CIVL
Concurrency Intermediate Verification Language

Tutorial

Stephen F. Siegel

Department of Computer and Information Sciences
University of Delaware

December 9, 2013

What is CIVL?

CIVL is . . .

1. . . . a programming language, CIVL-C
   - based on subset of C
   - extensions for concurrency, naming of scopes
2. . . . a suite of tools for analyzing CIVL-C programs
   - running + dynamic checking
   - model checking
   - static analyses (coming)
3. . . . a set of translators from common programming language/concurrency API combinations to CIVL-C
   - coming

Example: adder.cvl

```
#include <civlc.h>

$input int B;
$input int N;
$assume 0<=N && N<=B;
$input double a[N];

double adderSeq(double *p, int n) {
  double s = 0.0;

  for (int i = 0; i < n; i++)
    s += p[i];
  return s;
}

double adderPar(double *p, int n) {
  double s = 0.0;
  _Bool mutex = 0;
  $proc workers[n];

  void worker(int i) {
    double t;
    $when (mutex == 0) mutex = 1;
    t = s;
    t += p[i];
    s = t;
    mutex = 0;
  }

  for (int j = 0; j < n; j++)
    workers[j] = $spawn worker(j);
  for (int j = 0; j < n; j++)
    $wait workers[j];
  return s;
}

void main() {
  double seq = adderSeq(&a[0], N);
  double par = adderPar(&a[0], N);

  $assert seq == par;
}
```

Verifying adder.cvl

```
concurrency$ civl verify -inputB=5 adder.cvl
CIVL v0.4 of 2013-12-06 -- http://vsl.cis.udel.edu/civl
=================== Stats ===================
   validCalls             : 23883
   proverCalls            : 29
   memory (bytes)         : 374341632
   time (s)               : 5.35
   maxProcs               : 6
   statesInstantiated     : 28761
   statesSaved            : 3082
   statesSeen             : 3082
   statesMatched          : 1968
   transitions            : 5049

The standard properties hold for all executions.
concurrency$
```

Download and Installation

1. Get a Java 7 VM.
2. Go to `http://vsl.cis.udel.edu/civl`
3. Navigate to downloads, *latest stable release*.
4. Download version corresponding to your platform.
   - for now, pre-compiled versions for OS X and linux (32- and 64-bit)
   - other platforms must build from source
5. Unpack and move resulting directory `CIVL-`*`tag`* under `/opt`.
6. Download the VSL dependencies archive.
   - contains a number of pre-compiled open source libraries used by CIVL
   - `http://vsl.cis.udel.edu/tools/vsl_depend`
7. Unpack and move resulting directory `vsl` under `/opt`.
8. Put `/opt/CIVL-`*`tag`*`/bin/civl` in your path.
   - however you want: move it, symlink, . . .

Test your installation

From command line . . .

```
concurrency$ civl
CIVL v0.4 of 2013-12-06 -- http://vsl.cis.udel.edu/civl
Missing command
Type "civl help" for command line syntax.

concurrency$ civl help
...
```

Copy `/opt/CIVL-`*tag*`/examples/concurrency/adder.cvl` to your working directory and try

```
    civl verify -inputB=5 adder.cvl
```

What features are inherited from C?

- most syntax
- types
  - `_Bool` $\to \{0,1\}$
  - `int`, `long`, `short`, ... $\to \mathbb{Z}$
  - `double`, `float`, ... $\to \mathbb{R}$
  - structure, array, pointer, and function types
- expressions
  - addition, multiplication, division, subtraction, unary minus (`+`, `*`, `/`, `-`)
  - integer division (`/`) and modulus (`%`)
  - pointer dereference (`*`), address-of (`&`)
  - array subscript (`[...]`)
  - structure navigation (`.`)
  - logical and (`&&`), or (`||`), not (`!`)
  - `==`, `!=`, `<`, `>`, `<=`, `>=`
  - pointer addition (`+`) and subtraction (`-`)
  - `++`, `--`
  - no bit-wise operations for now

Inherited from C, cont.

