In this document, we summarize our recommendations to Accellera based on that experience, as well as document our past recommendations and their influence on the standard.

Contents

Table of Revisions	iii
Authors.....	iii
Executive Summary	iii
Purpose.....	iii
Intended Audience	iii
Background	iv
Glossary	vi
1 Introduction.....	1
2 Recommendations.....	3
Previously submitted recommendations (resolved)	3
Previously submitted recommendations (unresolved)	5
New recommendations.....	9
3 References.....	13

Glossary

Flavour

PSL comes in various flavours, corresponding to the main hardware description languages. Two of the flavours are Verilog and VHDL.

HDL

Hardware description language.

IEEE

Institute of Electrical and Electronics Engineers. A non-profit, technical professional association of more than 360,000 individual members in approximately 175 countries.

PSL

Property Specification Language, the language for specification of designs upon which PROSYD is based.

SERE

Sequential Extended Regular Expression, a basic type of temporal expression in PSL.

Sequence

A specific type of SERE.

Structural contradiction

A contradiction which happens because of incompatible temporal requirements (e.g. that a SERE last for N cycles and also for N+1 cycles), as opposed to a logical contradiction that happens because of incompatible logical requirements (e.g. that both "b" and "not b" hold simultaneously).

Suffix implication

A type of temporal property in PSL.

Verilog

One of two standardized HDLs used to specify the structure and behaviour of electronic systems in textual format. Developed in the mid-1980s as a proprietary language and acquired by Cadence Design Systems, it became a de facto industry standard. In the mid-90s Cadence placed it into the public domain and it became a de jure standard promulgated by the IEEE. Verilog is also the name of a legacy simulation tool offered by Cadence.

VHDL

The second of two standardized HDLs used to specify the structure and behaviour of electronic systems in textual format. VHDL stands for VHSIC Hardware Description Language. It was developed starting in 1983 with support from the U.S. government, and in 1987 was adopted by the IEEE as a standard (IEEE 1076).

1 Introduction

There are three relevant versions of the standard:

- Accellera PSL Version 1.01 was the official version as of the start of this project. It was released on April 25, 2003 and approved by Accellera on May 29, 2003.
- Accellera PSL Version 1.1 was released on June 9, 2004 and was approved by Accellera on June 9, 2004. Following this release, the Accellera board approved the transfer of the standard to the IEEE (IEEE standardisation is the ultimate goal of most Accellera work).
- IEEE P1850-2005 (PSL) was released on October 17, 2005.

The following PROSYD personnel serve or served on the Accellera Formal Verification Technical Committee (FVTC) that released the PSL standard, and/or on the IEEE P1850 working group (the follow-on to the Accellera FVTC) and its various sub-committees:

Lyes Benalycherif (ST France)
Cindy Eisner (IBM)
Andrea Fedeli (ST Italy)
Dana Fisman (Weizmann/IBM)
Harry Foster (Accellera)
Danny Geist (IBM)
Adriana Maggiore (OneSpin)
Anthony McIsaac (ST UK)
Avigail Orni (IBM)
Dmitry Pidan (IBM)
Sitvanit Ruah (IBM)
Klaus Winkelmann (OneSpin)
Yaron Wolfsthal (IBM)

PROSYD personnel have actively participated in the Accellera FVTC, as documented in the mail archives of the committee (see <http://www.eda.org/vfv/>), in which 424/1334 (over 30%) of the messages originate from PROSYD members.

With the transfer of the work to the IEEE, PROSYD personnel have continued their active participation in the standardisation process, as documented in the mail archives of the IEEE P1850 working group and its three sub-committees, in which 283/752 (more than 35%) of the messages originate from PROSYD members, as follows:

Working group: 76/218 messages, see

<http://www.eda.org/ieee-1850/hm/author.html>

Extensions sub-committee: 55/173 messages, see

<http://www.eda.org/ieee-1850/ieee-1850-extensions/hm/author.html>

Issues sub-committee: 122/287 messages, see

<http://www.eda.org/ieee-1850/ieee-1850-issues/hm/author.html>

LRM sub-committee: 30/74 messages, see
<http://www.eda.org/ieee-1850/ieee-1850-lrm/hm/author.html>

In addition, we note that Sitvanit Ruah of IBM has recently taken over as chair of the IEEE 1850 issues subcommittee.

