MUSer2: An Efficient MUS Extractor

SYSTEM DESCRIPTION

Anton Belov and Joao Marques-Silva

Complex and Adaptive Systems Laboratory
University College Dublin, Ireland

PoS 2012
June 16, 2012
Trento, Italy

Introduction

Minimal Unsatisfiability

- $\mathcal{F}$ is *minimally unsatisfiable* ($\mathcal{F} \in \mathrm{MU}$), if $\mathcal{F} \in \mathrm{UNSAT}$ and for any $C \in \mathcal{F}$, $\mathcal{F} \setminus \{C\} \in \mathrm{SAT}$.

Introduction

Minimal Unsatisfiability

- $\mathcal{F}$ is *minimally unsatisfiable* ($\mathcal{F} \in \mathrm{MU}$), if $\mathcal{F} \in \mathrm{UNSAT}$ and for any $C \in \mathcal{F}$, $\mathcal{F} \setminus \{C\} \in \mathrm{SAT}$.
- $\mathcal{F}'$ is *minimally unsatisfiable subformula (MUS)* of $\mathcal{F}$ ($\mathcal{F}' \in \mathrm{MUS}(\mathcal{F})$) if $\mathcal{F}' \subseteq \mathcal{F}$ and $\mathcal{F}' \in \mathrm{MU}$.

Introduction

Minimal Unsatisfiability

- $\mathcal{F}$ is *minimally unsatisfiable* ($\mathcal{F} \in \text{MU}$), if $\mathcal{F} \in \text{UNSAT}$ and for any $C \in \mathcal{F}$, $\mathcal{F} \setminus \{C\} \in \text{SAT}$.
- $\mathcal{F}'$ is *minimally unsatisfiable subformula (MUS)* of $\mathcal{F}$ ($\mathcal{F}' \in \text{MUS}(\mathcal{F})$) if $\mathcal{F}' \subseteq \mathcal{F}$ and $\mathcal{F}' \in \text{MU}$.

Example

$$C_1 = x \vee y \qquad C_3 = x \vee \neg y$$

$$C_2 = \neg x \vee y \qquad C_4 = \neg x \vee \neg y$$

- $\{C_1, C_2, C_3, C_4\} \in \text{MU}$.

Introduction

Minimal Unsatisfiability

- $\mathcal{F}$ is *minimally unsatisfiable* ($\mathcal{F} \in \text{MU}$), if $\mathcal{F} \in \text{UNSAT}$ and for any $C \in \mathcal{F}$, $\mathcal{F} \setminus \{C\} \in \text{SAT}$.
- $\mathcal{F}'$ is *minimally unsatisfiable subformula (MUS)* of $\mathcal{F}$ ($\mathcal{F}' \in \text{MUS}(\mathcal{F})$) if $\mathcal{F}' \subseteq \mathcal{F}$ and $\mathcal{F}' \in \text{MU}$.

Example

$$C_1 = x \vee y \qquad C_3 = x \vee \neg y \qquad C_5 = y \vee z$$

$$C_2 = \neg x \vee y \qquad C_4 = \neg x \vee \neg y \qquad C_6 = y \vee \neg z$$

- $\{C_1, C_2, C_3, C_4\} \in \text{MU}$.
- $\mathcal{F} = \{C_1, \ldots, C_6\} \in \text{UNSAT}$, but $\notin \text{MU}$.

Introduction

Minimal Unsatisfiability

- $\mathcal{F}$ is *minimally unsatisfiable* ($\mathcal{F} \in \text{MU}$), if $\mathcal{F} \in \text{UNSAT}$ and for any $C \in \mathcal{F}$, $\mathcal{F} \setminus \{C\} \in \text{SAT}$.
- $\mathcal{F}'$ is *minimally unsatisfiable subformula (MUS)* of $\mathcal{F}$ ($\mathcal{F}' \in \text{MUS}(\mathcal{F})$) if $\mathcal{F}' \subseteq \mathcal{F}$ and $\mathcal{F}' \in \text{MU}$.

Example

$$C_1 = x \vee y \qquad C_3 = x \vee \neg y \qquad C_5 = y \vee z$$

$$C_2 = \neg x \vee y \qquad C_4 = \neg x \vee \neg y \qquad C_6 = y \vee \neg z$$

- $\{C_1, C_2, C_3, C_4\} \in \text{MU}$.
- $\{C_1, C_2, C_3, C_4\} \in \text{MUS}(\mathcal{F})$.

Introduction

Minimal Unsatisfiability

- $\mathcal{F}$ is *minimally unsatisfiable* ($\mathcal{F} \in \text{MU}$), if $\mathcal{F} \in \text{UNSAT}$ and for any $C \in \mathcal{F}$, $\mathcal{F} \setminus \{C\} \in \text{SAT}$.
- $\mathcal{F}'$ is *minimally unsatisfiable subformula (MUS)* of $\mathcal{F}$ ($\mathcal{F}' \in \text{MUS}(\mathcal{F})$) if $\mathcal{F}' \subseteq \mathcal{F}$ and $\mathcal{F}' \in \text{MU}$.

Example

$C_1 = x \vee y$

$C_2 = \neg x \vee y$

$C_3 = x \vee \neg y$

$C_4 = \neg x \vee \neg y$

$C_5 = y \vee z$

$C_6 = y \vee \neg z$

- $\{C_1, C_2, C_3, C_4\} \in \text{MU}$.
- $\{C_3, C_4, C_5, C_6\} \in \text{MUS}(\mathcal{F})$.

