



On the Architecture of System Verification Environments

Mark A. Hillebrand¹ Wolfgang J. Paul²

¹German Research for Artificial Intelligence (DFKI), Saarbrücken, Germany

²Department of Computer Science, Saarland University, Saarbrücken, Germany

Haifa Verification Conference, 2007



The Verisoft Project

- Funded by the German government
- Partners from industry and academia
- Goal: pervasively verify four computer systems, three of which in industrial context
 ~> improve quality, increase productivity

The Verisoft Project

- Funded by the German government
- Partners from industry and academia
- Goal: pervasively verify four computer systems, three of which in industrial context
~> improve quality, increase productivity
- Six sub projects:
 - SP1 Tools and methods
 - SP2 Academic System
 - SP3 Processor and system-on-chip
 - SP4 Chipcard-based biometric identification system
 - SP5 Project management
 - SP6 Automotive

The Verisoft Project

- Funded by the German government
- Partners from industry and academia
- Goal: pervasively verify four computer systems, three of which in industrial context
~> improve quality, increase productivity
- Six sub projects:
 - SP1 Tools and methods
 - SP2 Academic System
 - SP3 Processor and system-on-chip
 - SP4 Chipcard-based biometric identification system
 - SP5 Project management
 - SP6 Automotive

The Verisoft Project

- Funded by the German government
- Partners from industry and academia
- Goal: pervasively verify **four** computer systems, three of which in industrial context
~> improve quality, increase productivity
- Six sub projects:
 - SP1 Tools and methods
 - SP2 Academic System**
 - SP3 Processor and system-on-chip**
 - SP4 Chipcard-based biometric identification system**
 - SP5 Project management
 - SP6 Automotive**

The Verisoft Project

- Funded by the German government
- Partners from industry and academia
- Goal: pervasively verify four computer systems, **three** of which in industrial context
~> improve quality, increase productivity
- Six sub projects:
 - SP1 Tools and methods
 - SP2 Academic System
 - SP3 Processor and system-on-chip**
 - SP4 Chipcard-based biometric identification system**
 - SP5 Project management
 - SP6 Automotive**

Outline

- Verisoft Systems and System Layers
- Verified Stacks
 - Computational Models Stack
 - Semantics Stack
 - Example
- Verisoft Repository and Publication
- Conclusion

SP2 Verisoft Academic System



- Scenario: general-purpose computer system

SP2 Verisoft Academic System



- Scenario: general-purpose computer system
- Hardware, microkernel, operating system, applications

SP2 Verisoft Academic System



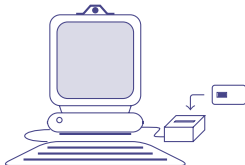
- Scenario: general-purpose computer system
- Hardware, microkernel, operating system, applications
- Services: user processes may use file I/O, IPC, networking via sockets, and RPC

SP2 Verisoft Academic System



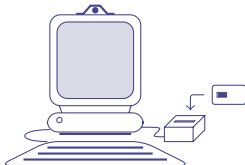
- Scenario: general-purpose computer system
- Hardware, microkernel, operating system, applications
- Services: user processes may use file I/O, IPC, networking via sockets, and RPC
- Application example: everything required to write, sign, send, and receive email

SP4 Verisoft Biometric Identification System



- Scenario: biometric sensor used to authenticate user against biometric reference data stored on smartcard

SP4 Verisoft Biometric Identification System

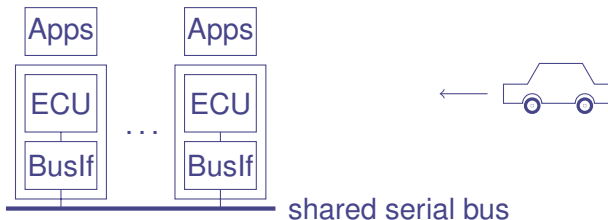


- Scenario: biometric sensor used to authenticate user against biometric reference data stored on smartcard
- Biometric data must be protected (according to German privacy regulations)
- Cryptographic protocol between smartcard and system

