On Lazy and Eager Interactive Reconfiguration

Mikoláš Janota\textsuperscript{1}
Goetz Botterweck\textsuperscript{2} Joao Marques-Silva\textsuperscript{1,3}

\textsuperscript{1}INESC-ID/IST, Lisbon, Portugal
\textsuperscript{2}Lero, Limerick, Ireland
\textsuperscript{3}CASL/CSI, University College Dublin, Ireland

VAMOS 2014, January 22–24
Nice, France
Reconfiguration Example

```
Laptop

Docking Station

Extra Power Adapter

Display

Graphics Accelerator

14"

15"

17"
```

Janota et al.

On Lazy and Eager Interactive Reconfiguration
Reconfiguration Example

Laptop

Docking Station

Extra Power Adapter

Display

Graphics Accelerator

14"

15"

17"

Janota et al.
On Lazy and Eager Interactive Reconfiguration
Reconfiguration Example

- Laptop
- Docking Station
- Extra Power Adapter
- Display
- Graphics Accelerator

14" ✔
15" ✗
17" ✗

requires
excludes
Reconfiguration Example

Laptop

Docking Station
Extra Power Adapter
Display
Graphics Accelerator

14"
15"
17"

requires
excludes
Scenarios in Reconfiguration

- **LEGAL**: can be changed without further changes
- **RECONFIGURE**: can be changed but some other default decisions need to be changed.
- **ILLEGAL**: cannot be changed without violating the formula or without altering other user decisions.
Scenarios in Reconfiguration

- **LEGAL**: can be changed without further changes
- **RECONFIGURE**: can be changed but some other default decisions need to be changed.
Scenarios in Reconfiguration

- **LEGAL**: can be changed without further changes
- **RECONFIGURE**: can be changed but some other default decisions need to be changed.
- **ILLEGAL**: cannot be changed without violating the formula or without altering other user decisions.
Some Concepts from (Propositional) Logic

- CNF: express an instance \textit{clauses}

\{\
\neg x \lor z,\
\neg y \lor z,\
\neg x \lor \neg y\
\}
Some Concepts from (Propositional) Logic

- CNF: express an instance *clauses*
  
  - E.g.
    \[
    \{\neg x \lor z, \neg y \lor z, \neg x \lor \neg y\} \]

- A CNF is *satisfiable* if there's a truth assignment that makes all the clauses true. It is *unsatisfiable* otherwise. We can use a SAT solver to decide formula's satisfiability.

- A Minimally Unsatisfiable Set (MUS) of clauses is a irreducible unsatisfiable CNF.

- A Minimal Correction Subset (MCS) of clauses is a irreducible set of clauses whose removal makes the original formula satisfiable.
Some Concepts from (Propositional) Logic

- CNF: express an instance *clauses*
  - E.g.
    \[
    \{ (\neg x \lor z), (\neg y \lor z), (\neg x \lor \neg y) \}\n    \]

- A CNF is *satisfiable* if there’s a truth assignment that makes all the clauses true. It is *unsatisfiable* otherwise. We can use a SAT solver to decide formula’s satisfiability.
Some Concepts from (Propositional) Logic

- CNF: express an instance \textit{clauses}
  - E.g.
    \[
    \{ (\neg x \lor z), (\neg y \lor z), (\neg x \lor \neg y) \}
    \]

- A CNF is \textit{satisfiable} if there’s a truth assignment that makes all the clauses true. It is \textit{unsatisfiable} otherwise. We can use a \textit{SAT solver} to decide formula’s satisfiability.

- A \textit{Minimally Unsatisfiable Set (MUS)} of clauses is a irreducible unsatisfiable CNF.
Some Concepts from (Propositional) Logic

- CNF: express an instance *clauses*
  - E.g.
    \[ \{ (\neg x \lor z), (\neg y \lor z), (\neg x \lor \neg y) \} \]

- A CNF is *satisfiable* if there’s a truth assignment that makes all the clauses true. It is *unsatisfiable* otherwise. We can use a SAT solver to decide formula’s satisfiability.