Introduction

Minimal Unsatisfiability

- $\mathcal{F}$ is *minimally unsatisfiable* ($\mathcal{F} \in \text{MU}$), if $\mathcal{F} \in \text{UNSAT}$ and for any $C \in \mathcal{F}$, $\mathcal{F} \setminus \{C\} \in \text{SAT}$.
- $\mathcal{F}'$ is *minimally unsatisfiable subformula (MUS)* of $\mathcal{F}$ ($\mathcal{F}' \in \text{MUS}(\mathcal{F})$) if $\mathcal{F}' \subseteq \mathcal{F}$ and $\mathcal{F}' \in \text{MU}$.

Applications of MUSes

- Early 2000's: type debugging in programming languages; circuit error diagnosis; error localization in automotive product configuration data.
- More recently: model checking (proof-based abstraction refinement); formal equivalence checking; logic synthesis.

Computation of MUSes

- Based on detection of *necessary* (or, *transition*) clauses
  - $C \in \mathcal{F}$ is *necessary* for $\mathcal{F}$ if $\mathcal{F} \in \text{UNSAT}$ and $\mathcal{F} \setminus \{C\} \in \text{SAT}$.
  - The set of all necessary clauses of $\mathcal{F}$ is precisely $\bigcap \text{MUS}(\mathcal{F})$.
  - $\mathcal{F} \in \text{MU}$ if and only if every $C \in \mathcal{F}$ is necessary for $\mathcal{F}$.
  - If C is necessary for $\mathcal{F}$, C is necessary for any UNSAT subset of $\mathcal{F}$.
- Iterative calls to SAT solver. Main approaches:
  - Deletion-based: necessary clauses are detected on transition from UNSAT to SAT. Unnecessary clauses are removed from the formula. Maintain over-approximation of an MUS.
  - Insertion-based: necessary clauses are detected on transition from SAT to UNSAT. Maintain under-approximation of an MUS.
  - Dichotomic: binary search.
- SAT solving is the main bottleneck of the computation, hence reduction in the number of SAT solver calls, and making SAT solver calls easier is the key to efficiency.

MUSer2 features

- Algorithms:
  - Hybrid algorithm (default): deletion-based, but builds MUSes bottom-up.
  - Insertion-based (`-ins`)
  - Dichotomic (`-dich`)
- Optimizations:
  - Clause-set refinement (default) and trimming (`[-trim|-tfp|-tpcrt]`)
  - Recursive model rotation (default)
  - (Adaptive) redundancy removal (`[-rr|-rra]`)
- Control/heuristics for clause ordering (`-order`)
- Testing of computed MUSes (`-test`)
- SAT solvers are used in a black-box manner; can use various SAT solvers (`-minisat|-picosat`)
- Software eng.: `C++11`, designed for extensibility/experimentation.
- Licensing: source – GPLv3; binaries (incl. extra/experimental features) – free for academic use.

Hybrid MUS Extraction [Marques-Silva&Lynce'11] w/o optimizations

Input : Unsatisfiable CNF Formula $\mathcal{F}$
Output: $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

$\mathcal{F}' \leftarrow \mathcal{F}$ // Working CNF formula
$\mathcal{M} \leftarrow \emptyset$ // MUS under-approximation
while $\mathcal{F}' \neq \emptyset$ **do** // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$
  $C \leftarrow \mathtt{PickClause}(\mathcal{F}')$
  $\mathrm{st} = \mathtt{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$ // Redundancy removal
  if st $=$ true **then** // If SAT, C is necessary for $\mathcal{M} \cup \mathcal{F}'$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
    $\mathrm{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$ // Recursive model rotation
  else
    $\mathcal{F}' \leftarrow \mathcal{F}' \setminus \{C\}$ // Clause-set refinement
return $\mathcal{M}$ // $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

- `MUSer2` options: default; `[-ins|-dich]` to change.

Optimizations: clause-set refinement/trimming

- **Fact:** Every unsatisfiable formula contains at least one MUS.
- Hence, if $\mathcal{U}$ is an unsatisfiable core of $\mathcal{F}$, all clauses outside of $\mathcal{U}$ can be removed from $\mathcal{F}$.
- Relies on the capability of SAT solvers to return unsatisfiable core.
- Effect: remove multiple unnecessary clauses at once.
- Applied to the working formula inside the main loop (e.g. $\mathcal{M} \cup \mathcal{F}'$ in the Hybrid algorithm) — *clause-set refinement*. Default in `MUSer2`.
- Applied to the input formula prior to MUS extraction — *clause-set trimming*.
  - Until fix point: `MUSer2` option `-tfp`
  - A fixed number of times: `MUSer2` option `-trim N`
  - Until size change is bounded: `MUSer2` option `-tpcrt P`

Hybrid MUS Extraction [Marques-Silva&Lynce'11]: clause-set refinement

Input : Unsatisfiable CNF Formula $\mathcal{F}$
Output: $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

$\mathcal{F}' \leftarrow \mathcal{F}$ // Working CNF formula
$\mathcal{M} \leftarrow \emptyset$ // MUS under-approximation
while $\mathcal{F}' \neq \emptyset$ **do** // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$
  $C \leftarrow \mathtt{PickClause}(\mathcal{F}')$
  $\mathrm{st} = \mathtt{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$ // Redundancy removal
  if st $=$ true **then** // If SAT, C is necessary for $\mathcal{M} \cup \mathcal{F}'$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
    $\mathrm{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$ // Recursive model rotation
  else
    $\mathcal{F}' \leftarrow \mathcal{F}' \setminus \{C\}$ // Clause-set refinement
return $\mathcal{M}$ // $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