SP6 Verisoft Automotive System

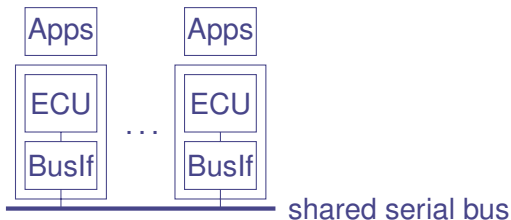


SP6 Verisoft Automotive System



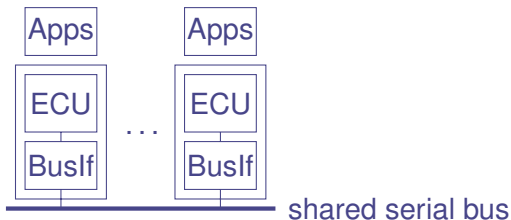
- Scenario: distributed system of electronic control units (ECUs) communicating over time-triggered bus

SP6 Verisoft Automotive System



- Scenario: distributed system of electronic control units (ECUs) communicating over time-triggered bus
- Applications (seem to) use shared variables to communicate
- Application example: automatic emergency call

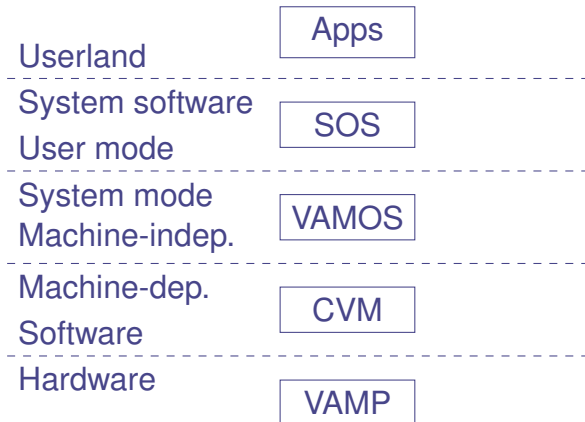
SP6 Verisoft Automotive System



- Scenario: distributed system of electronic control units (ECUs) communicating over time-triggered bus
- Applications (seem to) use shared variables to communicate
- Application example: automatic emergency call
- ! Clock synchronization and WCET analysis required

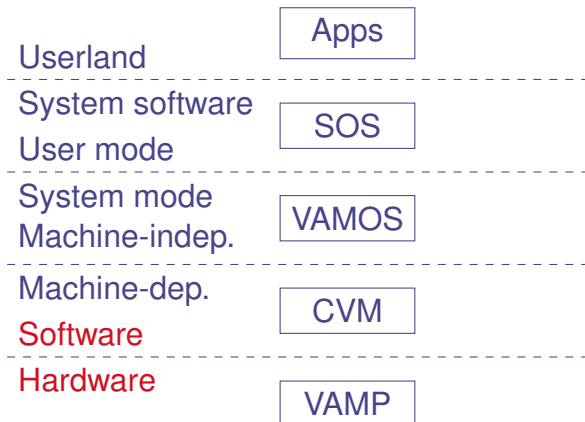
Implementation Languages and Layers

SP2 Academic System / SP4 CBI:



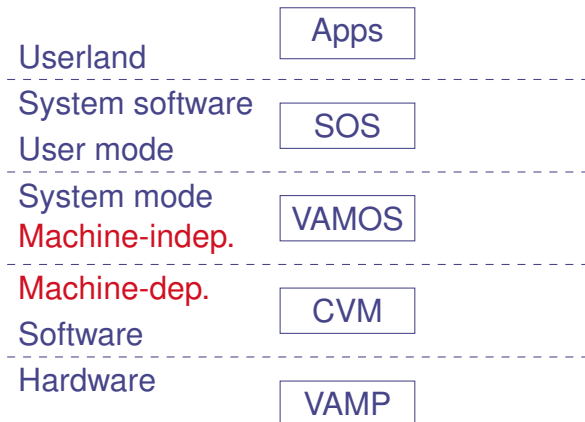
Implementation Languages and Layers

SP2 Academic System / SP4 CBI:



