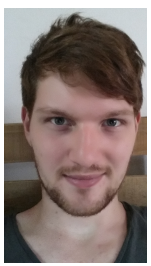
Introducing a new nonlinear expressions framework for SCIP

Benjamin Müller, **Felipe Serrano**, **Stefan Vigerske**, and **Fabian Wegscheider**



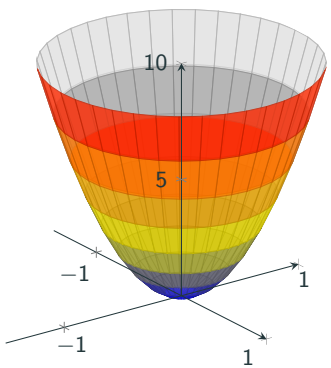



SCIP Workshop · March 7, 2018

Mixed-Integer Nonlinear Programming

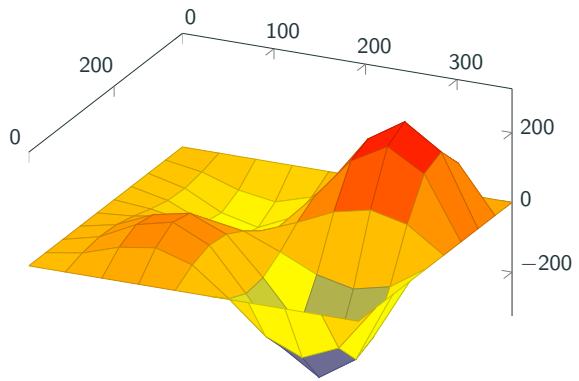
$$
\begin{aligned}
\min\ & c^\mathsf{T} x \\
\text{s.t. } & g_k(x) \leq 0 && \forall k \in [m] \\
& x_i \in \mathbb{Z} && \forall i \in \mathcal{I} \subseteq [n] \\
& x_i \in [\ell_i, u_i] && \forall i \in [n]
\end{aligned}
$$

The functions $g_k : [\ell, u] \to \mathbb{R}$ can be



convex

or



nonconvex

and are given in algebraic form.

SCIP source files that are (primarily) for MINLP (SCIP 5.0.1)

wc -l	<file>
17082	nlpi/expr.c
16718	scip/cons_quadratic.c
10028	scip/cons_nonlinear.c
8085	scip/cons_bivariate.c
7429	scip/cons_abspower.c
6368	scip/nlp.c
5540	scip/cons_soc.c
4174	scip/intervalarith.c
3817	scip/reader_pip.c
3680	scip/heur_undercover.c
3315	scip/prop_obbt.c
3158	nlpi/nlpioracle.c
2989	nlpi/nlpi_ipopt.cpp
2906	nlpi/nlpi_filtersqp.c
2892	scip/sepa_eccuts.c
2837	scip/reader_gms.c
2808	scip/heur_nlpdiving.c
2670	nlpi/exprinterpret_cppad.cpp
2588	scip/reader_osil.c
2545	scip/heur_subnlp.c
2459	nlpi/nlpi_worhp.c
2009	scip/presol_qpkktref.c
1757	nlpi/pub_expr.h
1225	nlpi/nlpi_all.c
1101	scip/heur_multistart.c
1085	scip/sepa_gauge.c

wc -l	<file>
926	scip/sepa_convexproj.c
853	nlpi/nlpi.c
841	scip/cons_quadratic.h
834	scip/nlp.h
769	scip/heur_mpec.c
758	scip/prop_nlobbt.c
697	scip/intervalarith.h
516	nlpi/type_nlpi.h
491	nlpi/nlpi.h
473	scip/cons_nonlinear.h
442	nlpi/nlpioracle.h
424	nlpi/intervalarithext.h
339	nlpi/nlpi_ipopt_dummy.c
314	nlpi/type_expr.h
237	scip/cons_soc.h
230	scip/pub_nlp.h
211	scip/cons_abspower.h
205	nlpi/exprinterpret_none.c
188	nlpi/struct_expr.h
184	nlpi/exprinterpret.h
181	scip/cons_bivariate.h
179	scip/struct_nlp.h
141	scip/heur_subnlp.h
112	nlpi/nlpi_ipopt.h
111	scip/prop_nlobbt.h
103	scip/presol_qpkktref.h