Hybrid MUS Extraction [Marques-Silva&Lynce'11]: clause-set refinement

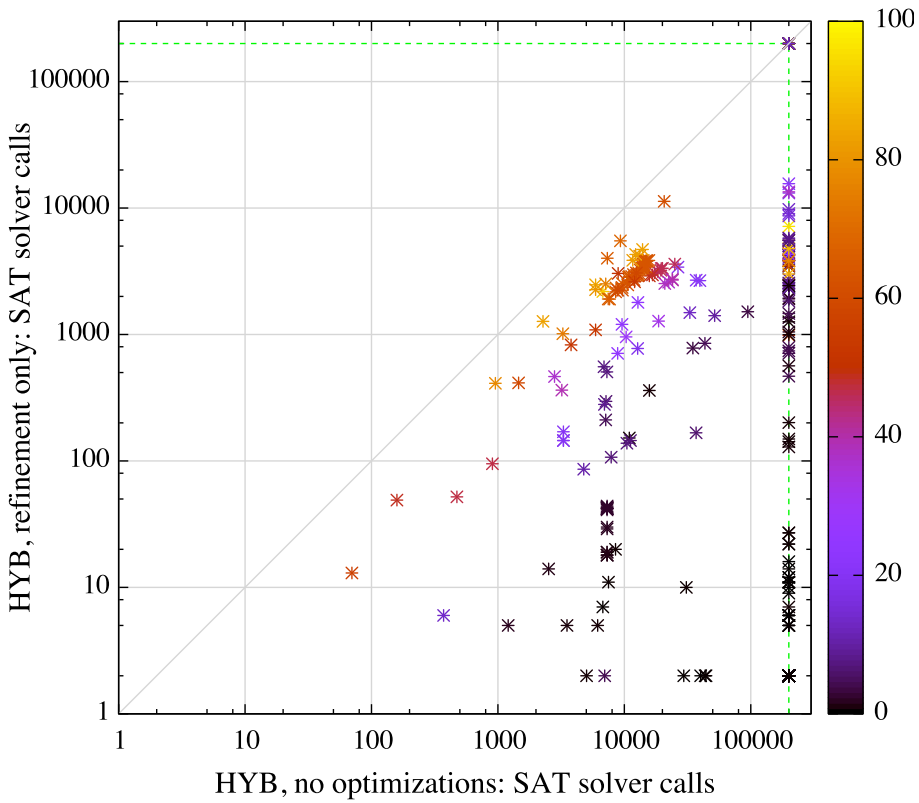
Input : Unsatisfiable CNF Formula $\mathcal{F}$
Output: $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

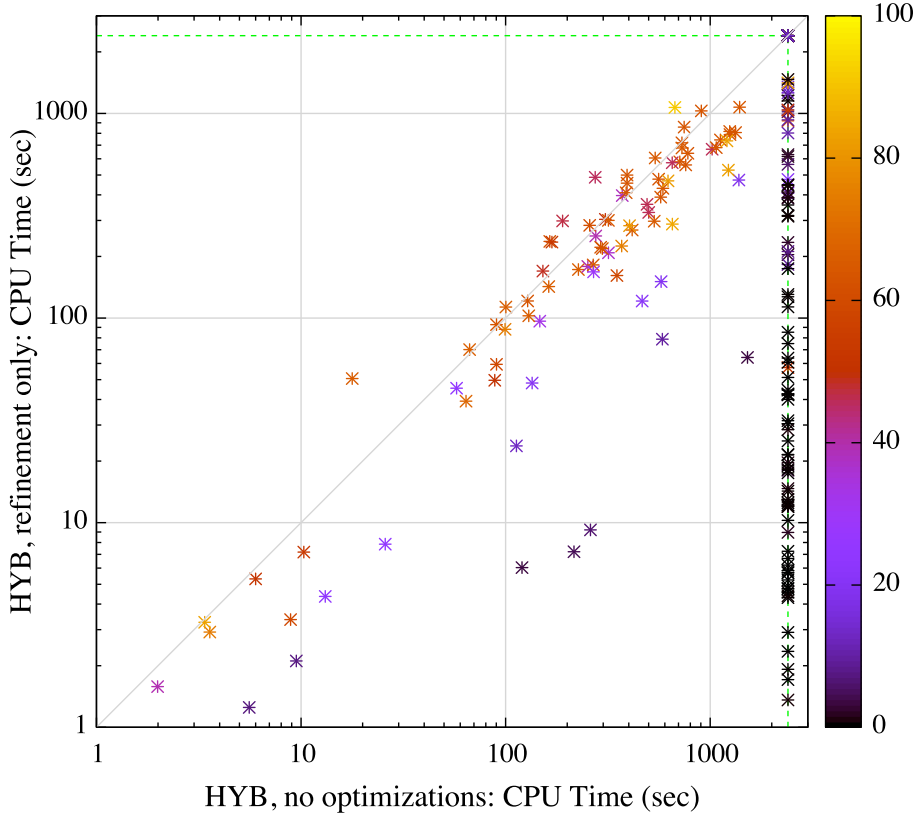
$\mathcal{F}' \leftarrow \mathcal{F}$ // Working CNF formula
$\mathcal{M} \leftarrow \emptyset$ // MUS under-approximation
while $\mathcal{F}' \neq \emptyset$ **do** // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$
  $C \leftarrow \mathtt{PickClause}(\mathcal{F}')$
  $(\mathrm{st}, \mathcal{U}) = \mathtt{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$ // Redundancy removal
  if st $=$ true **then** // If SAT, C is necessary for $\mathcal{M} \cup \mathcal{F}'$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
    $\mathrm{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$ // Recursive model rotation
  else
    $\mathcal{F}' \leftarrow \mathcal{U} \setminus \mathcal{M}$ // Clause-set refinement
return $\mathcal{M}$ // $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

- `MUSer2` options: default; `-norf` to disable.