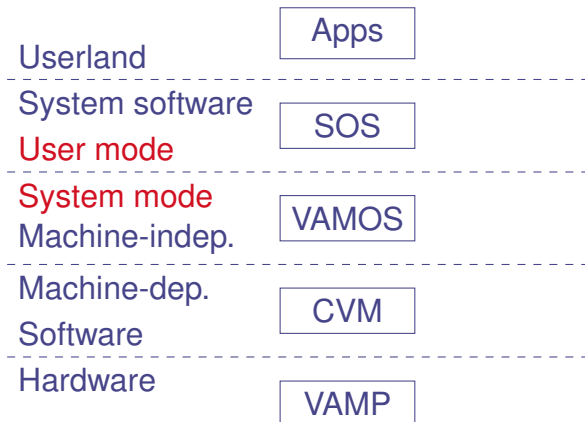
Implementation Languages and Layers

SP2 Academic System / SP4 CBI:



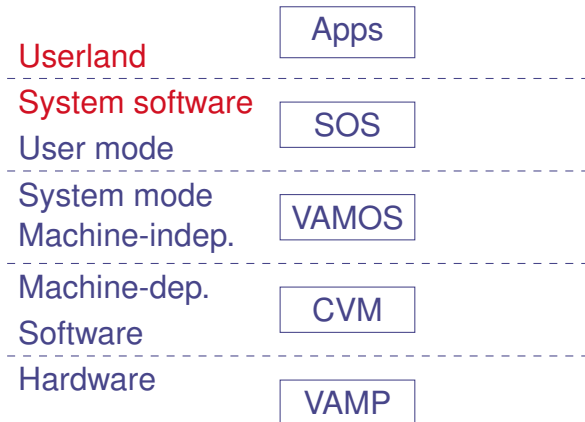
Implementation Languages and Layers

SP2 Academic System / SP4 CBI:



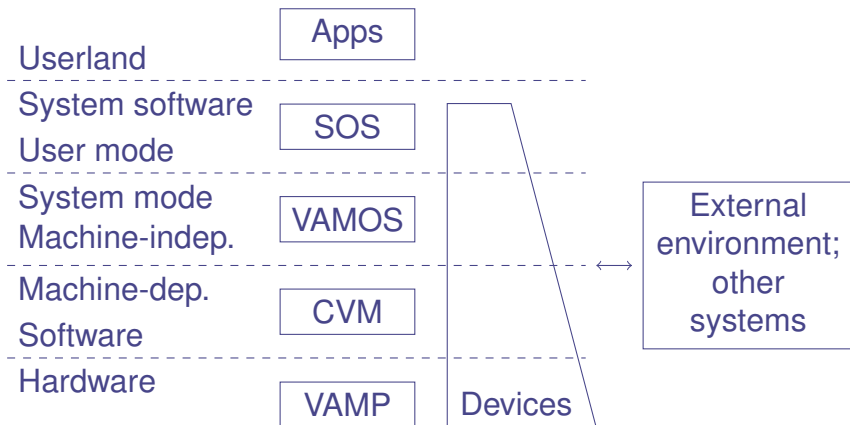
Implementation Languages and Layers

SP2 Academic System / SP4 CBI:



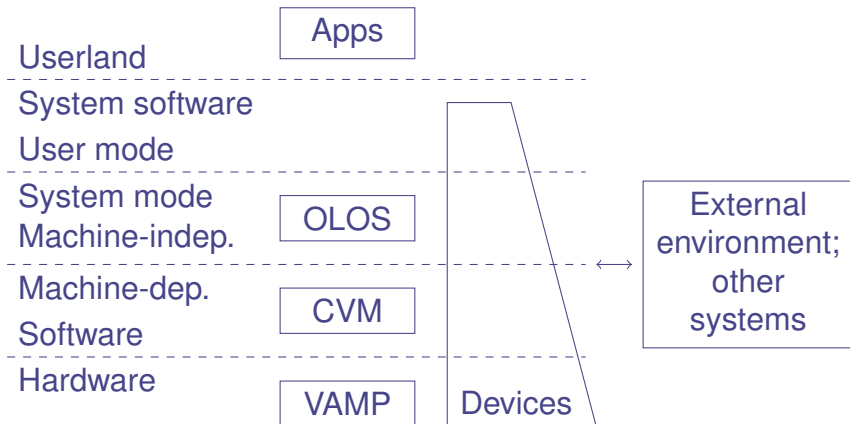
Implementation Languages and Layers

SP2 Academic System / SP4 CBI:



