Starexec for Termination

Johannes Waldmann, HTWK Leipzig, Germany

SE'12

Summary: Termination Competitions

Automatically decide termination of programs
in various models of computation.
yearly since 2003, 23 solvers, 9 categories, 36 people, `http://www.termination-portal.org/`

basic model (easy for Star-Exec):

- input (benchmark): a program
- out: YES/NO + proof trace (informal or formal)

extensions (challenging for Star-Exec?):

- (polynomial) derivational complexity
- machine verification of formal proof traces
  as part of the competition

Summary: Termination Competitions

Automatically decide termination of programs
in various models of computation.
yearly since 2003, 23 solvers, 9 categories, 36 people, `http://www.termination-portal.org/`
basic model (easy for Star-Exec):

- input (benchmark): a program
- out: YES/NO + proof trace (informal or formal)

extensions (challenging for Star-Exec?):

- (polynomial) derivational complexity
- machine verification of formal proof traces
  as part of the competition

Summary: Termination Competitions

Automatically decide termination of programs
in various models of computation.
yearly since 2003, 23 solvers, 9 categories, 36 people, `http://www.termination-portal.org/`
basic model (easy for Star-Exec):

- input (benchmark): a program
- out: YES/NO + proof trace (informal or formal)

extensions (challenging for Star-Exec?):

- (polynomial) derivational complexity
- machine verification of formal proof traces
  as part of the competition

Termcomp Categories

given by (not completely orthogonal) combination of

- models of computation:
  term rewriting (first order, higher order), string rewriting, Prolog, Haskell, Java Bytecode.
- variants of models:
  e.g., Prolog with/without Cut, rewriting modulo theory (AC,...), restricted by strategy
- question:
  - termination: YES/NO
  - polynomial derivational complexity: YES(degree)/NO
- proof trace: formal or not

Termcomp Categories

given by (not completely orthogonal) combination of

- models of computation:
  term rewriting (first order, higher order), string rewriting, Prolog, Haskell, Java Bytecode.
- variants of models:
  e.g., Prolog with/without Cut, rewriting modulo theory (AC,...), restricted by strategy
- question:
  - termination: YES/NO
  - polynomial derivational complexity: YES(degree)/NO
- proof trace: formal or not

Termcomp Categories

given by (not completely orthogonal) combination of

- models of computation:
  term rewriting (first order, higher order), string rewriting, Prolog, Haskell, Java Bytecode.
- variants of models:
  e.g., Prolog with/without Cut, rewriting modulo theory (AC,...), restricted by strategy
- question:
  - termination: YES/NO
  - polynomial derivational complexity: YES(degree)/NO
- proof trace: formal or not

Termcomp Categories

given by (not completely orthogonal) combination of

- models of computation:
  term rewriting (first order, higher order), string rewriting, Prolog, Haskell, Java Bytecode.
- variants of models:
  e.g., Prolog with/without Cut, rewriting modulo theory (AC,...), restricted by strategy
- question:
  - termination: YES/NO
  - polynomial derivational complexity: YES(degree)/NO
- proof trace: formal or not

Termcomp Categories

given by (not completely orthogonal) combination of

- models of computation:
  term rewriting (first order, higher order), string rewriting, Prolog, Haskell, Java Bytecode.
- variants of models:
  e.g., Prolog with/without Cut, rewriting modulo theory (AC,...), restricted by strategy
- question:
  - termination: YES/NO
  - polynomial derivational complexity: YES(degree)/NO
- proof trace: formal or not

Termcomp Categories

given by (not completely orthogonal) combination of

- models of computation:
  term rewriting (first order, higher order), string rewriting, Prolog, Haskell, Java Bytecode.
- variants of models:
  e.g., Prolog with/without Cut, rewriting modulo theory (AC,...), restricted by strategy
- question:
  - termination: YES/NO
  - polynomial derivational complexity: YES(degree)/NO
- proof trace: formal or not

termcomp platform and data

- developed and hosted at research group Computational Logic at U Innsbruck, Austria
- used in competitions since 2008
- cummulative for 2008–2011 competitions: 8950 benchmarks (TPDB), 83 solver versions, 114194 results (job pairs), 10750 formal proof traces
- a “full run” took 419 hours
- hard/software: machine with 16 cores, CentOS, JBoss/Seam, Postgres, JSP.
- we are willing to share *experience* and *code*

termcomp platform and data

- developed and hosted at research group Computational Logic at U Innsbruck, Austria
- used in competitions since 2008
- cummulative for 2008–2011 competitions: 8950 benchmarks (TPDB), 83 solver versions, 114194 results (job pairs), 10750 formal proof traces
- a “full run” took 419 hours
- hard/software: machine with 16 cores, CentOS, JBoss/Seam, Postgres, JSP.
- we are willing to share *experience* and *code*

termcomp platform and data

- developed and hosted at research group Computational Logic at U Innsbruck, Austria
- used in competitions since 2008
- cummulative for 2008–2011 competitions: 8950 benchmarks (TPDB), 83 solver versions, 114194 results (job pairs), 10750 formal proof traces
- a “full run” took 419 hours
- hard/software: machine with 16 cores, CentOS, JBoss/Seam, Postgres, JSP.
- we are willing to share *experience* and *code*

termcomp platform and data

- developed and hosted at research group Computational Logic at U Innsbruck, Austria
- used in competitions since 2008
- cummulative for 2008–2011 competitions: 8950 benchmarks (TPDB), 83 solver versions, 114194 results (job pairs), 10750 formal proof traces
- a “full run” took 419 hours
- hard/software: machine with 16 cores, CentOS, JBoss/Seam, Postgres, JSP.
- we are willing to share *experience* and *code*