Impact of clause-set refinement

- 295 benchmarks from track of SAT Competition 2011.
- Time limit 1800 sec, memory limit 4 GB.





- HYB, no optimizations (#sol=132) vs refinement only (#sol=221)
  - Left: number of SAT solver calls. Right: CPU time (sec).
  - Color: MUS size (% of input size).

Optimizations: recursive model rotation (RMR)

- **Fact:** C is necessary for $\mathcal{F}$ iff $\mathcal{F} \in$ UNSAT and $\exists \tau$ such that $\mathit{Unsat}(\mathcal{F}, \tau) = \{C\}$. τ is a *witness* (of necessity) for C.
  - During (hybrid) MUS extraction: when $M \cup (\mathcal{F}' \setminus \{C\}) \in$ SAT, the assignment τ found by the SAT solver is a witness for C.
  - Witnesses are also available in other algorithms for MUS extraction.
- *Model rotation* [Marques-Silva&Lynce'11]: given a witness τ for C, try to modify it into a witness τ' for another clause C': take $x \in \mathit{Var}(C)$, let $\tau' = \tau|_{\neg x}$, if $\mathit{Unsat}(\mathcal{F}, \tau') = \{C'\}$, then C' is necessary; continue with C' and τ'.
- *Recursive model rotation* [Belov&Marques-Silva'11]: for each necessary clause explore all possible flips (recursively).
- Effect: detect multiple necessary clauses in a single SAT solver call.
- Default in `MUSer2`.

Hybrid MUS Extraction [Marques-Silva&Lynce'11]: RMR

Input : Unsatisfiable CNF Formula $\mathcal{F}$
Output: $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

$\mathcal{F}' \leftarrow \mathcal{F}$ // Working CNF formula
$\mathcal{M} \leftarrow \emptyset$ // MUS under-approximation
while $\mathcal{F}' \neq \emptyset$ **do** // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$
  $C \leftarrow \mathtt{PickClause}(\mathcal{F}')$
  $(\mathrm{st}, \mathcal{U}) = \mathtt{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$ // Redundancy removal
  if st $=$ true **then** // If SAT, C is necessary for $\mathcal{M} \cup \mathcal{F}'$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
    $\mathrm{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$ // Recursive model rotation
  else
    $\mathcal{F}' \leftarrow \mathcal{U} \setminus \mathcal{M}$ // Clause-set refinement
return $\mathcal{M}$ // $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

- MUSer2 options: default; `-norot` to disable.

Hybrid MUS Extraction [Marques-Silva&Lynce'11]: RMR

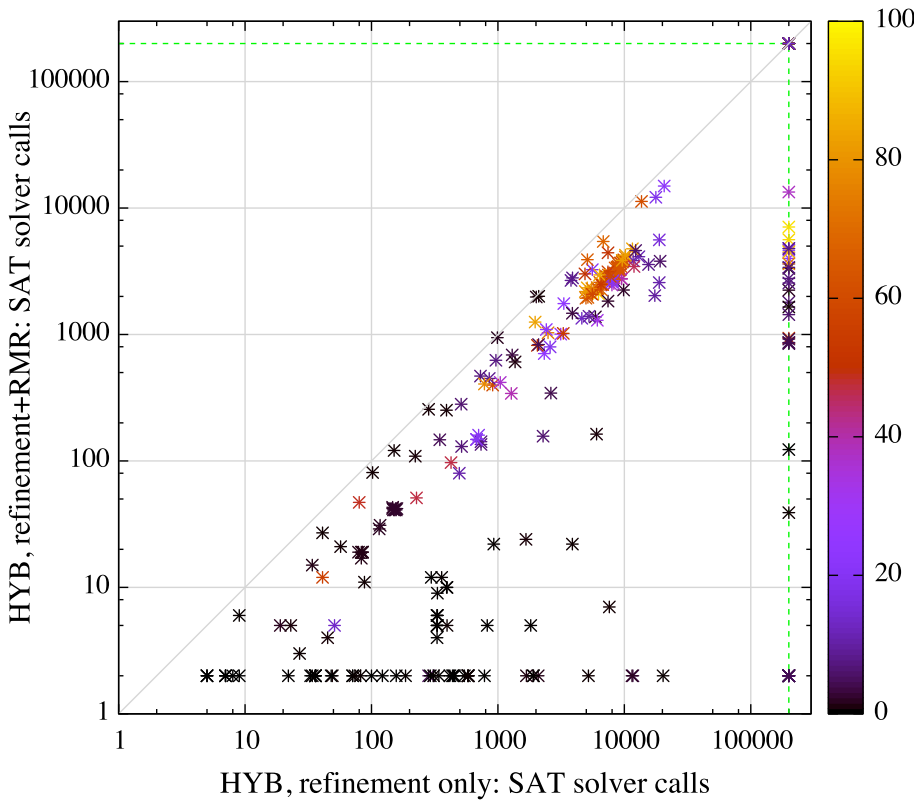
Input : Unsatisfiable CNF Formula $\mathcal{F}$
Output: $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