Implementation Languages and Layers

SP6 Automotive System:



System Verification Recipe

Using appropriate formal specification & proof tools:

1. specify layers and languages used in the system
2. specify and verify algorithms used by the tool chain
(or, alternatively, validate their output)
3. prove simulation statements between layers, arguing about the programs residing at the different layers

System Verification Recipe

Using appropriate formal specification & proof tools:

1. specify layers and languages used in the system
2. specify and verify algorithms used by the tool chain
(or, alternatively, validate their output)
3. prove simulation statements between layers, arguing about the programs residing at the different layers

All of this should enable to transfer correctness results for top-layer programs to their bottom-layer representation (\rightsquigarrow obtain: verified stack).

Computational Models

- Language models
 - Register-transfer language
 - Machine language, assembler
 - C0, a type-safe, Pascal-like subset of C

Computational Models

- Language models
 - Register-transfer language
 - Machine language, assembler
 - C0, a type-safe, Pascal-like subset of C
- Device models
 - DFAs modeling device behavior
 - Interact with the local system and an external environment
 - Timing: mostly abstracted; use external environment also to introduce non-determinism
 - ! But not for the automotive system. . .

Computational Models

- Language models
 - Register-transfer language
 - Machine language, assembler
 - C0, a type-safe, Pascal-like subset of C
- Device models
 - DFAs modeling device behavior
 - Interact with the local system and an external environment
 - Timing: mostly abstracted; use external environment also to introduce non-determinism
 - ! But not for the automotive system...
- Both types of models specified by next-state functions of the form $\delta(in, c) = (c', out)$ (small steps!)

Computational Models (cont.)

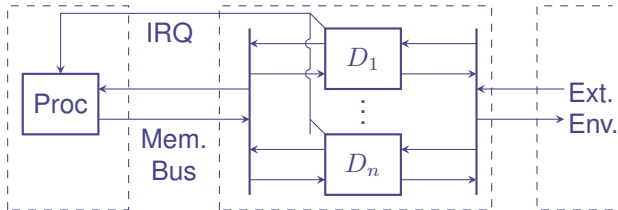
- Build system / layer models from previous models

Computational Models (cont.)

- Build system / layer models from previous models
- Multiple copies of models may be involved
(many devices; virtualisation of resources)
- Concurrency (or even parallelism) is involved
(device communication; interprocess communication)

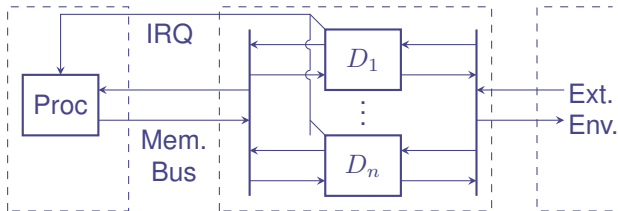
Computational Models (cont.)

- Build system / layer models from previous models
- Multiple copies of models may be involved (many devices; virtualisation of resources)
- Concurrency (or even parallelism) is involved (device communication; interprocess communication)



Computational Models (cont.)

- Build system / layer models from previous models
- Multiple copies of models may be involved (many devices; virtualisation of resources)
- Concurrency (or even parallelism) is involved (device communication; interprocess communication)



- Distributed models: connect via external environment

Semantics Stack for $C0$

- How to prove implementation of a layer correct wrt its model? Small-steps, concurrent semantics too cumbersome to use!

Semantics Stack for $C0$

- How to prove implementation of a layer correct wrt its model? Small-steps, concurrent semantics too cumbersome to use!
 - Stack of $C0$ semantics formalized in Isabelle/HOL
 - Machine-level small steps semantics (memory layout)
 - Small steps semantics
 - Big steps semantics
 - Axiomatic semantics / $C0$ Hoare logics (with VCG)
 - Layers in semantics stack related to each other via equivalence results
- ~> Use $C0$ Hoare logics for the bulk of verification work; integrate automatic proof tools into $C0$ Hoare logics.

