Tyrolean Complexity Tool: Features and Usage

Martin Avanzini Georg Moser Andreas Schnabl

Institute of Computer Science
University of Innsbruck, Austria

June 16, 2013

Tyrolean Complexity Tool TCT

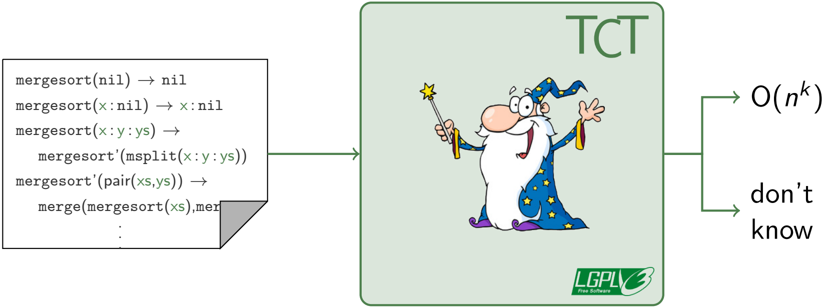
- (runtime) complexity analyser for term rewrite systems (TRSs)

  `http://cl-informatik.uibk.ac.at/software/tct`

Tyrolean Complexity Tool TCT

- (runtime) complexity analyser for term rewrite systems (TRSs)

http://cl-informatik.uibk.ac.at/software/tct



```
mergesort(nil) → nil
mergesort(x : nil) → x : nil
mergesort(x : y : ys) →
   mergesort'(msplit(x : y : ys))
mergesort'(pair(xs,ys)) →
   merge(mergesort(xs),mer
                 ⋮
```

$O(n^k)$

don't know

History

2008 **version 1.0** *extension to termination prover $\mathsf{T_TT_2}$*

- 3 dedicated complexity techniques

History

2008 **version 1.0** *extension to termination prover T_TT_2*

- 3 dedicated complexity techniques

2009 **version 1.5** *new implementation*

- in `Haskell`
- 9 methods implemented
- $\approx$ 3.400 lines of code

History

2008 **version 1.0** *extension to termination prover $\mathsf{T_TT_2}$*

- 3 dedicated complexity techniques

2009 **version 1.5** *new implementation*

- in `Haskell`
- 9 methods implemented
- ≈ 3.400 lines of code

2013 **version 2.0** *current version*

- 23 methods implemented
- ≈ 13.000 lines of code / 4.000 lines of comment

Interfaces

1. **web**

   `http://cl-informatik.uibk.ac.at/software/tct`

2. **command line**
   - automatic mode
   - customisable through search strategies

3. **interactive**
   - semi-automatic mode

Interfaces

1. **web**

   `http://cl-informatik.uibk.ac.at/software/tct`

2. **command line**
   - automatic mode
   - customisable through search strategies

3. **interactive**
   - semi-automatic mode

Interfaces

1. **web**

   `http://cl-informatik.uibk.ac.at/software/tct`

2. **command line**
   - automatic mode
   - customisable through search strategies

   demo

3. **interactive**
   - semi-automatic mode

Interfaces

1. **web**

   `http://cl-informatik.uibk.ac.at/software/tct`

2. **command line**
   - automatic mode
   - customisable through search strategies

3. **interactive**
   - semi-automatic mode

Interfaces

1. **web**

   `http://cl-informatik.uibk.ac.at/software/tct`

2. **command line**
   - automatic mode
   - customisable through search strategies

3. **interactive**
   - semi-automatic mode

Command Line Interface

1. runs on GNU/Linux

termcomp or tpdb format

```
$ tct [ options ] <file>
```

Command Line Interface

termcomp or tpdb format

1. runs on GNU/Linux

```
$ tct [ options , -s <search strategy> ] <file>
```

S-expression syntax

```
(<name> [:<argname> <arg>]* [<arg>]*)
```

- `matrix`
- `matrix :degree 2`
- `fastest (matrix :degree 2)`
  `(timeout 3 (bounds :enrichment match))`

Command Line Interface

1. runs on GNU/Linux

```
$ tct [ options , -s <search strategy> ] <file>
```

demo

Command Line Interface

1. runs on GNU/Linux

```
$ tct [ options , -s <search strategy> ] <file>
```

Command Line Interface

1. runs on GNU/Linux

```
$ tct [ options , -s <search strategy> ] <file>
```

2. configured in ~/.tct/tct.hs

```haskell
import Tct.Configuration
import Tct.Interactive
import Tct.Instances
import qualified Termlib.Repl as TR

main :: IO ()
main = tct config

config :: Config
config = defaultConfig
```

Customisation

```
import Tct.Configuration
import Tct.Interactive
import Tct.Instances
import qualified Termlib.Repl as TR

main = tct config

config = defaultConfig { strategies = strategies }
  where
    strategies =
      [ matrices ::: strategy "matrices" ( optional naturalArg "start" (Nat 1)
                                           :+: naturalArg )
      , withDP   ::: strategy "withDP" ]

matrices (Nat start :+: Nat n) =
  fastest [ matrix ‘withDimension‘ d ‘withBits‘ bitsForDimension d
          | d <- [start..start+n] ]
    where
      bitsForDimension d
        | d < 3 = 2
        | otherwise = 1
withDP = ...
```