Due to the intense nature of the PROSYD involvement in the day-to-day standardisation work, the contents of this document will not come as a surprise to the IEEE P1850 committee. Instead, the intention is to gather together in one place all recommendations that originated in the PROSYD consortium, as documentation on how PROSYD influenced the evolution of the standard.

First, we comment that the first IEEE version of PSL, known as IEEE Std 1850-2005, was released on October 17, 2005. All of our new recommendations, detailed below, relate to this most updated version of the standard. We include in this document recommendations that were previously made but not yet resolved one way or another in the standard. For purposes of documentation, we also include recommendations made in the first and second versions of this document (4.3/1(a), 4.3/1(b)) that have already been discussed and resolved (either accepted or rejected) by the standards committee, as well as their disposition. Finally, it is important to note that some of the issues, discussion is still ongoing in the consortium (and may continue into the IEEE committee).

2 Recommendations

Previously submitted recommendations (resolved)

For the record, we summarize here issues that were submitted in the first version of this document (4.3/1(a)) and already resolved by the standards committee.

1. **HDL Embedding:** The syntax {a} is ambiguous in the Verilog flavour. It can be either a SERE consisting of the single Verilog Boolean expression a, or the Verilog Boolean expression {a} (a trivial concatenation). Several suggestions for syntactic changes were made.

Resolution (for version 1.1): Make no syntactic changes. Rather, define that {a} will always be parsed as a SERE.

2. **Tokenisation:** Accellera Version 1.01 does not clearly enumerate the tokens of the language.

Resolution: Appendix A.2 of Accellera Version 1.1 includes a list of tokens.

3. **Case-sensitivity:** Accellera Version 1.01 states that keywords are case-sensitive in all flavours, but this is inconsistent with the spirit of VHDL for the VHDL flavour.

Resolution: Accellera Version 1.1 states that keywords are reserved identifiers. Therefore, the case-sensitivity rules of the underlying flavour apply.

4. **Operator precedence:** The operator precedence table of Accellera Version 1.01 is missing some operators.

Resolution: Missing operators were added to Accellera Version 1.1.

5. **Sequence:** The syntax definition of sequences in a draft version of LRM 1.1 was inconsistent (different definition in Section 6 and Appendix A).

Resolution: fixed in Accellera Version 1.1.

6. **Sequence operators (&, &&, |):** The sequence operators (&, &&, |) have different precedence in Verilog and in PSL.

Resolution: fixed in Accellera Version 1.1.

7. **Suffix implication:** There are two forms of suffix implication (the old form r(f) and the new form r |-> f) but Section 6 describes their semantics using different wording. This is confusing.

Resolution: Accellera Version 1.1 clarifies that the semantics are exactly the same.

8. **SERE concatenation:** Section 6 of a draft of Accellera Version 1.1 uses "holds" rather than "holds tightly" to describe the semantics.

Resolution: Fixed in Accellera Version 1.1.

9. **Inheritance:** Section 5 of a draft of Accellera Version 1.1 is unclear regarding whether or not it is legal to have multiple definitions of the same signal in the transitive closure of all inherited verification units.

Resolution: Clarified in Accellera Version 1.1.

10. **Replicated properties:** In a draft of Accellera Version 1.1, a repetition count is required to be statically computable, but on the other hand, it is suggested that a forall Name can be used as a repetition count.

Resolution: Clarified in Accellera Version 1.1 that the parameter defined by a replicator is considered to be static.

11. **Verification directives:** In a draft of Accellera Version 1.1, it was not clear what the scope of a label is.

Resolution: Clarified in Accellera Version 1.1.

12. **Core language:** It was suggested that perhaps the language is too complex, and that standardization should be split into at least two parts – a small subset to be standardized now, and the rest of the standardization process to be postponed to a later date. Note: this opinion was not held by all members of the PROSYD consortium, but was nevertheless raised for discussion in Accellera.

Resolution: This proposal was rejected, on the basis that PSL was already a standard at the time it was made, and therefore the proposal was irrelevant.

13. **Typed forall:** the forall construct introduces a variable whose type is not specified. Consider requiring it to be typed.

Resolution: This proposal was considered as part of the Group I proposals in the IEEE extensions committee. Ultimately, it was resolved by requiring that each value in the value range be statically computable, and that all be of the same type.