Example: Page Fault Handler Verification I

- Memory virtualisation via demand paging
- Implemented in C0 and assembler (swap access)

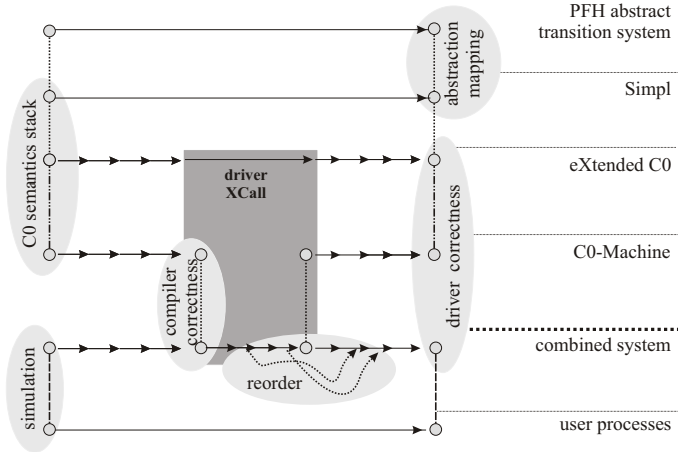
Example: Page Fault Handler Verification I

- Memory virtualisation via demand paging
- Implemented in C0 and assembler (swap access)
- Verification of the C0 part
 - Verify in sequential C0 Hoare logics *enriched with* axiomatic semantics for swap memory access
 - Transfer down to the C0 big-steps level
 - Transfer down to the C0 small-steps level

Example: Page Fault Handler Verification I

- Memory virtualisation via demand paging
- Implemented in C0 and assembler (swap access)
- Verification of the C0 part
 - Verify in sequential C0 Hoare logics *enriched with* axiomatic semantics for swap memory access
 - Transfer down to the C0 big-steps level
 - Transfer down to the C0 small-steps level
- Verification of page in / out operations
 - Verify in assembler model with a hard disk
 - Generalize to model with other devices (trace reordering required!)
 - Wrap as C0 functions, verify wrapper (does not interfere with regular C0 small steps)

Example: Page Fault Handler Verification II



Repository Implementation and Structure

- Internally, we manage
 - documentations,
 - specifications,
 - implementations,
 - proofs, and
 - (proof) tools

of Verisoft project partners in a standard VCS.
(\rightsquigarrow concurrent development, continuous integration and testing)

Repository Implementation and Structure

- Internally, we manage

- documentations,
- specifications,
- implementations,
- proofs, and
- (proof) tools

of Verisoft project partners in a standard VCS.

(\rightsquigarrow concurrent development, continuous integration and testing)

- Everything is organized in *modules*, which have dependencies ('X needs / builds upon on Y').



Repository Implementation and Structure

- Internally, we manage
 - documentations,
 - specifications,
 - implementations,
 - proofs, and
 - (proof) tools
- } modules

of Verisoft project partners in a standard VCS.
(\rightsquigarrow concurrent development, continuous integration and testing)

- Everything is organized in *modules*, which have dependencies ('X needs / builds upon on Y').

Repository Realization

 DEFGEMT VOM Bundesministerium für Bildung und Forschung					
					
Verisoft Repository Modules and Their Dependencies					
Module	Name	Source	Verification Environment	Maintainer	InformationDependencies
Isabelle	Isabelle Theorem Prover	/tools/isabelle		StefanBerghofer TobiasNipkow MakariusWenzel	
libisa	Isabelle Lemma Library	/verification/libisa			Isabelle
NuSMV	NuSMV model checker	/tools/NuSMV		(third party)	
lHaVeIt	Isabelle Hardware Verification Infrastructure	/tools/lHaVeIt		SergeyTverdyshev	Isabelle NuSMV
simplify	Simplify theorem prover	/tools/simplify		(third party)	
swmc	Hoare Logic Software Model Checker Interface	/tools/swmc		StefanMaus	Hoare ACSR Bohne armc3 c0check idList simplify
ACSR	Automatic Checker of Safety Properties Based on Abstraction Refinement	/tools/ACSR		NassimSeghir	
Bohne	Pointer Verification / Shape Analysis Tool	/tools/Bohne		ThomasWies	
armc3	Termination Checker	/tools/armc3		AndreyRybalchenko	simplify
Hoare libHoare	Hoare Logic Library	/tools/Hoare /verification/libHoare		NorbertSchirmer	Isabelle Hoare
Development Environment / Tool Chain					
dkxasm	DLX Assembler	/tools/dkxasm		MarkHillebrand	
dkxsim	DLX Simulator	/tools/dkxsim		MarkHillebrand	
c0check	C0 checker and preprocessor	/tools/c0check		DirkLeinenbach ArtemStarostin	ic0compiler idList iString
stdinclude	Standard C0 Include Files	/implementation/software/include		MatthiasDaum	
rpcgen	SOS RPC Interface Generator	/tools/rpcgen		AndreyShadrin	libbpn libsos
VAMP-FPGA	VAMP-FPGA	/tools/VAMP-FPGA		DirkLeinenbach	lHaVeIt