termcomp platform and data

- developed and hosted at research group Computational Logic at U Innsbruck, Austria
- used in competitions since 2008
- cummulative for 2008–2011 competitions: 8950 benchmarks (TPDB), 83 solver versions, 114194 results (job pairs), 10750 formal proof traces
- a “full run” took 419 hours
- hard/software: machine with 16 cores, CentOS, JBoss/Seam, Postgres, JSP.
- we are willing to share *experience* and *code*

termcomp platform and data

- developed and hosted at research group Computational Logic at U Innsbruck, Austria
- used in competitions since 2008
- cummulative for 2008–2011 competitions: 8950 benchmarks (TPDB), 83 solver versions, 114194 results (job pairs), 10750 formal proof traces
- a “full run” took 419 hours
- hard/software: machine with 16 cores, CentOS, JBoss/Seam, Postgres, JSP.
- we are willing to share *experience* and *code*

Nice to have (and we already have it)

- data model:
  - *Solver* is a set of *Implementations*
  - solver is registered for competition category
  - *Team* is a set of persons
  - team maintains set of solvers
  - teams have quotas (CPU time, disk space)
- after upload of new implementation, it is automatically run on a subset of benchmarks
- displays:
  - termcomp start page show category summaries of current competition
  - and “news feed” of 10 most recent “jobs pairs”
  - category results shown as table, configurable

Nice to have (and we already have it)

- data model:
  - *Solver* is a set of *Implementations*
  - solver is registered for competition category
  - *Team* is a set of persons
  - team maintains set of solvers
  - teams have quotas (CPU time, disk space)
- after upload of new implementation, it is automatically run on a subset of benchmarks
- displays:
  - termcomp start page show category summaries of current competition
  - and “news feed” of 10 most recent “jobs pairs”
  - category results shown as table, configurable

Nice to have (and we already have it)

- data model:
  - *Solver* is a set of *Implementations*
  - solver is registered for competition category
  - *Team* is a set of persons
  - team maintains set of solvers
  - teams have quotas (CPU time, disk space)
- after upload of new implementation, it is automatically run on a subset of benchmarks
- displays:
  - termcomp start page show category summaries of current competition
  - and “news feed” of 10 most recent “jobs pairs”
  - category results shown as table, configurable

Important to have: Validation

termination competition consists of *two phases*:

1. *solvers* run on benchmarks, emit proof traces
2. *matcher* (postproc.) checks that trace matches benchmark
3. *validators* run on traces

(non)termination proof trace $\approx$ model, or unsat core.

automatic validation is highly recommended:

- advance formalized mathematics (validator source code is extracted from formal proof)
- discover bugs in solvers

We (termcomp) definitely need it, and others (SAT/SMT) should want it.

Important to have: Validation

termination competition consists of *two phases*:

1. *solvers* run on benchmarks, emit proof traces
2. *matcher* (postproc.) checks that trace matches benchmark
3. *validators* run on traces

(non)termination proof trace $\approx$ model, or unsat core.

automatic validation is highly recommended:

- advance formalized mathematics (validator source code is extracted from formal proof)
- discover bugs in solvers

We (termcomp) definitely need it, and others (SAT/SMT) should want it.

Important to have: Validation

termination competition consists of *two phases*:

1. *solvers* run on benchmarks, emit proof traces
2. *matcher* (postproc.) checks that trace matches benchmark
3. *validators* run on traces

(non)termination proof trace $\approx$ model, or unsat core.

automatic validation is highly recommended:

- advance formalized mathematics (validator source code is extracted from formal proof)
- discover bugs in solvers

We (termcomp) definitely need it, and others (SAT/SMT) should want it.

Important to have: detailed scoring

- for complexity categories, solvers answer YES (d_1, d_2) meaning $\Omega(n^{d_1}) \cap O(n^{d_2})$.
- Scoring for each benchmark depends on inclusion between answers of solvers.
- Scorer must see, for each benchmark, all solver's outputs.

How could this be realized?

Star-Exec's "post-processor" model extended:

- *individual* post-processor should see
  - (stdout separately from stderr)
  - also the original benchmark (to create or check the validation problem for the second stage)
- *bulk (display/scoring)* post-processor should see, per benchmark, the set of all (post-processed) solver outputs

Implementation:

- make Star-Exec open-source,
- we fork it, we implement the above (we already have it), you merge it back

Yes We Want This

already planned for Star-Exec, and we are looking forward to using it:

- stable and session/login-independent URLs for each data item:
  benchmark, solver, job (collection), job pair
- flexible query language, for the full data set.
  e.g., “the 10 smallest problems from category X that were unsolved in all previous competitions”, “all results where solver Y’s output contains the words Z”
- should offer queries everywhere (at each point in the GUI where some subset is selected)

And some more . . .

helpful for competition organizers, platform users (and their students):

- upload (and some checking) of new benchmarks (to be considered for future competitions)
- (controlled, random) selection of benchmarks for competitions
- import of legacy data (results of previous competitions), so it can be queried
- “on-the-fly” jobs: edit/upload a benchmark and run some solvers (cf. `http://rise4fun.com/z3`), store interesting (small and hard) submissions

Conclusion

- We (Termination) support the idea behind Star-Exec, and intend to use it.
- The current design does not fit all of the Termination Competition categories
  - second stage for validation,
  - scoring for complexity

  probably there are manual (or script-able) work-arounds
- We understand that resources (developer time) are limited, so . . . open-source it.