14. **Non-determinism:** PSL needs better support for non-determinism, especially in the Verilog and VHDL flavours.

Resolution: This was examined by the IEEE committee as part of Group J, and a proposal for extended non-determinism support was adopted. The adopted proposal aligns the syntax and semantics of the new built-in functions nondet() and nondet_vector() with those of forall.

15. The paper “Combining System Level Modeling with Assertion Based Verification”, by A. Dahan, D. Geist, L. Gluhowski, D. Pidan, G. Shapir, Y. Wolfsthal, M.L. Benalycherif, R. Kamdem, Y. Lahbib, 6th International Symposium on Quality Electronic Design, IEEE Computer Society ACM 2005 describes a method of using PSL in conjunction with SystemC. We

recommend that a **SystemC flavour** of PSL be considered.

Resolution: A SystemC flavour is included in the IEEE P1850 standard.

16. In Accellera Version 1.1, the only supported types for formal parameters of named properties are Boolean, constant, sequence and property. We recommend that the **supported types be extended** to include Bit, BitVector, Numeric and String.

Resolution: Additional types are supported in the IEEE P1850 standard.

Previously submitted recommendations (unresolved)

For the record, we summarize here issues that were submitted in the first and second versions of this document (4.3/1(a), 4.3/1(b)) but not yet resolved by the standards committee.

1. **Length matching:** the logical contradiction false is treated differently than the structural contradiction. It has been known since Accellera Version 1.01 (where it is mentioned in Appendix B) that there is an inconsistency in the way that structural contradictions are treated vs. the way that logical contradictions are treated. During the past year some ideas for the direction of a solution have been explored, and are described in [EFH05]. We recommend that a future version of the formal semantics base a solution to this problem on [EFH05].

Benefit: Language consistency.

Status: Recorded as issue 131 in the database.

2. **The semantics of $r[*0]$** are different depending on whether or not $r[*0]$ is standalone or is concatenated with another SERE. There is no good reason for this. We recommend that it be changed, and note that the formal semantics of [EFH05] fix this problem as well.

Benefit: Language consistency.

Status: Recorded as issue 132 in the database.

3. **Sampling semantics¹:** Build an HDL-independent representation of the DUV, in PSL terms. This is currently delegated to the verification tool, but should instead be coded by the LRM, so that any possible added flavour has a clear representation to match with, and any possible implementation of a PSL translator has a flavour independent reference for the DUV. One of the first steps along the path to this result is the definition of a sampling semantics. For synthesizable designs, this could be derived from a standardized transition system associated with the synthesized model. Note: some members of the PROSYD consortium object to this recommendation.

¹ This recommendation has been reworded since the previous version of this deliverable, at the request of Anthony McIsaac and Andrea Fedeli.

Benefit: Flavour independence, portability.

Status: Recorded as issue 115 in the issues database.

4. Suppose we would like to say that signal "a" is independent of signal "b". This could be checked by a cone-of-influence reduction. Currently, there is no way to say such **meta-properties in PSL**. We recommend that inclusion of such language constructs be considered.

Benefit: Add expressive power.

Status: Recorded as issue 133 in the database.

5. We recommend that a new directive called "**witness**" be added. This will instruct a verification tool to produce a trace on which the property holds (as opposed to "assert", which instructs the tool to verify that it holds on all traces).

Benefit: Prevent the coding of ostensible "assertions" which are actually intended to fail.

Status: Recorded as issue 134 in the database.

6. **PSL_Identifier:** The BNF is missing the definition of PSL_Identifier. Note: what is actually missing is not the definition, but the italics (*PSL_Identifier*) that would indicate that *PSL_Identifier* is equivalent to Identifier.

Benefit: Clarification.

Status: Recorded as issue 135 in the database.

7. The PSL language only supports temporal operators allowing to speak about the future. We would like to suggest extending PSL to deal also with **past temporal operators**. Here follows some phrases from [CRS'04] that justify the proposal.

Temporal logics with past operators are being devoted increasing interest in a number of application areas (e.g. formal verification [GLL99, CKM01, FLP+03], requirement engineering [KPV01, vL01], and automated task planning [BK95]).

In the widely-used setting of Linear Temporal Logics (LTL), past operators do not add expressive power with respect to pure-future: any LTL formula with past operators can be rewritten by only using future-time operators [Gab89]. On the other hand, past operators are very useful in practice, since they help to keep specifications compact, simple, and easier to understand. This practical consideration has a formal counterpart in the fact that the elimination of past operators for pure-future operators may come at the cost of a non-elementary blow-up [LMS02].