Repository Publication

- Goal: make stable, self-contained snapshots of non-confidential parts of the internal repository available publicly.


Repository Publication

- Goal: make stable, self-contained snapshots of non-confidential parts of the internal repository available publicly.
- Currently published:
 - Code verification of a doubly-linked list library
 - Code verification of a string library
 - Code verification of the Verisoft email client
 - Code verification of a big integer library
 - Code verification of the C0 compiler
(includes: assembler and C0 small-steps semantics)

Repository Publication



GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung



- ▶ Home
- ▶ Consortium
- ▶ Project Structure
- ▶ Goals and Results
 - ▶ SP1: Methods and Tools
 - ▶ SP2: Academic System
 - ▶ SP3: Correct Industrial Hardware/Software-System
 - ▶ SP4: Biometric Identification System
 - ▶ SP5: Project Management
 - ▶ SP6: Automotive
 - ▶ Verisoft Repository
- ▶ Publications
- ▶ Press
- ▶ Contact
- ▶ Internal
- ▶ Auf Deutsch, Bitte!

Verisoft Repository

In the Verisoft project, the formal pervasive verification of four exemplary computer systems, three of which come from the industrial sector, is attempted. The layers, which are considered, range from the gate-level hardware over system software to communicating and distributed applications.

The Verisoft Repository allows to concurrently develop the many individual results that contribute to the overall verification results and to manage them in a tractable manner.

In the repository, the artefacts developed by the project partners are being collected as *modules* and related to each other via *dependencies*. Modules include documentations, specifications, implementations, proofs, and tools for development and verification.

In the end, the repository must be self-contained: the set of modules for a given computer system under verification must allow

- to build an executable implementation of that system and
- to prove its top-level correctness.

Currently, substantial parts of the Verisoft theories and systems have been imported into the Verisoft repository and the repository is being used for further development.

At this location, portions of the internal Verisoft repository that appear sufficiently stable and do not contain confidential data of industry partners, will be published.

Publications

- [vString-4<11594.tar.gz \(1.8M\)](#) — Code-Level Verification of a String Library (14 Dec 2006)

This release contains the code-level verification of a doubly-linked list library and a string library; the implementation language is the C-like programming language C0. The verification is done in a Hoare-logic-based interactive software verification environment for the theorem proving environment Isabelle, which is also included.

- [vemall-trunk-115868.tar.gz \(3.6M\)](#) — Code-Level Verification of an Email Client (18 May 2007)

This release contains the code-level verification of the email client of [Subproject 2: Academic System](#) relative to the services provided by the simple operating system (SOS) and applications for signing and email transfer. The implementation language is the C-like programming language C0. The verification is done in a Hoare-logic-based interactive software verification environment for the theorem proving environment Isabelle.


- [vbigint-trunk-119285.tar.gz \(2.2M\)](#) — Code-Level Verification of a Big Integer Library (10 October 2007)

This release contains the code-level verification of a big integer library used in [Subproject 2: Academic System](#) and [Subproject 4: Biometric Identification System](#) for the implementation of cryptographic primitives. The implementation language is the C-like programming language C0. The verification is done in a Hoare-logic-based interactive software verification environment for the theorem proving environment Isabelle.