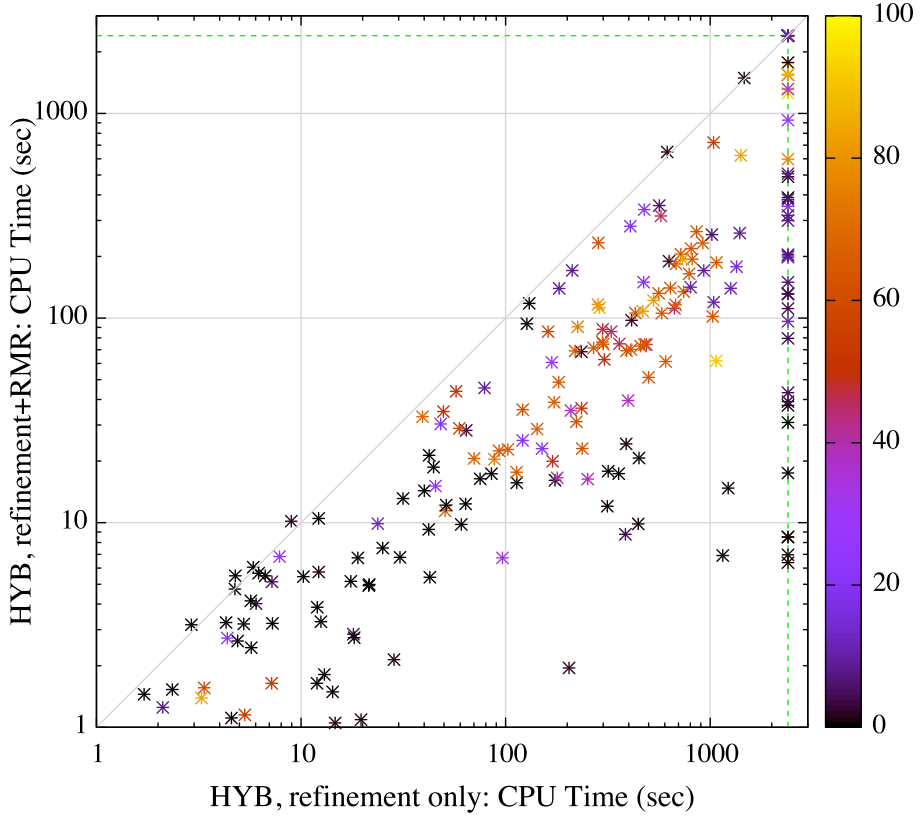
$\mathcal{F}' \leftarrow \mathcal{F}$ // Working CNF formula
$\mathcal{M} \leftarrow \emptyset$ // MUS under-approximation
while $\mathcal{F}' \neq \emptyset$ **do** // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$
  $C \leftarrow \mathtt{PickClause}(\mathcal{F}')$
  $(\mathrm{st}, \mathcal{U}, \tau) = \mathtt{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$ // Redundancy removal
  if st = true **then** // If SAT, C is necessary for $\mathcal{M} \cup \mathcal{F}'$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
    $\mathtt{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$ // Recursive model rotation
  else
    $\mathcal{F}' \leftarrow \mathcal{U} \setminus \mathcal{M}$ // Clause-set refinement
return $\mathcal{M}$ // $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

- MUSer2 options: default; -norot to disable.

Impact of recursive model rotation

- 295 benchmarks from track of SAT Competition 2011.
- Time limit 1800 sec, memory limit 4 GB.





- HYB, refinement only (#sol=221) vs refinement+RMR (#sol=254)
  - Left: number of SAT solver calls. Right: CPU time (sec).
  - Color: MUS size (% of input size).

Optimizations: redundancy removal

- **Fact:** If $\mathcal{F} \in \text{UNSAT}$, then $\mathcal{F} \setminus \{C\} \equiv \mathcal{F} \setminus \{C\} \cup \{\neg C\}$
  - $\{\neg C\}$ stands for $\bigcup_{l \in C} \neg l$.
  - During (hybrid) MUS extraction: add $\{\neg C\}$ to the formula before SAT solver call [Marques-Silva&Lynce'11].
  - Can also be done for other algorithms [v.Maaren&Wieringa'08].
- Effect: make SAT calls easier.
- But: if $\mathcal{F} \setminus \{C\} \cup \{\neg C\} \in \text{UNSAT}$ and any of the literals from $\{\neg C\}$ are in the unsatisfiable core $\mathcal{U}$, the core cannot be safely used for refinement ($\mathcal{F} \cap \mathcal{U}$ may be SAT).
- Adaptive approach: if a core is "tainted", disable redundancy removal until the next SAT outcome.
- `MUSer2` options: `-rr|-rra`

Hybrid MUS Extraction [Marques-Silva&Lynce'11]: redundancy removal

Input : Unsatisfiable CNF Formula $\mathcal{F}$
Output: $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

$\mathcal{F}' \leftarrow \mathcal{F}$ // Working CNF formula
$\mathcal{M} \leftarrow \emptyset$ // MUS under-approximation
while $\mathcal{F}' \neq \emptyset$ **do** // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$
  $C \leftarrow \mathtt{PickClause}(\mathcal{F}')$
  $(\mathrm{st}, \mathcal{U}, \tau) = \mathtt{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}))$ // Redundancy removal
  if st = true **then** // If SAT, C is necessary for $\mathcal{M} \cup \mathcal{F}'$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
    $\mathtt{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$ // Recursive model rotation
  else
    $\mathcal{F}' \leftarrow \mathcal{U} \setminus \mathcal{M}$ // Clause-set refinement
return $\mathcal{M}$ // $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

- MUSer2 options: `-rr`, `-rra` for adaptive.