- statements
  - no-op, labeled-statement, compound-statement
  - assignments (`=`, `+=`, `-=`, ...)
  - function call
  - `if`...`else`
  - `goto`, `while`, `do`, `for`, `switch`, `break`
- procedure (function) prototypes and definitions
- `typedef`
- preprocessing directives

New features

- functions can be declared in any scope
- concurrency primitives
  - spawning processes, waiting for a process to terminate, guarded commands
  - nondeterministic choice
  - explicit naming of scopes
  - scope-parameterized pointers
  - other primitives useful for verification
    - input qualifier, assert, assume, procedure contracts
- library-level constructs supporting message-passing, . . .

Some CIVL-C primtives

`$proc`	the process type
`$self`	the evaluating process (constant of type `$proc`)
`$scope`	the scope type
`$input`	type qualifier declaring variable to be a program input
`$output`	type qualifier declaring variable to be a program output
`$spawn`	create a new process running procedure
`$wait`	wait for a process to terminate
`$assert`	check something holds
`$true`	boolean value true, used in assertions
`$false`	boolean value false, used in assertions
`$assume`	assume something holds
`$when`	guarded statement
`$choose`	nondeterministic choice statement
`$choose_int`	nondeterministic choice of integer

CIVL Command line tools

- `civl run filename`
  - run the CIVL program making nondeterministic choices randomly
  - `-seed=LONG` : use this random seed (reproducible)
- `civl verify filename`
  - explore reachable state space, checking properties at each state
    - absence of deadlock, assertion violations, division by 0, invalid pointer dereference, out of bounds array access, . . .
  - may specify bounds using `$input` variables and command line
    - `-inputX=value`
  - `-errorBound=INT` specifies maximum number of errors that will be logged before quitting
- `civl replay`
  - if a violation was found during `verify`, its trace is saved to a file; this will run the trace
  - `-id=INT` can be used to specify the ID of the trace if more than one
  - `-trace=tracefile` can be used to specify the exact filename containing trace

Scope-parameterized pointers

- a declaration of the form `$scope s;` assigns the name `s` to the containing scope
  - what you can do with `s` is very limited
  - cannot be assigned, passed as parameter
- `int *<s> p;`
  - declares `p` to have type "pointer-to-`int`-in-`s`"
  - `p` can only hold a pointer to an object in scope `s`

Message Passing example: ring.cvl

```
/* Create nprocs processes.  Have them exchange data
 * in a cycle.  Commandline example:
 *     civl verify -inputNPROCS=3 ring.cvl -simplify=false
 */
#include<civlc.h>
#include "mp_root.cvh"

void MPI_Process (int rank) {
#include "mp_proc.cvh"

  double x=rank, y;

  send(&x, 1, (rank+1)%NPROCS, 0);
  recv(&y, 1, (rank+NPROCS-1)%NPROCS, 0);
  $assert y==(rank+NPROCS-1)%NPROCS;
}
```

File mp_root.cvh

```
$input int NPROCS;
$proc __procs[NPROCS];
_Bool __start = 0;
$comm MPI_COMM_WORLD;

void MPI_Process (int rank);

void init() {
  for (int i=0; i<NPROCS; i++)
    __procs[i] = $spawn MPI_Process(i);
  MPI_COMM_WORLD = $comm_create(NPROCS, __procs);
  __start=1;
}

void finalize() {
  for (int i=0; i<NPROCS; i++)
    $wait __procs[i];
}

void main() {
  init();
  finalize();
}
```

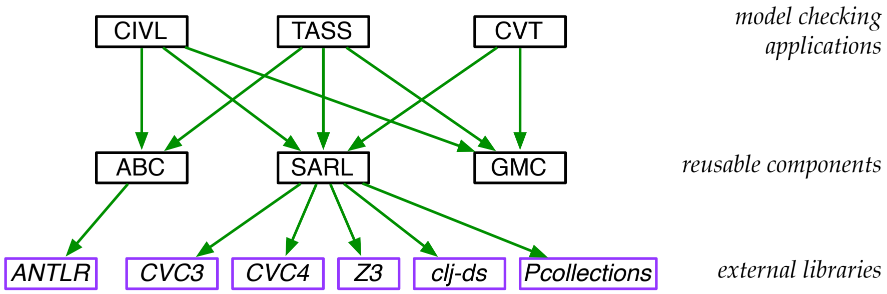
File mp_proc.cvh

```
void send(void *buf, int count, int dest, int tag) {
  $message out = $message_pack(rank, dest, tag, buf, count*sizeof(double));
  $comm_enqueue(&MPI_COMM_WORLD, out);
}

void recv(void *buf, int count, int source, int tag) {
  $message in = $comm_dequeue(&MPI_COMM_WORLD, source, rank, tag);
  $message_unpack(in, buf, count*sizeof(double));
}

$when (__start);
```