The Y (for "Yesterday ") operator is the temporal dual of X and refers to the previous time instant. At any non-initial time, Y a is true if and only if a holds at the previous time instant. The Z operator is similar to the Y operator, and it only differs in the way the initial time instant is dealt with: at time zero, Y a is false, while Z a is true.

The O (for "Once ") operator is the temporal dual of F (sometimes in the future), so $O a$ is true iff a is true at some past time instant (including the present time). Likewise, H (for "Historically ") is the past-time version of G (always in the future), so that $H a$ is true iff a is always true in the past. The S (for "Since ") operator is the temporal dual of U (until), so that $S a b$ is true iff b holds somewhere in the past and a is true from then up to now. Finally, we have a T $b = \neg(\neg a S \neg b)$ (T is called the "Trigger " operator), exactly as in the future case we have $R b = \neg(\neg a U \neg b)$.

On an informal (but very important) level, past operators allows us to formalize properties more naturally. For instance, if a problem is diagnosed, then a failure must have previously occurred, can be represented in PLTL as:

$G(\text{problem} \rightarrow O \text{ failure})$

that is more natural than its pure-future counterpart $\neg(\neg \text{failure} U \text{ problem})$.

Similarly, the property grants are issued only upon requests can be easily specified as

$G(\text{grant} \rightarrow Y(\neg \text{grant} S \text{ request}))$

compared to the corresponding pure-future translation

$(\text{request} R \neg \text{grant}) \ \&\& \ G(\text{grant} \rightarrow (\text{request} \parallel (X(\text{request} R \neg \text{grant}))))$.

Benefit: as described above.

Status: Recorded as issue 136 in the database.

8. The PSL LRM lacks a formal description of verification directives, and consequently ambiguities may arise on their actual meaning. In particular this applies for the **assume directive**, that defines the assumptions under which the properties specified as assertions are requested to hold. The way the assumptions are applied is not precisely stated, and this gives place to the following possible interpretations:
 - a. Assumptions are used as premises of logical implications that have assertions as consequences: in this case, assertions are verified only on those paths that satisfy the assumptions (being the implication trivially true in the other paths); if we consider the following couple

assume always p ;

assert always ϕ ;

we would have to verify that $(\text{always } p) \rightarrow (\text{always } \phi)$ holds.
 - b. Assumptions are used to define the paths or portions of paths on which the verification of assertions must be applied; if we consider the following couple

assume always p;

assert always phi;

we would have to verify that (always p -> phi) holds.

Assumptions a la (a) may be used to tackle scenarios like the following: suppose we have specified the monitoring system of a plant and that we want to check that the monitoring system will never raise a warning if no failure conditions are detected, we would write something like

assume never failure;

assert never warning;

In this case the assertion would be checked only on whole paths that satisfy the assumption. Suppose now we want to check that no warning is issued as long as the value of the variable pressure_level is lower than 5, we would write

assume never pressure_level > 5;

assert never warning;

this time interpreting the assumptions ala (b), and the assertion would be checked only on those paths or portions of path that satisfy the assumption.

Since both interpretations are interesting and are potentially useful, a suggestion can be made to extend the set of verification directives to include them both. A proper attention, anyway, must be paid to define the kind of assertion described in 2, since it is not clear how to handle in this way assumptions that shape differently than always p.

Benefit: as described above.

Status: Recorded as issue 126 in the issues database.

9. **Group C:** vunits - binding, scoping, inheritance, parameterized. In addition to the existing issues in this group, we would like to address the following issues:

- a. In section 5.2.1, the rules for determining the meaning of a name can be interpreted as allowing only 4 levels of inheritance. This should be clarified, to unambiguously describe a model with multiple levels of inheritance.

Benefit: Added flexibility.

Status: Recorded as issue 92 in the database.

- b. Since the modeling layer may override the behavior of a signal, there is a need for the built-in function "original()": original(S) gives the original behavior of signal S.

Benefit: Access to original value, for instance, in order to allow comparison with abstract value.

Status: Recorded as issue 93 in the database.

