Optimising Quantified Expressions in Constraint Models

Ian P. Gent, Ian Miguel and Andrea Rendl

University of St Andrews, UK
AIT Austrian Institute of Technology, Austria

September 2010
Workshop on Modelling and Reformulation
Context of this Work

- **Quantified expressions** in solver-independent constraint modelling languages
Context of this Work

- **Quantified expressions** in solver-independent constraint modelling languages

- Example:
  
  \[
  \text{forall } i,j:\text{int}(1..n) . \\
  (i \neq j) \Rightarrow (q[i]-i \neq q[j]-j)
  \]
Context of this Work

- **Quantified expressions** in solver-independent constraint modelling languages

  Example:
  
  \[
  \text{forall } i,j:\text{int}(1..n) . \\
  (i \neq j) \Rightarrow (q[i]-i \neq q[j]-j)
  \]

- powerful means to compactly represent a set of expressions
Context of this Work

- **Quantified expressions** in solver-independent constraint modelling languages

- Example:
  \[
  \forall i,j : \text{int}(1..n) .
  (i \neq j) \Rightarrow (q[i]-i \neq q[j]-j)
  \]

- powerful means to compactly represent a set of expressions

- same structure in all constraint modelling languages
Quantified expressions in solver-independent constraint modelling languages

Example:

\[
\forall i,j : \text{int}(1..n) . \\
(i \neq j) \Rightarrow (q[i] - i \neq q[j] - j)
\]

powerful means to compactly represent a set of expressions

same structure in all constraint modelling languages

restriction: no decision variables in \( i_1, \ldots, i_m \) and \( \text{int}(lb..ub) \)
Goal and Contributions

- **Our Observation:**
  quantified expressions can contain **redundancies**, often when formulated by **novices**

- **Our Goal:**
  automatically improve poorly formulated quantified expressions

- **Our Contributions:**
  we consider 2 kinds of redundancies
  we propose means to detect and address those redundancies
Goal and Contributions

- **Our Observation:**
  quantified expressions can contain *redundancies*, often when formulated by *novices*

- **Our Goal:**
  automatically *improve* poorly formulated quantified expressions
Our Observation: quantified expressions can contain redundancies, often when formulated by novices.

Our Goal: automatically improve poorly formulated quantified expressions.

Our Contributions: we consider 2 kinds of redundancies.
Goal and Contributions

- **Our Observation:**
  quantified expressions can contain **redundancies**, often when formulated by **novices**

- **Our Goal:**
  automatically **improve** poorly formulated quantified expressions

- **Our Contributions:**
  - we consider **2 kinds of redundancies**
  - we propose means to **detect** and **address** those redundancies
1. Loop-invariant Expressions

2. Weak Guards

3. Summary
Loop-invariant Expressions

- **Idea:** analyse equivalent representations of quantified expressions

Example: 

\[(x = 0) \Rightarrow \forall \ i \in D. (x[i] = i) \equiv (x = 0) \Rightarrow (x[i] = i)\]

we call \((x = 0)\) loop-invariant

**Question:** which representation is better?
Loop-invariant Expressions

- **Idea**: analyse equivalent representations of quantified expressions

- **Example**: \( (x = 0) \Rightarrow \forall i \in D. (x[i] = i) \)
Loop-invariant Expressions

- **Idea**: analyse equivalent representations of quantified expressions
- Example:  

\[
(x = 0) \Rightarrow \forall i \in D. (x[i] = i) \equiv \forall i \in D. (x = 0) \Rightarrow (x[i] = i)
\]

we call ‘\(x = 0\)’ loop-invariant

**Question**: which representation is better?
Loop-invariant Expressions

- **Idea**: analyse equivalent representations of quantified expressions

- **Example**: 
  
  \[(x = 0) \Rightarrow \forall i \in D. (x[i] = i)\]
  
  \[\equiv\]
  
  \[\forall i \in D. (x = 0) \Rightarrow (x[i] = i)\]
  
- we call ‘\((x = 0)\)’ loop-invariant
Loop-invariant Expressions

- **Idea**: analyse equivalent representations of quantified expressions

- **Example**: \( (x = 0) \Rightarrow \forall i \in D. (x[i] = i) \)
  \[ \equiv \]
  \( \forall i \in D. (x = 0) \Rightarrow (x[i] = i) \)

- we call ‘\( (x = 0) \)’ loop-invariant

- **Question**: which representation is better?
Optimising Quantified Expressions

Loop-invariant Expressions

Many different cases....

