Do SAT Solvers Make Good Configurators?

Mikoláš Janota

Lero
University College Dublin
Ireland

ASPL at SPLC 2008

lero

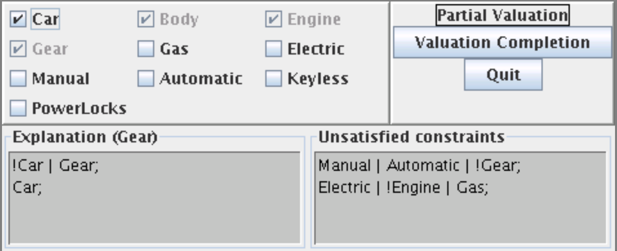
SFI grant no. 03/CE2/I303_1

Introduction

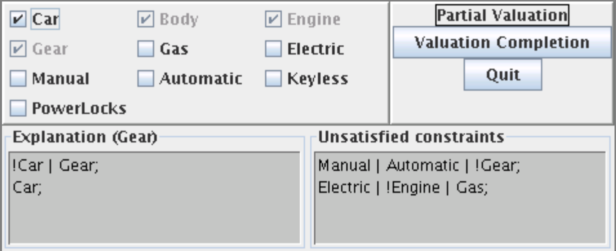
Configuration

- a *feature models* represent the set of products we are interested in and the dependencies between them
- customer selects a product in the *configuration process*
- the product should fulfill the desires of the customer but must respect the *constraints* imposed by the feature model

Interactive feedback



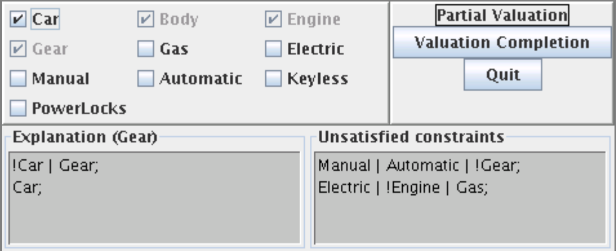
Interactive feedback



Functionality

- feedback takes place as the choices are being made

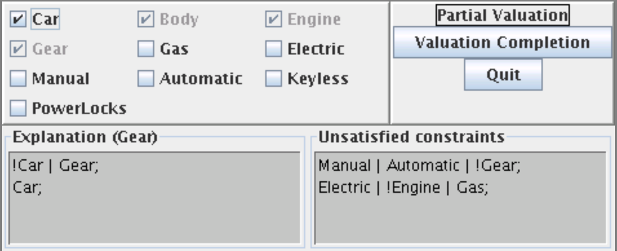
Interactive feedback



Functionality

- feedback takes place as the choices are being made
- disabling choices that do not lead to a solution

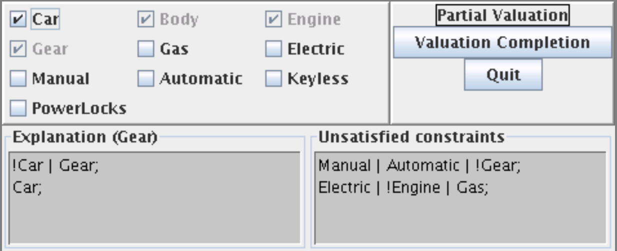
Interactive feedback



Functionality

- feedback takes place as the choices are being made
- disabling choices that do not lead to a solution
- never let the user violate the constrains (*backtrack freeness*)

Interactive feedback



Functionality

- feedback takes place as the choices are being made
- disabling choices that do not lead to a solution
- never let the user violate the constrains (*backtrack freeness*)
- explaining why a value is locked

How Do We Go About This?

Use a SAT Solver

- determines the satisfiability of a given Boolean formula
- operates on Conjunctive Normal Form (CNF)
- a certification of the response is produced
- nowadays SAT solvers are *very* efficient

How Do We Go About This?