10. **Group K:** cover. This includes the "covergroup" issue raised by Johan M. In the issues list, group K is listed under "LRM Changes Adopted in IEEE 1850-2005 PSL", but we believe it belongs in "issues to be addressed".

Benefit: Clarification.

Status: Recorded as issue 89 in the database.

11. **Formal and informal semantics:**

- c. need to define formal semantics for directives

Benefit: Unambiguous formal definition to replace English language description.

Status: Recorded as issue 94 in the database.

- d. need to informally clarify **the 4 levels of satisfiability**

Benefit: Clarify in a reader-friendly manner the formal semantics given in Appendix B.

Status: Recorded as issue 95 in the database.

12. Improving the **clock context definition** for built-in functions. Consider the following property:

```
assert always ( a -> rose(b, rose(c)));
```

According to the current definition, the inner rose is the explicit clock context for the outer rose. But the inner rose has no explicit clock, so by the LRM definition, its clock context is defined by the enclosing built-in function -- which is the outer rose. That means that the inner rose is the clock context of itself.

Benefit: Clarification.

Status: Recorded as issue 96 in the database.

New recommendations

In this section we detail new issues that we recommend Accellera and IEEE consider for the next version of the standard. They are:

1. **Passing a module as parameter:** Phrasing of the point-to-point property for the Wishbone case study was done using as parameters the relevant signals of one master and one slave. Each such property usually involved 5-8 parameters. These parameters were later instantiated in the top-level properties. It would have been much less cumbersome if it was possible to pass as a parameter to the property a slave and a master, rather than the relevant signals of each. This is not possible in PSL, since the allowed parameters are one of the following: Boolean, bit, bitvector, numeric,

string, sequence, property. An entire module may not be passed as a parameter. We suggest enhancing the set of allowed parameters to include a predefined module.

Benefit: Ease of use.

2. **Parameterized vprops, vmodes and vunits:** PSL mainly has very few specific features to handle parameterized code, named properties and sequences. These features are clearly insufficient for the bus protocol context (WP1 case study) which needs parameterized vprops, vmodes and vunits. We recommend this language extension.

Benefit: Ease of use.

3. **Analog extensions:** PROSYD deliverable D1.3/2 describes an extension of PSL to allow it to be used in real-time and analog properties. We recommend that the committee consider this extension.

Benefit: Applicability to analog designs.

4. **Counting untils.** Extend PSL to allow statements such as “p holds until the 3rd occurrence of q.” This property occurs for instance in buffer protocols in which the bus must be granted to a given client until the 3rd transfer has completed. This property is hard to express in PSL. In LTL style, we can write

$$(p \text{ until } (p * q * \text{ next}(p \text{ until } (p * q * \text{ next}(p \text{ U } q)))))) \text{ or}$$
$$(p \text{ until } q) \ \& \ \text{next_event_a}[1..2](q)(\text{next}(p \text{ until } q)),$$

neither of which is particularly intuitive. In SERE style, one can write

$$\{a[*]\} \ \&\& \ \{b[->3]\},$$

which is short but not particularly readable as it depends critically on the length-matching semantics of &&. It should also be noted that the LTL and SERE styles can be seen as complementary and it would be beneficial if important properties can readily be expressed in both.

It should be noted that the required functionality is not provided by clocked PSL, as clocked formulas do not make any requirements on the signals between the clock ticks.

Thus, we see room for an additional LTL-style operator. We would propose the notation `until[n]`. This notation would be generalized to the entire until family: `until_[n]`, `until![n]`, and `until_![n]`. The construct can likewise be extended to the before family. Note that the `next_event` operator already has a corresponding way to count occurrences. The interaction with the clocking constructs should be considered in detail but appears straightforward at first glance.

Benefit: Ease of use.

5. **Qualified Inheritance:** Currently only single properties can be assumed or asserted; if inheritance were combined with `assert` or `assume` qualifiers, it would be possible to assume all properties in the inherited vunit as either assertions or assumption, respectively. Example:

```
vunit myprops(<fparams>) {
```

```

property p1(<aparms1>) ...
property p2(<aparms2>) ...
...
}

vunit vu1(<fparms>) {
...
assert myprops(<aparms>);
...
}

vunit vu2(<fparms>) {
...
assume myprops(<aparms>);
...
}

```

in which `aparms` are either made of (subsets of) `fparms` and local vars/defines.

Benefit: Improve reuse.