Revision 10 Oct 2007

www.verisoft.de/VerisoftRepository.html

Repository Publication



GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung



Verification

Repository Publication

Overview

Verisoft's sub project 2 deals with the formal pervasive verification of a general-purpose computer system from gate-level hardware, to system software, and communicating applications. All code is implemented in the programming language C0, which is a subset of C. However, software verification in Verisoft does not stop at the C0 level. To allow execution of verified programs on the real hardware they must be compiled to binary code. This translation could itself introduce errors into an otherwise verified C0 program. Thus, verification of the translation process is essential for pervasive system verification when the system software and applications are implemented in a high-level programming language.

We have verified a simple non-optimizing C0 compiler. For pervasive verification it is not sufficient to have a verified code generation algorithm (also called compiling specification). We also need a verified compiler implementation in C0 (excluding parsing phase or I/O operations) which allows us (after boot strapping) to execute a verified compiler binary on the target platform. After verifying the correctness of the compiling specification it is sufficient to show that the compiler implementation produces the same code as the compiling specification.

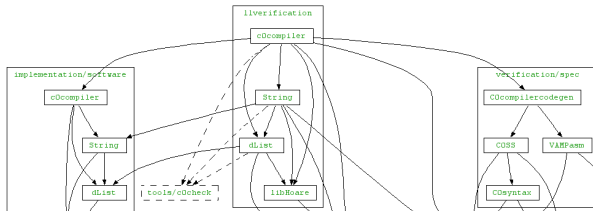
Here, we include only the compiler implementation and the corresponding correctness proofs. The correctness proof for the compiling specification will be published in an upcoming Verisoft repository release.

In addition to the compiler implementation, the implementation and verification of the additional libraries (for strings and lists) are also included here; please see the previous repository release ([vString-4-r11594.tar.gz](#) on the [Verisoft Repository page](#)) covering these verifications.

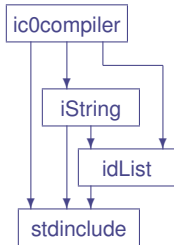
The verification of the compiler implementation is done in a Hoare logic verification environment, which is implemented in the theorem prover Isabelle/HOL. For this purpose, the C0 implementations (in concrete syntax) are translated into their Hoare logic representation; the translator itself is also part of this release ([c0check](#)). Based on this representation, Hoare triples for total and partial correctness are proven, supported by a verification condition generator. Additionally, the absence of certain runtime errors is shown (e.g., integer overflows and out-of-bounds array access). Absence of runtime errors is necessary (among other things), to translate the total correctness results down to lower-level semantics, i.e., in the end to the compiled program running on the target architecture.

As mentioned above, the verified implementation of the C0 compiler lacks a front-end and output routines. These are also implemented in the unverified [c0check](#) tool.

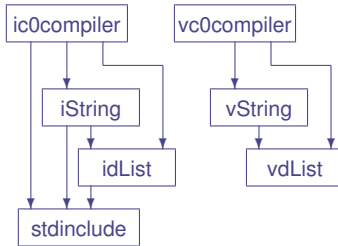
The files in this release are placed in a number of directories, which we also call *modules*. The following graph lists the modules present in this release and also indicates the dependencies between the modules; the top-level module corresponds to the top-level results, i.e., the code verification of the C0 compiler.