VSL Projects: Uses Relation



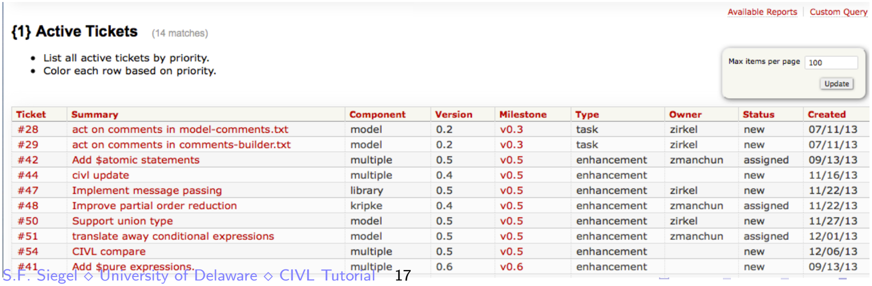
- reusable components
  - ABC: A Better C compiler? ANTLR-Based C compiler?
  - SARL: Symbolic Algebra & Reasoning Library
  - GMC: Generic Model Checking utilities
    - DFS, command line interface, trace saving/replay, error logging, random simulation
- model checking applications
  - CIVL: Concurrency Intermediate Verification Language
  - TASS: Toolkit for Accurate Scientific Software (C+MPI)
  - CVT: Chapel Verification Tool

Engineering

- all of the VSL software is in Java
- try to maintain coding standards
- clear module boundaries with interfaces

Web page	`http://vsl.cis.udel.edu/civl`
Subversion	`svn://vsl.cis.udel.edu/civl`
Trac repository	`https://vsl.cis.udel.edu/trac/civl`
Automated build/test	`http://vsl.cis.udel.edu/civl/test`

- replace `civl` with `sarl`, `abc`, `gmc`, or `tass`

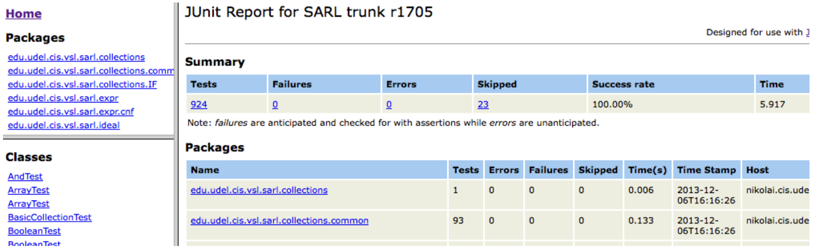


{1} Active Tickets (14 matches)

- List all active tickets by priority.
- Color each row based on priority.

Ticket	Summary	Component	Version	Milestone	Type	Owner	Status	Created
#28	act on comments in model-comments.txt	model	0.2	v0.3	task	zirkel	new	07/11/13
#29	act on comments in comments-builder.txt	model	0.2	v0.3	task	zirkel	new	07/11/13
#42	Add $atomic statements	multiple	0.5	v0.5	enhancement	zmanchun	assigned	09/13/13
#44	civl update	multiple	0.4	v0.5	enhancement		new	11/16/13
#47	Implement message passing	library	0.5	v0.5	enhancement	zirkel	new	11/22/13
#48	Improve partial order reduction	kripke	0.4	v0.5	enhancement	zmanchun	assigned	11/22/13
#50	Support union type	model	0.5	v0.5	enhancement	zirkel	new	11/27/13
#51	translate away conditional expressions	model	0.5	v0.5	enhancement	zmanchun	assigned	12/01/13
#54	CIVL compare	multiple	0.5	v0.5	enhancement		new	12/06/13
#41	Add $pure expressions.	multiple	0.6	v0.6	enhancement		new	09/13/13