1. $A \land \forall I E_I \equiv \forall I A \land E_I$
Many different cases....

1. $A \land \forall_i E_i \equiv \forall_i A \land E_i$
2. $A \lor \exists_i E_i \equiv \exists_i A \lor E_i$
3. $mA + \sum_i E_i \equiv \sum_i A + E_i$ \hspace{1cm} \text{where } m = |I|$
4. $A \lor (\forall_i E_i) \equiv \forall_i A \lor E_i$
5. etc

Intuitively, we expect the outside-representation to be better... is this true for all cases?
Many different cases....

1. \( A \land \forall E_i \equiv \forall E_i A \land E_i \)
2. \( A \lor \exists E_i \equiv \exists E_i A \lor E_i \)
3. \( mA + \sum E_i \equiv \sum E_i A + E_i \) where \( m = |I| \)
4. \( A \lor (\forall E_i) \equiv \forall E_i A \lor E_i \)
5. etc

Intuitively, we expect the outside-representation to be better...
Optimising Quantified Expressions

Loop-invariant Expressions

Many different cases....

1. $A \land \forall I E_i \equiv \forall I (A \land E_i)$
2. $A \lor \exists I E_i \equiv \exists I (A \lor E_i)$
3. $mA + \sum I E_i \equiv \sum I (mA + E_i)$ where $m = |I|$
4. $A \lor (\forall I E_i)) \equiv \forall I (A \lor E_i)$
5. etc

Intuitively, we expect the outside-representation to be better... is this true for all cases?
We compare representations at solver level (flat representation)
We compare representations at **solver level** (flat representation).

- We assume the solver provides:
  - (reifyable) $n$-ary conjunction ($\forall$)
  - (reifyable) $n$-ary disjunction ($\exists$)
  - $n$-ary sum ($\sum$)
We compare representations at \textbf{solver level} (flat representation).

We assume the solver provides:
- (reifyable) \(n\)-ary conjunction (\(\forall\))
- (reifyable) \(n\)-ary disjunction (\(\exists\))
- \(n\)-ary sum (\(\sum\))

Let’s look at one case (see paper for other cases):

\[ A \Rightarrow (\forall I E_I) \equiv \forall I A \Rightarrow E_I \]
## Comparing Representations

<table>
<thead>
<tr>
<th></th>
<th>Inside-Representation</th>
<th>Outside-Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>$(\forall I A \Rightarrow E_I)$</td>
<td>$A \Rightarrow (\forall I E_I)$</td>
</tr>
</tbody>
</table>
## Comparing Representations

<table>
<thead>
<tr>
<th></th>
<th>Inside-Representation</th>
<th>Outside-Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Original</strong></td>
<td>$(\forall I A \Rightarrow E_I)$</td>
<td>$A \Rightarrow (\forall I E_I)$</td>
</tr>
<tr>
<td><strong>Unrolled</strong></td>
<td>$(A \Rightarrow E_1) \land \ldots \land (A \Rightarrow E_k)$</td>
<td>$A \Rightarrow (E_1 \land \ldots \land E_k)$</td>
</tr>
</tbody>
</table>
Comparing Representations

<table>
<thead>
<tr>
<th></th>
<th>Inside-Representation</th>
<th>Outside-Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>$(\forall I A \Rightarrow E_I)$</td>
<td>$A \Rightarrow (\forall I E_I)$</td>
</tr>
<tr>
<td>Unrolled</td>
<td>$(A \Rightarrow E_1) \land \ldots \land (A \Rightarrow E_k)$</td>
<td>$A \Rightarrow (E_1 \land \cdots \land E_k)$</td>
</tr>
<tr>
<td>Flat (unnested)</td>
<td>$a \Rightarrow e_1 \land \ldots \land a \Rightarrow e_k$</td>
<td>$aux \iff (e_1 \land \cdots \land e_k)$, $a \Rightarrow aux$</td>
</tr>
</tbody>
</table>
Comparing Representations