wc -l	<file>
100	scip/heur_mpec.h
96	scip/heur_multistart.h
92	scip/sepa_eccuts.h
91	scip/reader_pip.h
90	scip/sepa_gauge.h
88	nlpi/struct_nlpi.h
85	scip/sepa_convexproj.h
82	scip/reader_gms.h
78	scip/heur_undercover.h
71	nlpi/nlpi_worhp.h
69	scip/prop_obbt.h
56	nlpi/nlpi_filtersqp_dummy.c
53	scip/heur_nlpdiving.h
53	nlpi/nlpi_all.h
52	scip/reader_osil.h
52	nlpi/nlpi_worhp_dummy.c
52	nlpi/nlpi_filtersqp.h
48	nlpi/type_exprinterpret.h
38	scip/type_nlp.h
26	nlpi/intervalarithext.cpp

133396 total

```
[src]$ wc -l blockmemshell/* dijkstra/* lpi/* nlpi/* objscip/* scip/* symmetry/* tclique/* tpi/* main.c | tail
```

773866 total

SCIP source files that are (primarily) for MINLP (SCIP 5.0.1)

wc -l	<file>
17082	nlpi/expr.c
16718	scip/cons_quadratic.c
10028	scip/cons_nonlinear.c
8085	scip/cons_bivariate.c
7429	scip/cons_abspower.c
6368	scip/nlp.c
5540	scip/cons_soc.c
4174	scip/intervalarith.c
3817	scip/reader_pip.c
3680	scip/heur_undercover.c
3315	scip/prop_obbt.c
3158	nlpi/nlpioracle.c
2989	nlpi/nlpi_ipopt.cpp
2906	nlpi/nlpi_filtersqp.c
2892	scip/sepa_eccuts.c
2837	scip/reader_gms.c
2808	scip/heur_nlpdiving.c
2670	nlpi/exprinterpret_cppad.cpp
2588	scip/reader_osil.c
2545	scip/heur_subnlp.c
2459	nlpi/nlpi_worhp.c
2009	scip/presol_qpkktref.c
1757	nlpi/pub_expr.h
1225	nlpi/nlpi_all.c
1101	scip/heur_multistart.c
1085	scip/sepa_gauge.c

wc -l	<file>
926	scip/sepa_convexproj.c
853	nlpi/nlpi.c
841	scip/cons_quadratic.h
834	scip/nlp.h
769	scip/heur_mpec.c
758	scip/prop_nlobbt.c
697	scip/intervalarith.h
516	nlpi/type_nlpi.h
491	nlpi/nlpi.h
473	scip/cons_nonlinear.h
442	nlpi/nlpioracle.h
424	nlpi/intervalarithext.h
339	nlpi/nlpi_ipopt_dummy.c
314	nlpi/type_expr.h
237	scip/cons_soc.h
230	scip/pub_nlp.h
211	scip/cons_abspower.h
205	nlpi/exprinterpret_none.c
188	nlpi/struct_expr.h
184	nlpi/exprinterpret.h
181	scip/cons_bivariate.h
179	scip/struct_nlp.h
141	scip/heur_subnlp.h
112	nlpi/nlpi_ipopt.h
111	scip/prop_nlobbt.h
103	scip/presol_qpkktref.h

wc -l	<file>
100	scip/heur_mpec.h
96	scip/heur_multistart.h
92	scip/sepa_eccuts.h
91	scip/reader_pip.h
90	scip/sepa_gauge.h
88	nlpi/struct_nlpi.h
85	scip/sepa_convexproj.h
82	scip/reader_gms.h
78	scip/heur_undercover.h
71	nlpi/nlpi_worhp.h
69	scip/prop_obbt.h
56	nlpi/nlpi_filtersqp_dummy.c
53	scip/heur_nlpdiving.h
53	nlpi/nlpi_all.h
52	scip/reader_osil.h
52	nlpi/nlpi_worhp_dummy.c
52	nlpi/nlpi_filtersqp.h
48	nlpi/type_exprinterpret.h
38	scip/type_nlp.h
26	nlpi/intervalarithext.cpp

133396 total

```
[src]$ wc -l blockmemshell/* dijkstra/* lpi/* nlpi/* objscip/* scip/* symmetry/* tclique/* tpi/* main.c | tail
```