- A *Minimally Unsatisfiable Set (MUS)* of clauses is a irreducible unsatisfiable CNF.

- A *Minimal Correction Subset (MCS)* of clauses is a irreducible set of clauses whose removal makes the original formula satisfiable.
Handling Reconfiguration Scenarios

Modeling the problem.

- $\mathcal{F}$ . . . configuration instance
Handling Reconfiguration Scenarios

Modeling the problem.

- \( \mathcal{F} \) ... configuration instance
- \( \mathcal{D}^a \) ... decisions made by the agent
Handling Reconfiguration Scenarios

Modeling the problem.

- $\mathcal{F}$ . . . configuration instance
- $\mathcal{D}^a$ . . . decisions made by the agent
  - E.g. $\{\neg x, y\}$

LEGAL: Just update $\mathcal{D}^a$ and $\mathcal{D}^d$.
RECONFIGURE: Change $\mathcal{D}^d$ so that the new decision can be accommodated. Use MCSes for that.
ILLEGAL: Provide an explanation why.
Handling Reconfiguration Scenarios

Modeling the problem.

- $\mathcal{F}$ ... configuration instance
- $\mathcal{D}^a$ ... decisions made by the agent
  - E.g. $\{\neg x, y\}$
- $\mathcal{D}^d$ ... default decisions.
Handling Reconfiguration Scenarios

Modeling the problem.

- $\mathcal{F}$ ... configuration instance
- $\mathcal{D}^a$ ... decisions made by the agent
  - E.g. $\{\neg x, y\}$
- $\mathcal{D}^d$ ... default decisions.

Changing from decision $l$ to $\neg l$

- LEGAL: Just update $\mathcal{D}^a$ and $\mathcal{D}^d$
Handling Reconfiguration Scenarios

Modeling the problem.

- \( \mathcal{F} \) ... configuration instance
- \( \mathcal{D}^a \) ... decisions made by the agent
  - E.g. \( \{\neg x, y\} \)
- \( \mathcal{D}^d \) ... default decisions.

Changing from decision \( l \) to \( \neg l \)

- LEGAL: Just update \( \mathcal{D}^a \) and \( \mathcal{D}^d \)
- RECONFIGURE: Change \( \mathcal{D}^d \) so that the new decision can be accommodated. Use MCSes for that.
Handling Reconfiguration Scenarios

Modeling the problem.

- $\mathcal{F}$ . . . configuration instance
- $\mathcal{D}^a$ . . . decisions made by the agent
  - E.g. $\{\neg x, y\}$
- $\mathcal{D}^d$ . . . default decisions.

Changing from decision $l$ to $\neg l$

- LEGAL: Just update $\mathcal{D}^a$ and $\mathcal{D}^d$
- RECONFIGURE: Change $\mathcal{D}^d$ so that the new decision can be accommodated. Use MCSes for that.
- ILLEGAL: Provide an explanation why.
Lazy Approach

1 Function Change (l)
2 begin
3 \((st_1, \mu_1, core_1) \leftarrow \text{SAT}(F \cup D^a \cup D^d \setminus \{l\} \cup \{-l\})\)
4 \((st_2, \mu_1, core_2) \leftarrow \text{SAT}(F \cup D^a \setminus \{l\} \cup \{-l\})\)
5 if \(st_1\) then
6 \(D^a \leftarrow D^a \setminus \{l\} \cup \{-l\}\)
7 \(D^d \leftarrow D^d \setminus \{l\}\)
8 return LEGAL
9 if \(st_2\) then
10 \(D^a \leftarrow D^a \setminus \{l\} \cup \{-l\}\)
11 \(D^d \leftarrow D^d \setminus \{l\}\)
12 \(D^d = \text{Reconfigure}(F \wedge D^a, D^d)\)
13 return RECONFIGURE
14 Explain\((F, D^a \setminus \{l\} \cup \{-l\})\)
15 return ILLEGAL
Eager Approach—Change

If $\mathcal{F} \cup \mathcal{D}^a \cup \mathcal{D}^d \setminus \{l\} \models l$, remember a reason $R_l \subseteq \mathcal{D}^a \cup \mathcal{D}^d$ s.t. $\mathcal{F} \cup R_l \models l$.