<table>
<thead>
<tr>
<th></th>
<th>Inside-Representation</th>
<th>Outside-Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Original</strong></td>
<td>$(\forall I A \Rightarrow E_I)$</td>
<td>$A \Rightarrow (\forall I E_I)$</td>
</tr>
<tr>
<td><strong>Unrolled</strong></td>
<td>$(A \Rightarrow E_1) \land$</td>
<td>$A \Rightarrow (E_1 \land \cdots \land E_k)$</td>
</tr>
<tr>
<td></td>
<td>$\cdots$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(A \Rightarrow E_k)$</td>
<td></td>
</tr>
<tr>
<td><strong>Flat</strong></td>
<td>$a \Rightarrow e_1$</td>
<td>$aux \Leftrightarrow (e_1 \land \cdots \land e_k)$</td>
</tr>
<tr>
<td><em>(unnested)</em></td>
<td>$\cdots$</td>
<td>$a \Rightarrow aux$</td>
</tr>
<tr>
<td></td>
<td>$a \Rightarrow e_k$</td>
<td></td>
</tr>
</tbody>
</table>

- **0 auxiliary variables**
- **k constraints**
- **1 auxiliary variable**
- **2 constraints**
Comparing Representations

- **Inside-Representation**: more constraints (increasing with \( k \)), no additional variables
Comparing Representations

- **Inside-Representation**: more constraints (increasing with $k$), no additional variables
- **Outside-Representation**: only two constraints but 1 additional variable
Comparing Representations

- **Inside-Representation**: more constraints (increasing with $k$), no additional variables
- **Outside-Representation**: only two constraints but 1 additional variable
- Let’s compare the representations in an example!
Example: Peaceful Army of Queens

Place two equally-sized armies of queens on a chess board such that they do not attack another, maximising the army size.
Non-attacking Constraints in model based on Smith et al (2004):

\[
\text{forall fields}(i,j) \text{ on the chess board}.
\]
Peaceful Army of Queens: Outside Representation

Non-attacking Constraints in model based on Smith et al (2004):

\[
\text{forall } fields(i,j) \text{ on the chess board.}
\]
\[
\text{white queen at field}(i,j) \implies
\]
Peaceful Army of Queens: Outside Representation

Non-attacking Constraints in model based on Smith et al (2004):

\[
\forall \text{fields}(i,j) \text{ on the chess board.} \\
\quad \text{white queen at field}(i,j) \implies \forall k. \\
\quad \quad \text{no black queen at field}(i,k) \text{ (same column)}
\]
Peaceful Army of Queens: Outside Representation

Non-attacking \textbf{Constraints} in model based on Smith et al (2004):

\begin{verbatim}
forall fields(i,j) on the chess board.
  white queen at field(i,j) \implies
    forall k.
      no black queen at field(i,k) (same column)
      \land no black queen at field(k,j) (same row)
\end{verbatim}
Non-attacking Constraints in model based on Smith et al (2004):

\[
\text{forall } fields(i,j) \text{ on the chess board.} \\
\quad \text{white queen at field}(i,j) \quad \Rightarrow \\
\quad \text{forall } k. \\
\quad \quad \text{no black queen at field}(i,k) \text{ (same column)} \\
\quad \quad \land \text{ no black queen at field}(k,j) \text{ (same row)} \\
\quad \quad \land \text{ no black queen at field}(i+k,j+k) \text{ (NW-diagonal)} \\
\quad \quad \land \text{ no black queen at field}(i-k,j+k) \text{ (SW-diagonal)} \\
\quad \quad \land \text{ no black queen at field}(i+k,j-k) \text{ (NE-diagonal)} \\
\quad \quad \land \text{ no black queen at field}(i-k,j-k) \text{ (SE-diagonal)}
\]
Alternatively, moving loop-invariant expression inside:

\[ \text{forall fields}(i,j) \text{ on the chess board.} \]
Peaceful Army of Queens: Inside Representation

*Alternatively*, moving loop-invariant expression inside:

\[
\forall i, j \text{ fields on the chess board.} \\
\forall k. \quad \text{white queen at field}(i, j) \implies \\
\quad \text{no black queen at field}(i, k) \text{ (column)}
\]
Peaceful Army of Queens: Inside Representation