Automated Build & Test Script



Home

Packages

- edu.udel.cis.vsl.sarl.collections
- edu.udel.cis.vsl.sarl.collections.comm
- edu.udel.cis.vsl.sarl.collections.IF
- edu.udel.cis.vsl.sarl.expr
- edu.udel.cis.vsl.sarl.expr.cnf
- edu.udel.cis.vsl.sarl.ideal

Classes

- AndTest
- ArrayTest
- ArrayTest
- BasicCollectionTest
- BooleanTest
- BooleanTest

JUnit Report for SARL trunk r1705

Designed for use with J

Summary

Tests	Failures	Errors	Skipped	Success rate	Time
924	0	0	23	100.00%	5.917

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Name	Tests	Errors	Failures	Skipped	Time(s)	Time Stamp	Host
edu.udel.cis.vsl.sarl.collections	1	0	0	0	0.006	2013-12-06T16:16:26	nikolai.cis.ude
edu.udel.cis.vsl.sarl.collections.common	93	0	0	0	0.133	2013-12-06T16:16:26	nikolai.cis.ude

For each project . . .

- script is run after each commit
- one directory for each branch and trunk
  - one subdirectory for each revision, up to some bounded history
- compiles all code and displays results
- runs JUnit test suite and displays results
- runs Jacoco coverage anaysis and displays results
- generates javadocs

Developer Eclipse Set-up

1. Download vsl dependencies archive from `http://vsl.cis.udel.edu/tools/vsl_depend`
2. Download and install Eclipse IDE for Java EE Developers
   - version Kepler or later
3. Install Apache Ant if you don't have it
4. Install an Eclipse SVN plugin (such as Subversive)
5. Create class path variable `VSL`:
   - Preferences→Java→Build Path→ClassPath Variables
   - select "New" and create a classpath variable `VSL`
   - specify its value to be `/opt/vsl`
6. Create string variable `vsl_lib`:
   - Preferences→Run/Debug→String Substitution→New
   - define an entry `vsl_lib`
   - set its value to be `/opt/vsl/lib`

Check out and install ABC

1. Check out ABC Eclipse project
   - “New Project. . . from SVN”
   - SVN repository: `svn://vsl.cis.udel.edu/abc`
   - Navigate and select `trunk` from within archive
   - Check out project using all default options
2. Build using Ant
   - right-click on `build.xml`
   - Choose “Run as Ant build”
   - Clean project
3. test the build
   - select Run→Run Configurations. . .
   - ceate a new JUnit 4 configuration called “ABC Tests”
   - select “Run all tests in the selected project. . . ”
   - navigate and select the `test` folder in the ABC project
   - under the Arguments tab, type `-ea` into the VM arguments field
   - click “Run” to run the tests

Check out and install GMC

1. Check out GMC Eclipse project
    - “New Project. . . from SVN”
    - SVN repository: `svn://vsl.cis.udel.edu/gmc`
    - Navigate and select `trunk` from within archive
    - Check out project using all default options
2. test the build
    - select Run→Run Configurations. . .
    - ceate a new JUnit 4 configuration called “GMC Tests”
    - select “Run all tests in the selected project. . . ”
    - navigate and select the `test` folder in the GMC project
    - under the Arguments tab, type `-ea` into the VM arguments field
    - click “Run” to run the tests