Hybrid MUS Extraction [Marques-Silva&Lynce'11]: redundancy removal

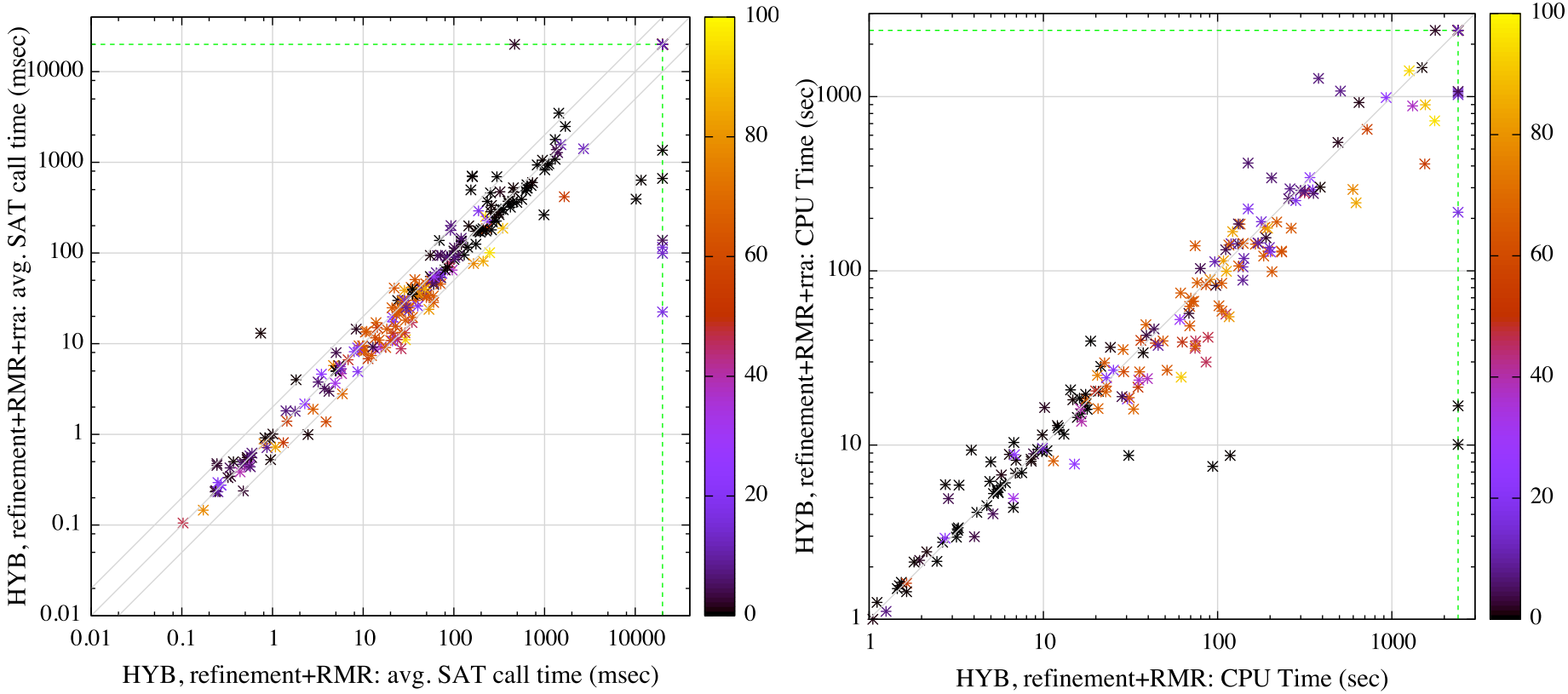
Input : Unsatisfiable CNF Formula $\mathcal{F}$
Output: $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

$\mathcal{F}' \leftarrow \mathcal{F}$ // Working CNF formula
$\mathcal{M} \leftarrow \emptyset$ // MUS under-approximation
while $\mathcal{F}' \neq \emptyset$ **do** // Inv: $\mathcal{M} \subseteq \mathcal{F}$, and $\forall C \in \mathcal{M}$ is nec. for $\mathcal{M} \cup \mathcal{F}'$
  $C \leftarrow \mathtt{PickClause}(\mathcal{F}')$
  $(\mathrm{st}, \tau, \mathcal{U}) = \mathtt{SAT}(\mathcal{M} \cup (\mathcal{F}' \setminus \{C\}) \cup \{\neg C\})$ // Redundancy removal
  if st = true **then** // If SAT, C is necessary for $\mathcal{M} \cup \mathcal{F}'$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{C\}$
    $\mathtt{RMR}(\mathcal{F}' \cup \mathcal{M}, \mathcal{M}, \tau)$ // Recursive model rotation
  else if $\mathcal{U} \cap \{\neg C\} = \emptyset$ **then** // If the core is ''clean''
    $\mathcal{F}' \leftarrow \mathcal{U} \setminus \mathcal{M}$ // Clause-set refinement
return $\mathcal{M}$ // $\mathcal{M} \in \mathrm{MUS}(\mathcal{F})$

- MUSer2 options: `-rr`, `-rra` for adaptive.