Alternatively, moving loop-invariant expression inside:

\[
\forall \text{fields}(i,j) \text{ on the chess board.} \\
\quad \forall \, k. \\
\quad \text{white queen at field}(i,j) \implies \\
\quad \quad \text{no black queen at field}(i,k) \text{ (column)} \\
\land \forall \, k. \\
\quad \text{white queen at field}(i,j) \implies \\
\quad \quad \land \text{no black queen at field}(k,j) \text{ (row)}
\]
Peaceful Army of Queens: Inside Representation

Alternatively, moving loop-invariant expression inside:

\[
\text{forall fields}(i,j) \text{ on the chess board.}
\]
\[
\text{forall } k.
white \text{ queen at field}(i,j) \Rightarrow
\]
\[
\text{no black queen at field}(i,k) \text{ (column)}
\]
\[
\land \text{forall } k.
white \text{ queen at field}(i,j) \Rightarrow
\]
\[
\land \text{no black queen at field}(k,j) \text{ (row)}
\]
\[
\land \text{forall } k.
white \text{ queen at field}(i,j) \Rightarrow
\]
\[
\land \text{no black queen at field}(i+k,j+k) \text{ (NW-diagonal)}
\]

...
What did we do?

1. We modelled two different PAQ models (in Essence')
Comparing Inside- and Outside-Representation

What did we do?

1. We modelled two different PAQ models (in Essence’)
2. We translated both models to solvers Gecode and Minion (using Tailor), generating:
Comparing Inside- and Outside-Representation

What did we do?

1. We modelled two different PAQ models (in Essence’)
2. We translated both models to solvers Gecode and Minion (using Tailor), generating:
   - outside-representation
   - inside-representation

for both models
What did we do?

1. We modelled two different PAQ models (in Essence’)
2. We translated both models to solvers Gecode and Minion (using Tailor), generating:
   - outside-representation
   - inside-representation
   for both models
3. We solved both representations using the same solving setup
Comparing Number of Constraints

**Inside**-Representation has far **more** constraints than **Outside**-Representation.

![Diagram showing the comparison of constraints between Inside and Outside representations for different problem classes.](image-url)
Comparing Number of Auxiliary Variables

**Inside**-Representation has **30% less** auxiliary variables than **Outside**-Representation

![Graph showing variable reduction with Inside Representation](image)
Comparing Number Solving Performance

- **Inside-Rep.** better in Minion (speedup of max. 300%)
- **Inside-Rep.** slightly better in Gecode (speedup of max. 30%)
Conclusion on Loop-Invariant Expressions

- **Against our expectations**: it can be beneficial to move loop-invariant expressions into quantifications
Conclusion on Loop-Invariant Expressions

- **Against our expectations**: it can be beneficial to move loop-invariant expressions into quantifications

- **Difficult to make a *general* statement**
  - depends on solver (provided propagators, architecture, etc)
  - depends on problem structure
Conclusion on Loop-Invariant Expressions

- **Against our expectations**: it can be beneficial to move loop-invariant expressions into quantifications.

- Difficult to make a **general** statement:
  - depends on solver (provided propagators, architecture, etc)
  - depends on problem structure

- Tailor can **automatically reformulate** quantifications to inside/outside-representation:
  - user can choose preferable representation (for each case) in translation settings
1. Loop-invariant Expressions

2. Weak Guards

3. Summary
Weak Guards

- A **guard** $B$ for an expression $E$ has to hold to enforce $E$
  - $B \Rightarrow E$

Example:

```
forall i, j in (1..n).
(i \neq j) \Rightarrow \text{queen}[i] + i \neq \text{queen}[j] + j
```
Weak Guards

- A **guard** $B$ for an expression $E$ has to hold to enforce $E$
  - $B \Rightarrow E$

- Often used in modelling, mostly to restrict quantifying variables
Optimising Quantified Expressions

Weak Guards

Weak Guards

- A **guard** $B$ for an expression $E$ has to hold to enforce $E$
  - $B \Rightarrow E$

- Often used in modelling, mostly to restrict quantifying variables

- **Example:**

  $\forall i,j \in (1..n). 
  (i \neq j) \Rightarrow queen[i] + i \neq queen[j] + j$
Weak Guards

- If guards are weak they yield duplicate constraints
If guards are weak they yield duplicate constraints

\[
\text{forall } i, j \text{ in } (1..n). \quad (i \neq j) \Rightarrow \text{queen}[i] + i \neq \text{queen}[j] + j
\]
Weak Guards