Use a SAT Solver

- determines the satisfiability of a given Boolean formula
- operates on Conjunctive Normal Form (CNF)
- a certification of the response is produced
- nowadays SAT solvers are *very* efficient

Assumptions

- constraints encoded in a CNF
- decisions so far encoded as a conjunction of literals

$$\phi \equiv f_1 \wedge \neg f_8 \wedge \ldots$$

SAT Solver for Configuration

Testing all free features after each user's decision

TEST-VARS()

foreach x that was not assigned to by the user
  do $\mathit{CanBeTrue} \leftarrow \mathrm{TEST\text{-}SAT}(\phi, x)$
    $\mathit{CanBeFalse} \leftarrow \mathrm{TEST\text{-}SAT}(\phi, \neg x)$
    if $\neg \mathit{CanBeTrue} \wedge \neg \mathit{CanBeFalse}$
      then error "Unsatisfiable constraint!"
    if $\neg \mathit{CanBeTrue}$ **then** SET(x, FALSE)
    if $\neg \mathit{CanBeFalse}$ **then** SET(x, TRUE)
    if $\mathit{CanBeTrue} \wedge \mathit{CanBeFalse}$
      then RESET(x)
        UNLOCK(x)
      else LOCK(x)

Can We Improve This?

SAT

- For satisfiable queries, the SAT solver returns with a satisfying assignment.
- All the values in this assignment are satisfiable and don't need to be queried for.

Can We Improve This?

SAT

- For satisfiable queries, the SAT solver returns with a satisfying assignment.
- All the values in this assignment are satisfiable and don't need to be queried for.

UNSAT

- Can a negative response of the solver help in the future?
- Example

$$\left.\begin{array}{l} f_1 => f_2 \\ \neg f_2 \\ \ldots \end{array}\right\} \neg f_1$$

- Recording disabled values *may* help with further queries.

Satisfiability with Caching

- *KnownValues* represent values known to be SAT
- *DisabledValues* represent values known to be UNSAT

$\textsc{Test-SAT}(\phi\text{: } \mathit{Formula},\ l\text{: } \mathit{Literal}) : \mathit{Boolean}$

if $l \in \mathit{KnownValues}$ **then return** $\textsc{True}$
if $l \in \mathit{DisabledValues}$ **then return** $\textsc{False}$
$L \leftarrow \mathrm{SAT}(\phi \wedge l \wedge \bigwedge_{k \in \mathit{DisabledValues}} \neg k)$
if $L \neq$ **null**
    then $\mathit{KnownValues} \leftarrow \mathit{KnownValues} \cup L$
    else $\mathit{DisabledValues} \leftarrow \mathit{DisabledValues} \cup \{l\}$
return $L \neq$ **null**

Explanations

- The solver produces a unsatisfiable subset of given formulas.
- This may not be minimal, several techniques how to minimize.
- In the tool an iterative technique by Zhang and Malik.

Discussion and Future Work

Comparing to Binary Decision Diagrams (BDDs)

- It is expected to be slower than *but* much less likely to choke.
- The form of the formula is preserved and hence can be used in the explanations.
- Requires CNF *however* any formula can be *clausified* in polynomial size.

Discussion and Future Work

Comparing to Binary Decision Diagrams (BDDs)

- It is expected to be slower than *but* much less likely to choke.
- The form of the formula is preserved and hence can be used in the explanations.
- Requires CNF *however* any formula can be *clausified* in polynomial size.

Future work

- Use proofs for more efficient cache discarding.
- Could this approach work for non-Boolean domains?

Discussion and Future Work

Comparing to Binary Decision Diagrams (BDDs)

- It is expected to be slower than *but* much less likely to choke.
- The form of the formula is preserved and hence can be used in the explanations.
- Requires CNF *however* any formula can be *clausified* in polynomial size.

Future work

- Use proofs for more efficient cache discarding.
- Could this approach work for non-Boolean domains?

A java implementation, including a SAT solver, available at

`kind.ucd.ie`

When to Discard Caches?

- With a new decision asserted, *KnownValues* may change and thus get discarded whereas *DisabledValues* remain the same.
- When a decision is retracted, *KnownValues* remain whereas *DisabledValues* are discarded.