Prerequisite: availability of parametrized vunits.

6. **Remove Implicit Binding and Override:** IEEE PSL states that vunit variables are bound to design entities (variables/register signals) by name, implicitly (clause 7.2.3). Although handy for RHS references, this is dangerous (can introduce both false positives and false negatives) for the implicit override when a design entity name is used as LHS of an assignment, and furthermore totally breaking reuse. Implicit override should be replaced by explicit override. As params usage is a binding, also RHS implicit binding should be removed, because relying on name correspondence is too restrictive from a reuse point of view. Reckoning that it can be handy in cases where name equivalence is in facts, a new keyword could be devoted to allow automatic binding by name.

Benefit: Remove a possible source of false results. Improve reuse.

Prerequisite: parametrized vunits (not strictly necessary, but almost useless without).

7. **Local Binding and scope:** variables should be visible only in the current vunit and in those inheriting from the current one; transitive closure of inheritance should require to always specify the referenced vunit in case of variable name conflict. Example:

```

vunit a {
var v:Boolean;
assign v:=1;
...
}
vunit b {
inherit a;
var v:Boolean;
assign v:=2; -- no clashing: by default v is local
...
}

```

```
}  
vunit c {  
  inherit b;  
  var v,w,y,z: Boolean;  
  assign v=a.v; -- NB: va visibility is inherited from b.  
  assign w=b.v;  
  assign y=v+a.v+b.v;  
  ...  
}
```

inherited variables could be used only as RHS in assignments.

Benefit: improve reuse.

3 References

- [ABF+06] G. Auerbach, L. Benalycherif, A. Fedeli, D. Fisman, A. McIsaac, K. Winkelmann, "Case Studies in Property-Based Requirements Specification", PROSYD Deliverable 1.4/1.
- [BK95] F. Bacchus and F. Kabanza. Control strategies in planning. In Proceedings of the AAAI Spring Symposium Series on Extending Theories of Action: Formal Theory and Practical Applications, pages 5-10, Stanford University, CA, USA, March 1995.
- [CKM01] J. Castro, M. Kolp, and J. Mylopoulos. A requirements-driven development methodology. In Proceedings of the 13th International Conference on Advanced Information Systems Engineering, 2001.
- [CRS'04] A. Cimatti, M. Roveri and D. Sheridan "Bounded Verification of Past LTL" Accepted for publication in FMCAD-2004, the 5th International Conference in Formal Methods in Computer-Aided Design. November 14-17, 2004 Austin, Texas, USA.
- [EFH05] C. Eisner, D. Fisman and J. Havlicek. A topological characterization of weakness. Proc. 24th Annual ACM Symposium on Principles of Distributed Computing (PODC), July 2005, pp. 1-8.
- [FLP+ 03] A. Fuxman, L. Liu, M. Pistore, M. Roveri, and J. Mylopoulos. Specifying and analyzing early requirements: Some experimental results. In IEEE Int. Symposium on Requirements Engineering, Monterey (USA), September 2003. IEEE Computer Society.
- [Gab89] Dov Gabbay. The declarative past and imperative future. In H. Barringer, editor, Proceedings of the Colloquium on Temporal Logic and Specifications, volume 398 of Lecture Notes in Computer Science, pages 409-448. Springer- Verlag, 1989.
- [GLL99] S. Gnesi, D. Latella, and G. Lenzini. Formal verification of cryptographic protocols using history dependent automata. In Proceedings of the of the 4th Workshop on Sistemi Distribuiti: Algoritmi, Architetture e Linguaggi, 1999.
- [KPV01] O. Kupferman, N. Piterman, and M. Vardi. Extended temporal logic revisited. In Proceedings of the 12th International Conference on Concurrency Theory, number 2154 in Lecture Notes in Computer Science, pages 519-534. Springer Verlag, 2001.
- [vL01] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In Proceedings of the 5th IEEE International Symposium on Requirements Engineering, pages 249-263, 2001
- [LMS02] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In Proceedings of the 17th IEEE Symp. Logic in Computer Science (LICS'2002), pages 383-392, Copenhagen, Denmark, July 2002. IEEE Comp. Soc. Press.
- [NMP+06] D. Nickovic, O. Maler, A. Pnueli, P. Caspi, A. Girard, "Final Proposal for PSL Analog Extensions", PROSYD Deliverable 1.3/2.