1 **Function** Change ($l$)
2 begin
3     foreach $k$ s.t. $l \in R_k$ do
4         remove $R_k$ as reason for $k$
5         mark $k$ as LEGAL
6     foreach $k$ is LEGAL do
7         Check($k$)
Eager Approach—Check Literal

Function Check ($l$)

begin

(st$_1$, $\mu_1$, core$_1$) ← SAT($\mathcal{F} \cup D^a \cup D^d \setminus \{l\} \cup \{-l\}$)
(st$_2$, $\mu_1$, core$_2$) ← SAT($\mathcal{F} \cup D^a \setminus \{l\} \cup \{-l\}$)

if st$_1$ then

mark $l$ as LEGAL

else if st$_2$ then

mark $l$ as RECONFIGURE

$R_1$ ← core$_1 \cap (D^a \cup D^d) \setminus \{-l\}$

else

mark $l$ as ILLEGAL

$R_1$ ← core$_2 \cap D^a \setminus \{-l\}$
## Experimental Results

<table>
<thead>
<tr>
<th></th>
<th>SPLOT rl</th>
<th>SPLOT rnd</th>
<th>LVAT mod</th>
<th>LVAT hard</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm</strong></td>
<td>eager</td>
<td>eager</td>
<td>lazy</td>
<td>lazy</td>
</tr>
<tr>
<td><strong>Maximal time</strong></td>
<td>0.008 s</td>
<td>1.3 s</td>
<td>2 s</td>
<td>41.76 s</td>
</tr>
<tr>
<td><strong>time &lt; 0.5 s</strong></td>
<td>100%</td>
<td>99.8%</td>
<td>99.9%</td>
<td>86%</td>
</tr>
<tr>
<td><strong>time &lt; 1.5 s</strong></td>
<td>100%</td>
<td>100%</td>
<td>99.9%</td>
<td>91%</td>
</tr>
<tr>
<td><strong># RECONF</strong></td>
<td>4199</td>
<td>79860</td>
<td>43821</td>
<td>28259</td>
</tr>
<tr>
<td><strong># ILLEGAL</strong></td>
<td>3987</td>
<td>97103</td>
<td>167795</td>
<td>55903</td>
</tr>
<tr>
<td><strong># LEGAL</strong></td>
<td>4314</td>
<td>73037</td>
<td>33384</td>
<td>5838</td>
</tr>
<tr>
<td><strong>Max. RECON</strong></td>
<td>0.006 s</td>
<td>0.5 s</td>
<td>2 s</td>
<td>41.76 s</td>
</tr>
<tr>
<td><strong>Max. ILLEG</strong></td>
<td>0.001 s</td>
<td>0.02 s</td>
<td>0.92 s</td>
<td>11.10 s</td>
</tr>
<tr>
<td><strong>Max. LEGAL</strong></td>
<td>0.006 s</td>
<td>0.32 s</td>
<td>0.005 s</td>
<td>0.01 s</td>
</tr>
<tr>
<td><strong>Max. INITIAL</strong></td>
<td>0.008 s</td>
<td>1.3 s</td>
<td>0.01 s</td>
<td>0.15 s</td>
</tr>
<tr>
<td><strong># models</strong></td>
<td>25</td>
<td>10</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td><strong># variables</strong></td>
<td>60–366</td>
<td>10,000–10,000</td>
<td>684–14910</td>
<td>23,516–62,482</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- SAT-based support for reconfiguration.

- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.
  - For RECONFIGURE, use MCS to change other variables.
  - For ILLEGAL, use MUS to explain.

- Developed a lazy algorithm, which computes a status of a variable on demand.
- Developed an eager algorithm, which maintains a status of each variable at all times.
- Eager—10^2 variables, Lazy—10^4 variables.