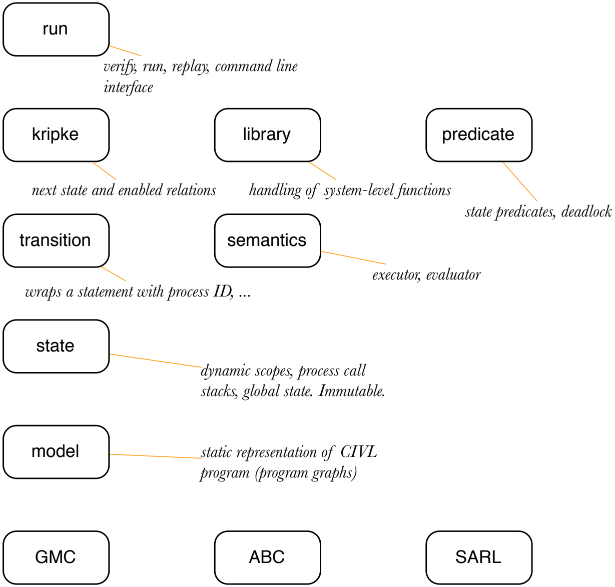
Check out and install SARL

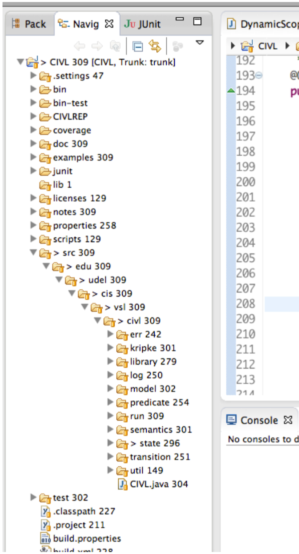
1. Check out SARL Eclipse project
   - “New Project. . . from SVN”
   - SVN repository: `svn://vsl.cis.udel.edu/sarl`
   - Navigate and select `trunk` from within archive
   - Check out project using all default options
2. test the build
   - select Run→Run Configurations. . .
   - ceate a new JUnit 4 configuration called “SARL Tests”
   - select “Run all tests in the selected project. . . ”
   - navigate and select the `test` folder in the SARL project
   - under Arguments tab, type `-ea` into the VM arguments field
   - under Environment tab, create an entry `DYLD_LIBRARY_PATH` (OS X) or `LD_LIBRARY_PATH` (linux), specify its value by clicking Variables, choose `vsl_lib` from the list
   - click “Run” to run the tests

Check out and install CIVL

1. Check out CIVL Eclipse project
   - “New Project. . . from SVN”
   - SVN repository: `svn://vsl.cis.udel.edu/civl`
   - Navigate and select `trunk` from within archive
   - Check out project using all default options
2. test the build
   - select Run→Run Configurations. . .
   - ceate a new JUnit 4 configuration called “CIVL Tests”
   - select “Run all tests in the selected project. . . ”
   - navigate and select the `test` folder in the CIVL project
   - under Arguments tab, type `-ea` into the VM arguments field
   - under Environment tab, create an entry `DYLD_LIBRARY_PATH` (OS X) or `LD_LIBRARY_PATH` (linux), specify its value by clicking Variables, choose `vsl_lib` from the list
   - click “Run” to run the tests

CIVL modules


run
verify, run, replay, command line interface
kripke
next state and enabled relations
library
handling of system-level functions
predicate
state predicates, deadlock
transition
wraps a statement with process ID, ...
semantics
executor, evaluator
state
dynamic scopes, process call stacks, global state. Immutable.
model
static representation of CIVL program (program graphs)
GMC
ABC
SARL



```
▼ > CIVL 309 [CIVL, Trunk: trunk]
  ▶ .settings 47
  ▶ bin
  ▶ bin-test
  ▶ CIVLREP
  ▶ coverage
  ▶ doc 309
  ▶ examples 309
  ▶ junit
    lib 1
  ▶ licenses 129
  ▶ notes 309
  ▶ properties 258
  ▶ scripts 129
  ▼ > src 309
    ▼ > edu 309
      ▼ > udel 309
        ▼ > cis 309
          ▼ > vsl 309
            ▼ > civl 309
              ▶ err 242
              ▶ kripke 301
              ▶ library 279
              ▶ log 250
              ▶ model 302
              ▶ predicate 254
              ▶ run 309
              ▶ semantics 301
              ▶ > state 296
              ▶ transition 251
              ▶ util 149
                CIVL.java 304
  ▶ test 302
    .classpath 227
    .project 211
    build.properties
```

No consoles to d