773866 total

The “classical” framework for (MI)NLP in SCIP

SCIP Constraint Handlers for NLP (in order of appearance)

Quadratic Constraints

$$\text{lhs} \leq \sum_{i,j=1}^{n} a_{i,j} x_i x_j + \sum_{i=1}^{n} b_i x_i \leq \text{rhs}$$

Second-Order Cone Constraints

$$\sqrt{\gamma + \sum_{i=1}^{n} (\alpha_i (x_i + \beta_i))^2} \leq \alpha_{n+1} (x_{n+1} + \beta_{n+1})$$

Absolute Power Constraints

$$\text{lhs} \leq \text{sign}(x + a)|x + a|^n + c\, z \leq \text{rhs}, \quad n \in \mathbb{R}_{\geq 1}$$

Nonlinear Constraints

$$\text{lhs} \leq \sum_{i=1}^{n} a_i x_i + \sum_{j=1}^{m} c_j f_j(x) \leq \text{rhs}$$

$f_j(\cdot)$ is given as expression tree

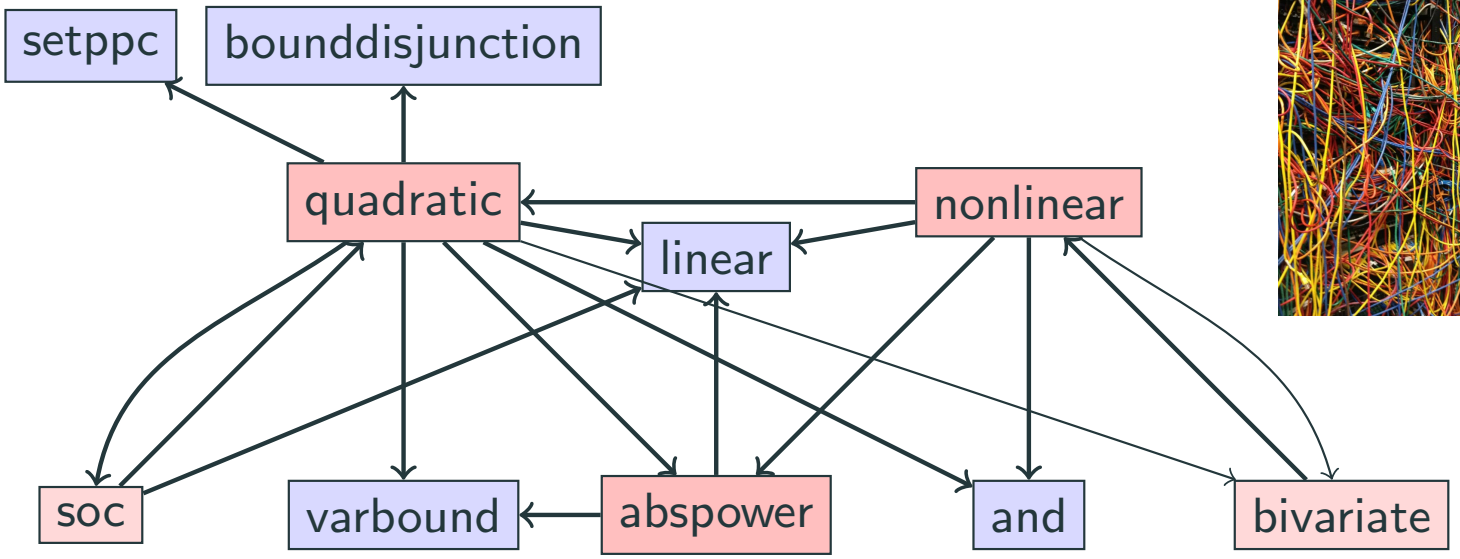
Bivariate Nonlinear Constraints

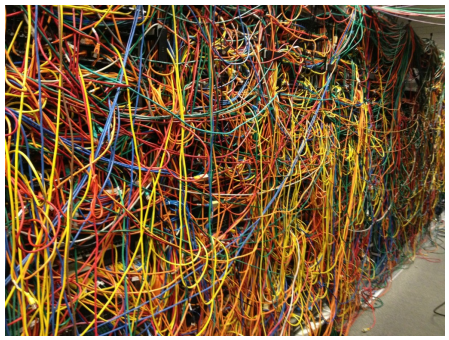
$$\text{lhs} \leq f(x, y) + c\, z \leq \text{rhs}$$