- How to deal with "large" reconfiguration?
- How to apply reconfiguration in the vicinity only?
- What do users want from reconfiguration?
Conclusions and Future Work

- SAT-based support for reconfiguration.
- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.

Developed a lazy algorithm, which computes a status of a variable on demand.
Developed an eager algorithm, which maintains a status of each variable at all times.

- Eager—10
- Lazy—10

How to deal with "large" reconfiguration?
How to apply reconfiguration in the vicinity only?
What do users want from reconfiguration?
Conclusions and Future Work

- SAT-based support for reconfiguration.
- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.
- For RECONFIGURE, use MCS to change other variables.

- Developed a lazy algorithm, which computes a status of a variable on demand.
- Developed an eager algorithm, which maintains a status of each variable at all times.

- Eager—10^2 variables, Lazy—10^4 variables.

- How to deal with "large" reconfiguration?
- How to apply reconfiguration in the vicinity only?
- What do users want from reconfiguration?
Conclusions and Future Work

- SAT-based support for reconfiguration.
- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.
- For RECONFIGURE, use MCS to change other variables.
- For ILLEGAL, use MUS to explain.

Developed a lazy algorithm, which computes a status of a variable on demand.
Developed an eager algorithm, which maintains a status of each variable at all times.

Eager—10 \(2\) variables, Lazy—10 \(4\) variables.

How to deal with “large” reconfiguration?

How to apply reconfiguration in the vicinity only?

What do users want from reconfiguration?
Conclusions and Future Work

- SAT-based support for reconfiguration.
- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.
- For RECONFIGURE, use MCS to change other variables.
- For ILLEGAL, use MUS to explain.
- Developed a lazy algorithm, which computes a status of a variable on demand.
Conclusions and Future Work

- SAT-based support for reconfiguration.
- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.
- For RECONFIGURE, use MCS to change other variables.
- For ILLEGAL, use MUS to explain.
- Developed a lazy algorithm, which computes a status of a variable on demand.
- Developed an eager algorithm, which maintains a status of each variable at all times.
Conclusions and Future Work

- SAT-based support for reconfiguration.
- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.
- For RECONFIGURE, use MCS to change other variables.
- For ILLEGAL, use MUS to explain.
- Developed a lazy algorithm, which computes a status of a variable on demand.
- Developed an eager algorithm, which maintains a status of each variable at all times.
- Eager—$10^2$ of variables, Lazy—$10^4$ variables.
Conclusions and Future Work

- SAT-based support for reconfiguration.
- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.
- For RECONFIGURE, use MCS to change other variables.
- For ILLEGAL, use MUS to explain.
- Developed a lazy algorithm, which computes a status of a variable on demand.
- Developed an eager algorithm, which maintains a status of each variable at all times.
- Eager—$10^2$ of variables, Lazy—$10^4$ variables.
- How to deal with “large” reconfiguration?
Conclusions and Future Work

- SAT-based support for reconfiguration.
- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.
- For RECONFIGURE, use MCS to change other variables.
- For ILLEGAL, use MUS to explain.
- Developed a lazy algorithm, which computes a status of a variable on demand.
- Developed an eager algorithm, which maintains a status of each variable at all times.
- Eager—$10^2$ of variables, Lazy—$10^4$ variables.
- How to deal with “large” reconfiguration?
- How to apply reconfiguration in the vicinity only?
Conclusions and Future Work

- SAT-based support for reconfiguration.
- 3 types of scenarios: LEGAL, RECONFIGURE, and ILLEGAL.
- For RECONFIGURE, use MCS to change other variables.
- For ILLEGAL, use MUS to explain.
- Developed a lazy algorithm, which computes a status of a variable on demand.
- Developed an eager algorithm, which maintains a status of each variable at all times.
- Eager—$10^2$ of variables, Lazy—$10^4$ variables.
- How to deal with “large” reconfiguration?
- How to apply reconfiguration in the vicinity only?
- What do users want from reconfiguration?
Thank you for your attention!

Questions?