Impact of (adaptive) redundancy removal

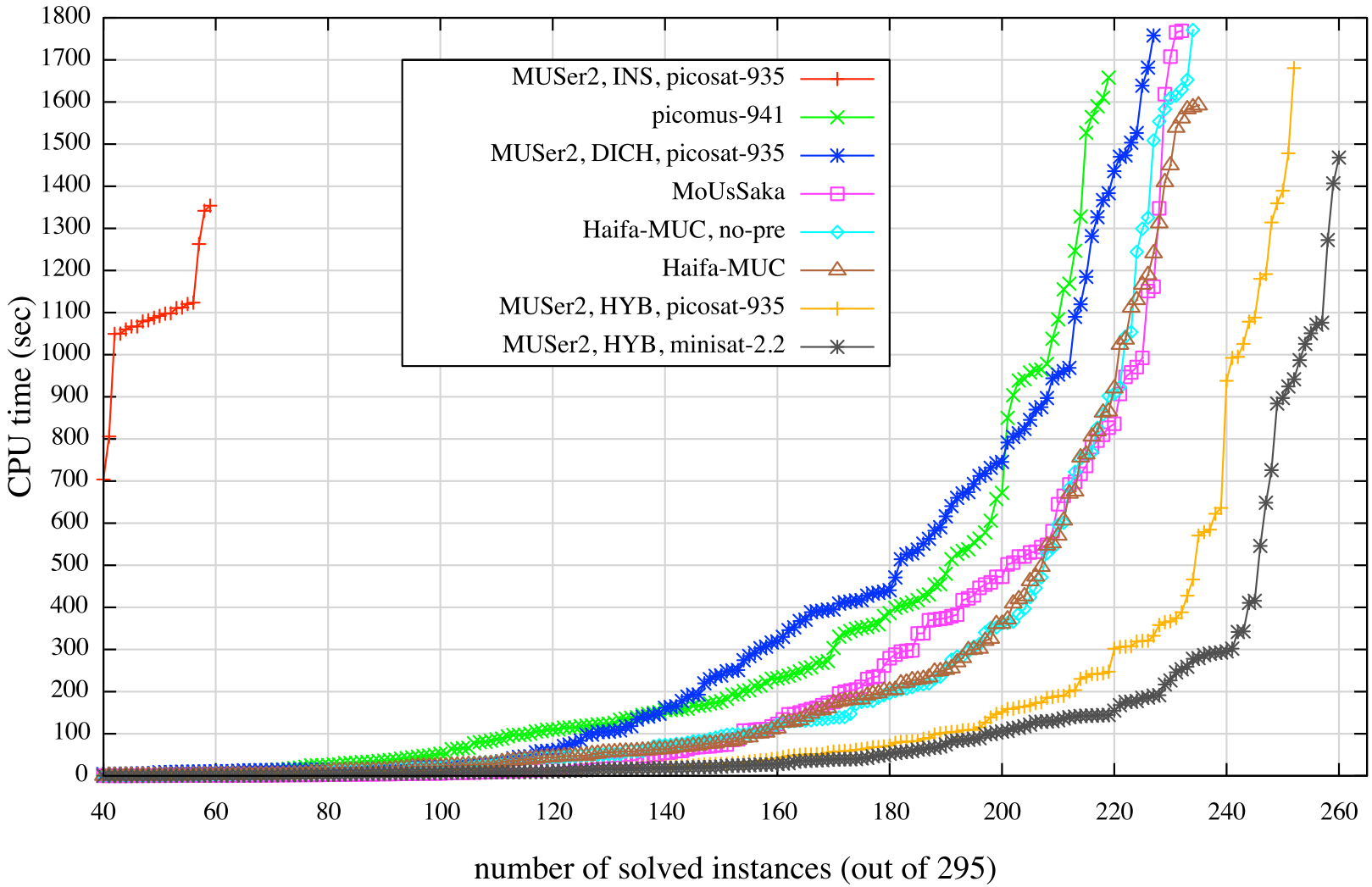
- 295 benchmarks from track of SAT Competition 2011.
- Time limit 1800 sec, memory limit 4 GB.



- HYB, refinement+RMR (#sol=254) vs ref+RMR+rra (#sol=260)
  - Left: avg. time per SAT call (msec). Right: CPU time (sec).
  - Color: MUS size (% of input size).