- If guards are **weak** they yield duplicate constraints

- **forall** \( i, j \) in \((1..n)\).
  \[
  (i \neq j) \; \Rightarrow \; queen[i] + i \neq queen[j] + j
  \]

- is unrolled to:
  
  \[
  \begin{align*}
  \end{align*}
  \]

  etc
Weak Guards

- If guards are **weak** they yield duplicate constraints

  - For all \( i, j \) in \((1..n)\).
    \[
    (i \neq j) \implies \text{queen}[i] + i \neq \text{queen}[j] + j
    \]

- Is unrolled to:
  
  - queen[1]+1 \neq \text{queen}[2]+2, \quad \text{queen}[1]+1 \neq \text{queen}[3]+3,
  
  - queen[2]+2 \neq \text{queen}[1]+1, \quad \text{queen}[2]+2 \neq \text{queen}[3]+3,
  
  - queen[3]+3 \neq \text{queen}[2]+2, \quad \text{queen}[3]+3 \neq \text{queen}[1]+1,

  etc
Weak Guards

- If guards are **weak** they yield duplicate constraints

- **forall** $i, j \text{ in } (1..n)$.
  \[(i \neq j) \Rightarrow \text{queen}[i] + i \neq \text{queen}[j] + j\]

- is unrolled to:
  
  queen[1]+1 \neq queen[2]+2,  
  queen[1]+1 \neq queen[3]+3,  
  queen[2]+2 \neq queen[1]+1,  
  queen[2]+2 \neq queen[3]+3,  
  queen[3]+3 \neq queen[2]+2,  
  queen[3]+3 \neq queen[1]+1,  
  \text{etc}
Addressing Weak Guards

- **Option1**: remove duplicate constraints after quantification is unrolled
Addressing Weak Guards

- **Option 1**: remove duplicate constraints after quantification is unrolled
  - **problem**: only possible when quantification can be unrolled, i.e. all parameters are known
Addressing Weak Guards

- **Option 1**: remove duplicate constraints after quantification is unrolled
  - **problem**: only possible when quantification can be unrolled, i.e. all parameters are known

- **Option 2**: strengthen the guard!
Our Idea: use unification to strengthen guards
Our Idea: use unification to strengthen guards

Unification Example:

What is the unifier for ‘x + i’ and ‘x + 3’?
Our Idea: use unification to strengthen guards

Unification Example:
- What is the unifier for ‘x + i’ and ‘x + 3’?
  - \( u = \{3/i\} \) (i substituted with 3)

We want to demonstrate the algorithm on an example...
A Golomb Ruler has \( n \) ticks such that the distance between each tick is different, minimising the length of the ruler.
A Golomb Ruler has \( n \) ticks such that the distance between each tick is different, minimising the length of the ruler.

Sample Golomb Ruler with 4 ticks and length 6:
Strengthening the Guard in Golomb Ruler

‘The distances between all ticks are different’-Constraint:
Strengthening the Guard in Golomb Ruler

‘The distances between all ticks are different’-Constraint:

\[
\forall i_1, i_2, i_3, i_4 : \text{TICKS.} \quad ((i_1 > i_2) \land (i_3 > i_4) \land (i_2 \neq i_4)) \implies (\text{ruler}[i_1] - \text{ruler}[i_2] \neq \text{ruler}[i_3] - \text{ruler}[i_4])
\]
STRENGTHEN_GUARD(∀_i : D.B_i ⇒ E_i)
STRENGTHEN_GUARD(∀I : D.BI ⇒ EI)

(1) If EI’s root node corresponds to a binary commutative operator then continue, otherwise stop.
STRENGTHEN_GUARD(∀ᵢ : D.Bᵢ ⇒ Eᵢ)

1. If Eᵢ’s root node corresponds to a binary commutative operator then continue, otherwise stop.

forall i₁, i₂, i₃, i₄ : TICKS.
((i₁ > i₂) ∧ (i₃ > i₄) ∧ (i₂ ≠ i₄)) ⇒ (ruler[i₁]-ruler[i₂] ≠ ruler[i₃]-ruler[i₄])
STRENGTHEN_GUARD(∀₁ : D.B₁ ⇒ E₁)

(2) Compute the set of unifiers $U$ for the two children of $E₁$, $e₁$ and $e₂$.