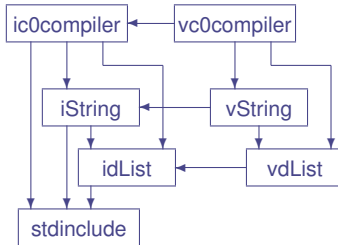
Dependencies' Example: Compiler



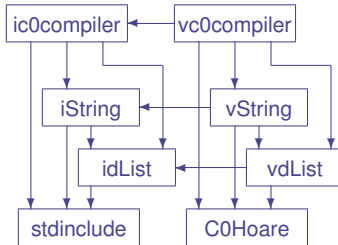
Dependencies' Example: Compiler



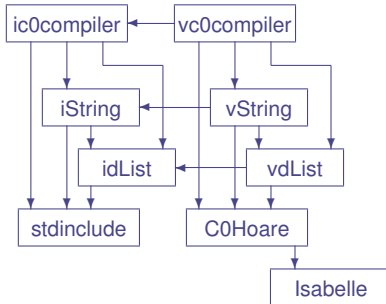
Dependencies' Example: Compiler



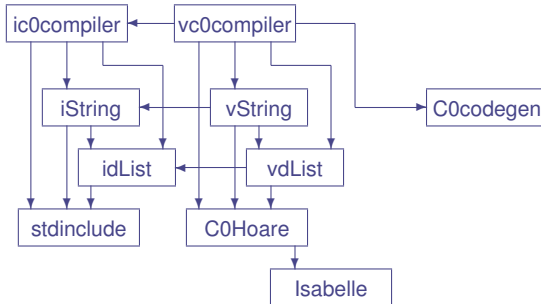
Dependencies' Example: Compiler



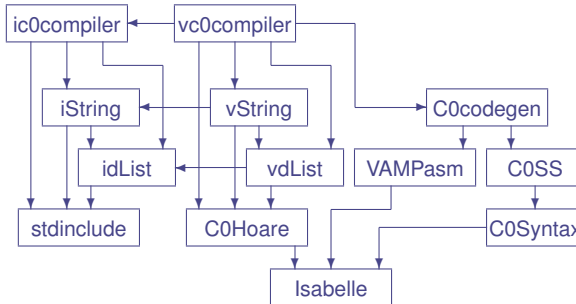
Dependencies' Example: Compiler



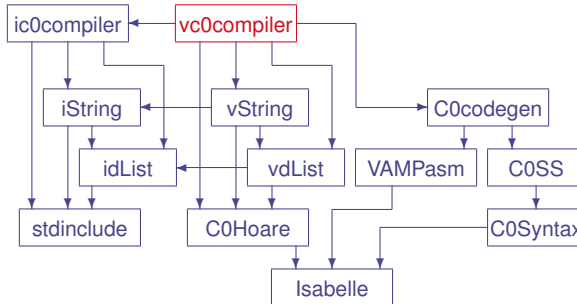
Dependencies' Example: Compiler



Dependencies' Example: Compiler

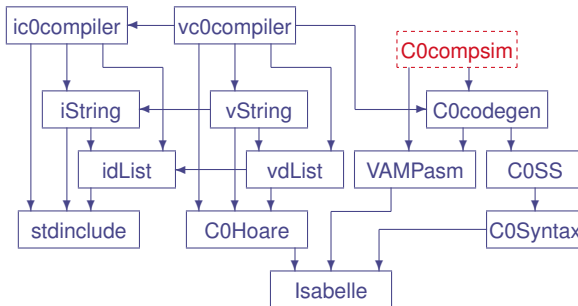


Dependencies' Example: Compiler



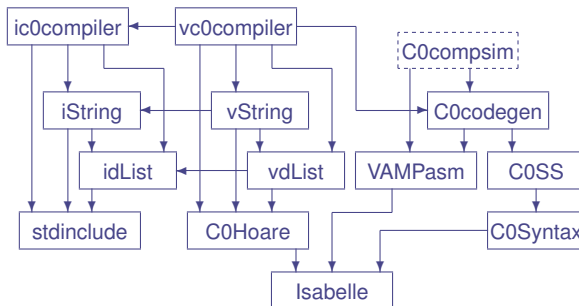
- Theorem: Compiler implementation produces the same code than a code generation algorithm

Dependencies' Example: Compiler



- Theorem: Compiler implementation produces the same code than a code generation algorithm
- Theorem: Generated code simulates C0 computation (part of an upcoming Verisort repository release)

Dependencies' Example: Compiler



- Theorem: Compiler implementation produces the same code than a code generation algorithm
- Theorem: Generated code simulates C0 computation (part of an upcoming Verisoft repository release)
- To use that theorem (e.g., bootstrap): more nodes. . .

Conclusion

- Verisoft: verification of entire systems of industrial interest
- System verification environment / repository:
 - Contains all artifacts needed for the verification
 - Architecture largely determined by structure of implementation and its tool chain
- Two verified stacks:
 - computational models (often concurrent, small steps)
 - semantics (increase verification productivity)
- Repository snapshots: `www.verisoft.de`