$f(\cdot)$ is given as expression tree using 2 variables

$f(\cdot, y)$ and $f(x, \cdot)$ are convex or concave

Constraint Conversion (Upgrades and Reformulations)





- Apart from the optional handlers for SOC and bivariate, nonlinearities are distinguished into
  - convex and concave (cons_nonlinear)
  - quadratic, esp. $x \cdot y$ (cons_quadratic)
  - odd power (x^3, x^5, $\text{sign}(x)|x|^n$) (cons_abspower)
- Separation (LP relaxation) is implemented for these 4 types

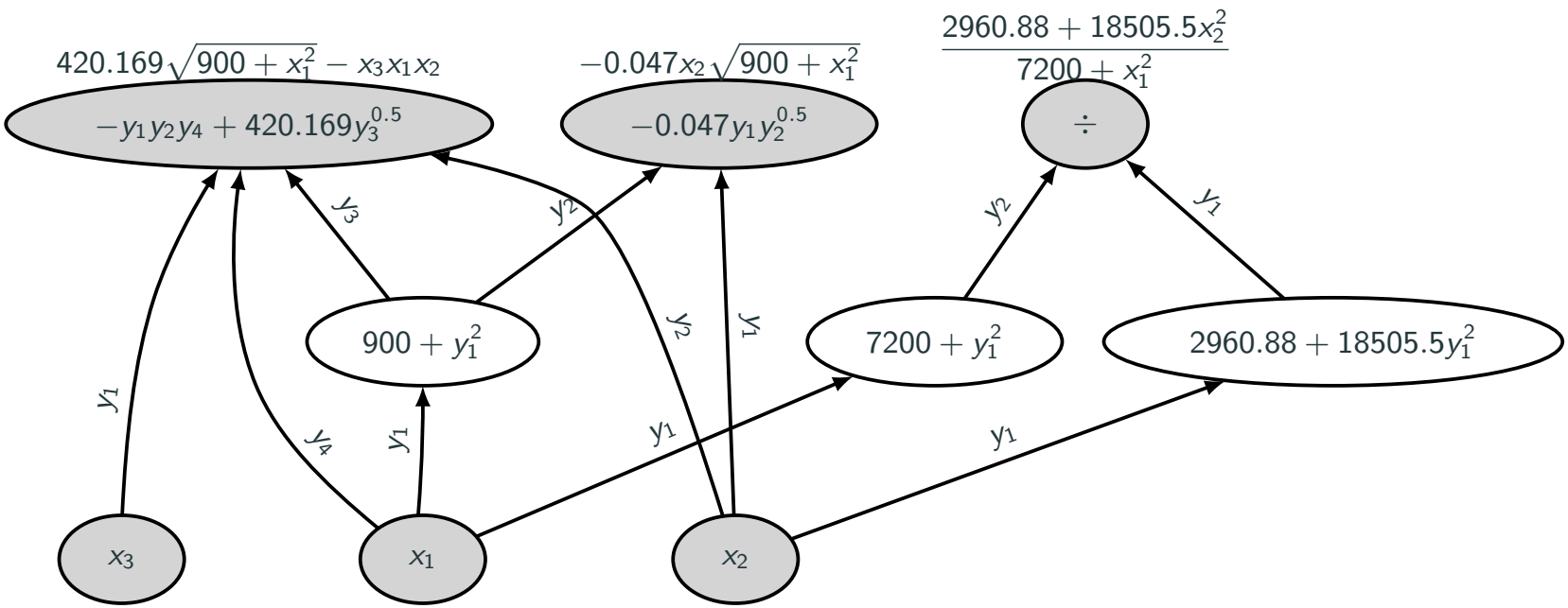
Expression trees and graphs

cons_nonlinear (lhs $\leq \sum_{i=1}^{n} a_i x_i + \sum_{j=1}^{m} c_j f_j(x) \leq$ rhs) stores the nonlinear functions f_j of *all* constraints in one expression graph (DAG).