UNIFY (ruler[i1]-ruler[i2], ruler[i3]-ruler[i4]):

\[
\begin{align*}
  u₁ &= \{i₁/i₃ \land i₂/i₄\} \\
  u₂ &= \{i₃/i₁ \land i₄/i₂\} \\
  u₃ &= \{i₃/i₁ \land i₂/i₄\} \\
  u₄ &= \{i₁/i₃ \land i₄/i₂\}
\end{align*}
\]
STRENGTHEN_GUARD(\(\forall I : D.B_I \Rightarrow E_I\))

- (3) Search \(U\) for unifiers from which we can deduce equivalence of the quantifying variables.

UNIFY (ruler[i1]-ruler[i2], ruler[i3]-ruler[i4]):

\[
\begin{align*}
  u_1 &= \{i_1/i_3 \land i_2/i_4\} \quad u_2 = \{i_3/i_1 \land i_4/i_2\} \\
  u_3 &= \{i_3/i_1 \land i_2/i_4\} \quad u_4 = \{i_1/i_3 \land i_4/i_2\}
\end{align*}
\]

we deduce that \((i_1 = i_3) \land (i_2 = i_4)\)
Optimising Quantified Expressions

Weak Guards

Strengthening the Guard in Golomb Ruler

STRENGTHEN_GUARD(∀i : D.Bi ⇒ Ei)

(4) Add lex-ordering constraint C on all quantifying variables whose equivalence renders e₁ and e₂ equivalent

C:  \( i₁, i₂ \leq_{\text{lex}} i₃, i₄ \)

hence \( (i₁ \leq i₃) \land (i₁ < i₃ \lor i₂ \leq i₄) \)
Yielding the constraint with **strengthened guard**:

\[
\text{forall } i_1, i_2, i_3, i_4 : \text{TICKS.} \\
((i_1 > i_2) \land (i_3 > i_4) \land (i_2 \neq i_4) \land \\
(i_1 \leq i_3) \land (i_1 < i_3 \lor i_2 \leq i_4)) \\
\Rightarrow \\
(ruler[i_1] - ruler[i_2] \neq ruler[i_3] - ruler[i_4])
\]
Yielding the constraint with **strengthened guard**: 

\[
\forall i_1, i_2, i_3, i_4: \text{TICKS.} \\
((i_1 > i_2) \land (i_3 > i_4) \land (i_2 \neq i_4) \land (i_1 \leq i_3) \land (i_1 < i_3 \lor i_2 \leq i_4)) \implies (\text{ruler}[i_1]-\text{ruler}[i_2] \neq \text{ruler}[i_3]-\text{ruler}[i_4])
\]

**However**: we have not implemented the algorithm yet!
Effects of Duplicate constraints

- How bad is the effect of duplicate constraints due to weak guards?
  - in other words: is it worth putting energy into strengthening guards?
Effects of Duplicate constraints

- How bad is the **effect of duplicate constraints** due to weak guards?
  - in other words: is it worth putting energy into strengthening guards?

- We analyse the effects on two naive models in solver Minion and Gecode:
  - Naive n-Queens
  - Naive Golomb Ruler
For both solvers: constant for n-Queens, linear within Golomb Ruler

The Number of Duplicate Constraints
Effect on Solving Performance

strong effect in Gecode, mild effect in Minion

Optimising Quantified Expressions

Weak Guards

Problem Classes
- golomb (Minion)
- golomb (Gecode)
- nQueensNaive (Minion)
- nQueensNaive (Gecode)
same solving time
Conclusions for Weak Guards

- Duplicate constraints *can impair* the solving performance
Conclusions for Weak Guards

- Duplicate constraints can impair the solving performance
- We have an idea on how to strengthen guards to address this redundancy
Conclusions for Weak Guards

- Duplicate constraints can impair the solving performance
- We have an idea on how to strengthen guards to address this redundancy
- We still need to implement/test/refine the algorithm.
Summary

- There is scope for optimisations in quantifications
Summary

- There is scope for optimisations in quantifications
- We can already provide some enhancement
Summary

- There is scope for optimisations in quantifications
- We can already provide some enhancement
- But there is still a lot to investigate!
Thank You.