Customisation

```
import Tct.Configuration
import Tct.Interactive
import Tct.Instances
i

m

c
  where
    strategies =
      [ matrices ::: strategy "matrices" ( optional naturalArg "start" (Nat 1)
                                           :+: naturalArg )
      , withDP   ::: strategy "withDP" ]
```

search strategy declaration

```
<code> ::: strategy "<name>" [<parameters-declaration>]
```

```
matrices (Nat start :+: Nat n) =
  fastest [ matrix ‘withDimension‘ d ‘withBits‘ bitsForDimension d
          | d <- [start..start+n] ]
    where
      bitsForDimension d
        | d < 3 = 2
        | otherwise = 1
withDP = ...
```

search strategy implementation

Proof Search Strategies

- **processors**
  - `matrix`
  - `poly`
  - `popstar`
  - ...
- **processor modifiers**
  - *<processor> ‘withDegree‘ <deg>*
  - *<processor> ‘withBits‘ <bits>*
  - ...
- **combinators**
  - `timeout` *<secs> <strategy>*
  - `best` *<strategy>* $\cdots$ *<strategy>*
  - `fastest` *<strategy>* $\cdots$ *<strategy>*
  - `ite` *<strategy> <strategy> <strategy>*
  - ...

Customisation

```
import Tct.Configuration
import Tct.Interactive
import Tct.Instances
import qualified Termlib.Repl as TR

main = tct config

config = defaultConfig { strategies = strategies }
  where
    strategies =
      [ matrices ::: strategy "matrices" ( optional naturalArg "start" (Nat 1)
                                           :+: naturalArg )
      , withDP   ::: strategy "withDP" ]

matrices (Nat start :+: Nat n) =
  fastest [ matrix `withDimension` d `withBits` bitsForDimension d
          | d <- [start..start+n] ]
    where
      bitsForDimension d
        | d < 3 = 2
        | otherwise = 1
withDP = ...
```

Transformations

- processors often generate sub-problems

$$\frac{\vdash \mathcal{P}_1 \colon f_1 \qquad \cdots \qquad \vdash \mathcal{P}_n \colon f_n}{\vdash \mathcal{P} \colon f}$$

Transformations

- processors often generate sub-problems

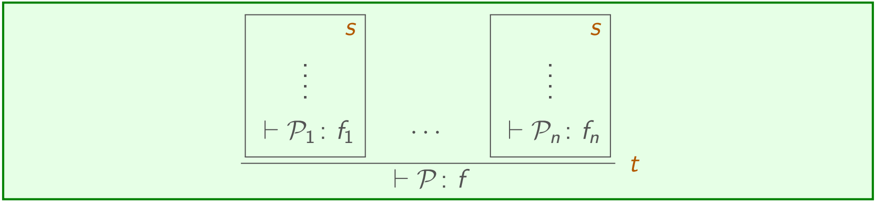
$$\frac{\vdash \mathcal{P}_1 \colon f_1 \quad \cdots \quad \vdash \mathcal{P}_n \colon f_n}{\vdash \mathcal{P} \colon f}$$

- implemented as transformations in TCT
  - `dependencyPairs` and `dependencyTuples`
  - `decompose` and `decomposeDG`
  - `pathAnalysis`
  - `weightGap`
  - …

Transformations

Lifting and Combinators

- lifting to strategies
  - $t \gg| \; s$ and $t \gg|| \; s$



Transformations

Lifting and Combinators

- lifting to strategies
  - $t \gg| \; s$ and $t \gg|| \; s$
- combinators
  - $t_1 \ggg t_2$

$$\dfrac{\dfrac{\vdash \mathcal{Q}_1 : g_1 \cdots \vdash \mathcal{Q}_k : g_k}{\vdash \mathcal{P}_1 : f_1}\; t_2 \quad \cdots \quad \dfrac{\vdash \mathcal{Q}_l : g_l \cdots \vdash \mathcal{Q}_m : g_m}{\vdash \mathcal{P}_n : f_n}\; t_2}{\vdash \mathcal{P} : f}\; t_1$$

Transformations

Lifting and Combinators

- lifting to strategies
  - $t \gg| \ s$ and $t \gg|| \ s$
- combinators
  - $t_1 \ggg t_2$
  - $t_1 <> t_2$ and $t_1 <|> t_2$

$$\frac{\vdash \mathcal{P}_1 : f_1 \quad \cdots \quad \vdash \mathcal{P}_m : f_m}{\vdash \mathcal{P} : f}\ t_1 \qquad \text{or} \qquad \frac{\vdash \mathcal{Q}_1 : g_1 \quad \cdots \quad \vdash \mathcal{Q}_n : g_n}{\vdash \mathcal{P} : f}\ t_2$$