For example (instance `nvs01`):

$$420.169\sqrt{900 + x_1^2} - x_3 x_1 x_2 = 0 \qquad \frac{2960.88 + 296088 \cdot 0.0625 x_2^2}{7200 + x_1^2} - x_3 \geq 0$$

$$x_{\text{obj}} - 0.047 x_2 \sqrt{900 + x_1^2} \geq 0$$

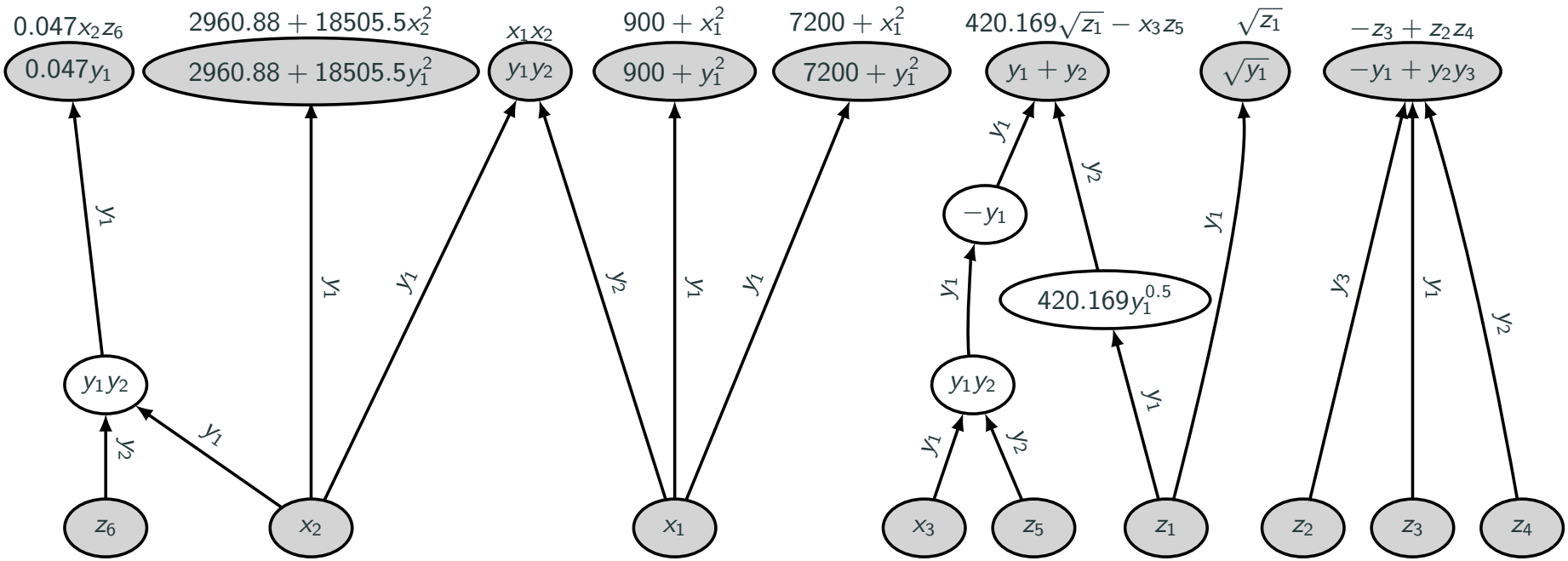


- some use of common subexpression

Reformulation in cons_nonlinear (CONSPRESOL)

Goal: Reformulate constraints such that only elementary cases (convex, concave, odd power, quadratic) remain.

$$900 + x_1^2 = z_1 \qquad 7200 + x_1^2 = z_4 \qquad 420.169\sqrt{z_1} - x_3 z_5 = 0$$
$$-z_3 + z_2 z_4 = 0 \qquad x_1 x_2 = z_5 \qquad z_2 - x_3 \geq 0$$
$$2960.88 + 18505.5 x_2^2 = z_3 \qquad \sqrt{z_1} = z_6 \qquad 0.047 x_2 z_6 \leq x_{\text{obj}}$$



- reformulates constraints by introducing new variables and new constraints
- other cons. handler can participate (SCIP_DECL_EXPRGRAPHNODEREFORM callback)