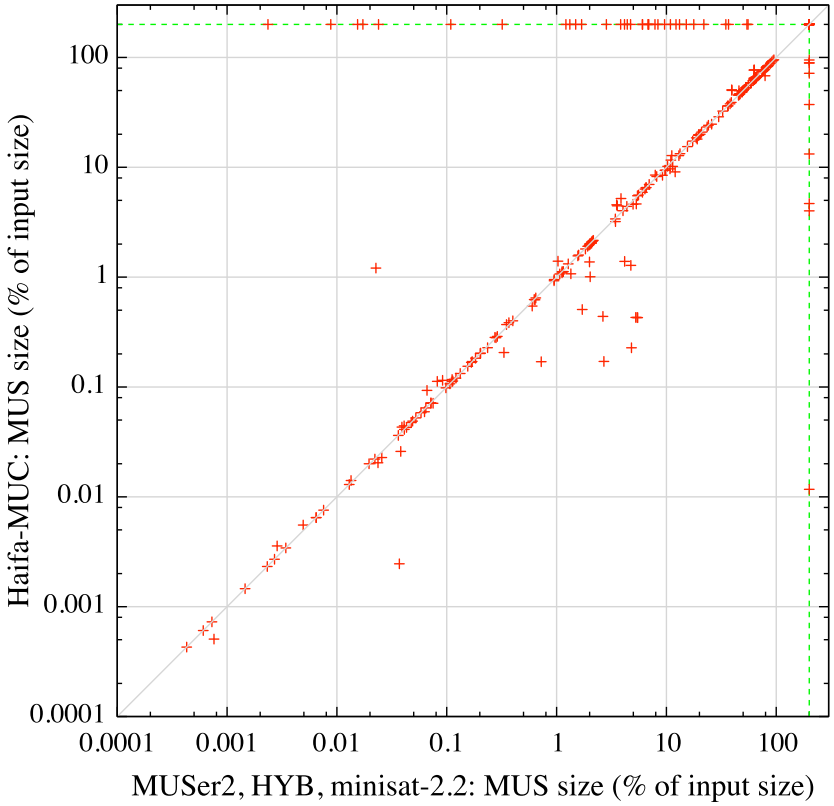
Performance comparison: run-time

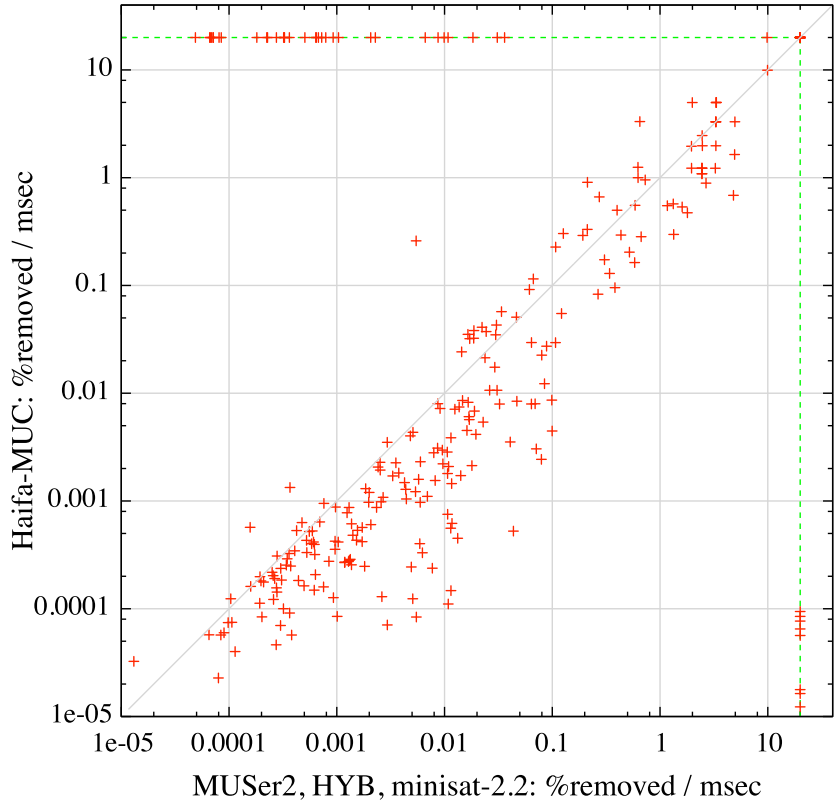
- 295 benchmarks used in the MUS track of SAT Competition 2011.
- Time limit 1800 sec, memory limit 4 GB.



Performance comparison: MUS size and velocity

- 295 benchmarks from track of SAT Competition 2011.
- Time limit 1800 sec, memory limit 4 GB.





- `MUSer2` (#sol=260) vs `Haifa-MUC` (#sol=235)
  - Left: MUS size (% of input size). Right: velocity (% removed/msec).
  - Note: the same order.

Summary

- `MUSer2` — state-of-the-art, open source MUS extractor.
- Also knows to compute group-MUSes.
  - All optimizations described in this talk (with the exception of redundancy removal) are implemented for group-MUSes.
- Single source for all the theory: AI Comm. 2012 [Belov,Lynce&Marques-Silva'12]
- Binary version: irredundant subformulas [Belov,Janota,Lynce&Marques-Silva'12], variable-MUSes [Belov,Ivrii,Matsliah&Marques-Silva'12], heuristics, and more.
- TODOs: redundancy removal for group-MUSes/insertion/dichotomic algorithms; wrappers for other SAT solvers.
- Download at `http://logos.ucd.ie/wiki/doku.php?id=muser`

Summary

- `MUSer2` — state-of-the-art, open source MUS extractor.
- Also knows to compute group-MUSes.
  - All optimizations described in this talk (with the exception of redundancy removal) are implemented for group-MUSes.
- Single source for all the theory: AI Comm. 2012 [Belov,Lynce&Marques-Silva'12]
- Binary version: irredundant subformulas [Belov,Janota,Lynce&Marques-Silva'12], variable-MUSes [Belov,Ivrii,Matsliah&Marques-Silva'12], heuristics, and more.
- TODOs: redundancy removal for group-MUSes/insertion/dichotomic algorithms; wrappers for other SAT solvers.
- Download at `http://logos.ucd.ie/wiki/doku.php?id=muser`

Thank you for your attention !