Transformations

Lifting and Combinators

- lifting to strategies
  - $t \gg| \; s$ and $t \gg|| \; s$
- combinators
  - $t_1 \ggg t_2$
  - $t_1 <> t_2$ and $t_1 <|> t_2$
  - `try` t and `force` t

Transformations

Lifting and Combinators

- lifting to strategies
  - t ≫| s and t ≫|| s
- combinators
  - t_1 ⋙ t_2
  - t_1 <> t_2 and t_1 <|> t_2
  - `try` t and `force` t
  - `exhaustively` t `=` t ⋙ `try (exhaustively` t`)`

Transformations

Example

```
withDP =
  (dps <> dts)
  ⋙ try (exhaustively decomposeIndependent)
  ⋙ try cleanSuffix
  ⋙ try usableRules
  where
    dps = dependencyPairs ⋙ try usableRules ⋙ timeout 5 wgOnRules
    dts = dependencyTuples
    wgOnRules = weightgap ‘withDimension‘ 1 ‘wgOn‘ WgOnTrs
```

Transformations

Example

```
withDP =
  (dps <> dts)
    >>> try (exhaustively decomposeIndependent)
    >>> try cleanSuffix
    >>> try usableRules
    where
      dps = dependencyPairs >>> try usableRules >>> timeout 5 wgOnRules
      dts = dependencyTuples
      wgOnRules = weightgap `withDimension` 1 `wgOn` WgOnTrs
```

Transformations

Example

```
withDP =
  (dps <> dts)
    ⋙ try (exhaustively decomposeI
    ⋙ try cleanSuffix
    ⋙ try usableRules
    where
      dps = dependencyPairs ⋙ try usableRules ⋙ timeout 5 wgOnRules
      dts = dependencyTuples
      wgOnRules = weightgap ‘withDimension‘ 1 ‘wgOn‘ WgOnTrs
```

abort if timeout 5 wgOnRules fails

continue if usableRules fails

Transformations

Example

```
withDP =
  (dps <> dts)
  >>> try (exhaustively decomposeIndependent)
  >>> try cleanSuffix
  >>> try usableRules
  where
    dps = dependencyPairs >>> try usableRules >>> timeout 5 wgOnRules
    dts = dependencyTuples
    wgOnRules = weightgap ‘withDimension‘ 1 ‘wgOn‘ WgOnTrs
```

Transformations

Example

```
withDP =
  (dps <> dts)
  >>> try (exhaustively decomposeIndependent)
  >>> try cleanSuffix
  >>> try usableRules
  where
    dps = dependencyPairs >>> try usableRules >>> timeout 5 wgOnRules
    dts = dependencyTuples
    wgOnRules = weightgap ‘withDimension‘ 1 ‘wgOn‘ WgOnTrs
```

Interfaces

1. **web**

   `http://cl-informatik.uibk.ac.at/software/tct`

2. **command line**
   - automatic mode
   - customisable through search strategies

3. **interactive**
   - semi-automatic mode

Interactive Interface

- run by command `tct -i`

Interactive Interface

proof & list of open problems

- run by command `tct -i`
- `ghci` & TCT library & proof state

Interactive Interface

proof & list of open problems

- run by command `tct -i`
- `ghci` & TCT library & proof state

basic functionality

1. modify proof state
   - `load` *"<filename>"*
   - `apply` *method*
   - `select` *lst* and `unselect` *lst*

Interactive Interface

proof & list of open problems

- run by command `tct -i`
- `ghci` & TCT library & proof state

basic functionality

1. modify proof state
   - `load` *"<filename>"*
   - `apply` *method*
   - `select` *lst* and `unselect` *lst*
2. history
   - `undo`, `reset`

Interactive Interface

proof & list of open problems

- run by command `tct -i`
- `ghci` & TCT library & proof state

basic functionality

1. modify proof state
   - `load` *"<filename>"*
   - `apply` *method*
   - `select` *lst* and `unselect` *lst*
2. history
   - `undo`, `reset`
3. inspect proof state
   - `state` and `proof`
   - `problems`, `uargs`, `wdgs`, ...
   - `writeProof` *"<filename>"*

Interactive Interface

proof & list of open problems

- run by command `tct -i`
- `ghci` & TCT library & proof state

basic functionality

1. modify proof state
   - `load` *"<filename>"*
   - `apply` *method*
   - `select` *lst* and `unselect` *lst*
2. history
   - `undo`, `reset`
3. inspect proof state
   - `state` and `proof`
   - `problems`, `uargs`, `wdgs`, ...
   - `writeProof` *"<filename>"*

demo

Interactive Interface

proof & list of open problems

- run by command `tct -i`
- `ghci` & TCT library & proof state

basic functionality

1. modify proof state
   - `load` *"<filename>"*
   - `apply` *method*
   - `select` *lst* and `unselect` *lst*
2. history
   - `undo`, `reset`
3. inspect proof state
   - `state` and `proof`
   - `problems`, `uargs`, `wdgs`, ...
   - `writeProof` *"<filename>"*

Conclusion

TCT is a complexity analyser for TRSs

- open source
- implements majority of techniques known for polynomial complexity analysis
- automatic & interactive mode