Expression operators

```
enum SCIP_ExprOp {
   /* Terminals (Leaves) */
   SCIP_EXPR_VARIDX    =  1,  /**< variable given by index (stored in data.idx) */
   SCIP_EXPR_CONST     =  2,  /**< constant (value stored in data.dbl) */
   SCIP_EXPR_PARAM     =  3,  /**< parameter = a constant that can be modified (should not be simplified away) */
   /* Simple Operands */
   SCIP_EXPR_PLUS      =  8,  /**< addition (2 operands) */
   SCIP_EXPR_MINUS     =  9,  /**< substraction (2 operands) */
   SCIP_EXPR_MUL       = 10,  /**< multiplication (2 operands) */
   SCIP_EXPR_DIV       = 11,  /**< division (2 operands) */
   SCIP_EXPR_SQUARE    = 12,  /**< square (1 operand) */
   SCIP_EXPR_SQRT      = 13,  /**< square root (1 operand) */
   SCIP_EXPR_REALPOWER = 14,  /**< power with real exponent (1 operand!, assumed to be nonnegative, exponent is stored in expression
   SCIP_EXPR_INTPOWER  = 15,  /**< power with integer exponent (1 operand!, exponent stored in expression data) */
   SCIP_EXPR_SIGNPOWER = 16,  /**< signed power (sign(x)|x|^a, 1 operand!, exponent is stored in expression data) */
   SCIP_EXPR_EXP       = 17,  /**< exponential (e^x, 1 operand) */
   SCIP_EXPR_LOG       = 18,  /**< natural logarithm (ln(x), 1 operand) */
   SCIP_EXPR_SIN       = 19,  /**< sinus (1 operand) */
   SCIP_EXPR_COS       = 20,  /**< cosinus (1 operand) */
   SCIP_EXPR_TAN       = 21,  /**< tangent (1 operand) */
   /* SCIP_EXPR_ERF       = 22, */  /**< gaussian error function (1 operand) */
   /* SCIP_EXPR_ERFI      = 23, */  /**< imaginary part of gaussian error function (1 operand) */
   SCIP_EXPR_MIN       = 24,  /**< minimum (2 operands) */
   SCIP_EXPR_MAX       = 25,  /**< maximum (2 operands) */
   SCIP_EXPR_ABS       = 26,  /**< absolute value (1 operand) */
   SCIP_EXPR_SIGN      = 27,  /**< sign of value (1 operand) */
   /* Complex Operands */
   SCIP_EXPR_SUM       = 64,  /**< summation sum_i=1^n op_i (n operands) */
   SCIP_EXPR_PRODUCT   = 65,  /**< product prod_i=1^n op_i (n operands) */
   SCIP_EXPR_LINEAR    = 66,  /**< linear term sum_i=1^n a_i op_i (n operands) */
   SCIP_EXPR_QUADRATIC = 67,  /**< quadratic term sum_i,j=1^n a_i,j op_i op_j (n operands) */
   SCIP_EXPR_POLYNOMIAL= 68,  /**< polynomial term sum_I a_Iops^I (I a multiindex, n operands) */
   SCIP_EXPR_USER      = 69,  /**< a user defined expression */
   SCIP_EXPR_LAST      = 70   /**< no expression, used for counting reasons */
};
```

So why rewriting a seemingly perfect piece of code?

So why rewriting a seemingly perfect piece of code?

Consider

$$\begin{aligned} \min\ & z \\ \text{s.t. } & \exp(\ln(1000) + 1 + x\,y) \leq z \\ & x^2 + y^2 \leq 2 \end{aligned}$$

An optimal solution:

$x = 1$

$y = -1$

$z = 1000$

So why rewriting a seemingly perfect piece of code?

Consider

$$\min z$$
$$\text{s.t. } \exp(\ln(1000) + 1 + x\,y) \leq z$$
$$x^2 + y^2 \leq 2$$

An optimal solution:

$x = 1$

$y = -1$

$z = 1000$

SCIP reports

```
SCIP Status        : problem is solved [optimal solution found]
Solving Time (sec) : 0.08
Solving Nodes      : 5
Primal Bound       : +9.99999656552062e+02 (3 solutions)
Dual Bound         : +9.99999656552062e+02
Gap                : 0.00 %
  [nonlinear] <e1>: exp((7.9077552789821368151 +1 (<x> * <y>)))-1<z>[C]  <= 0;
violation: right hand side is violated by 0.000673453314561812
best solution is not feasible in original problem

x                                    -1.00057454873626  (obj:0)
y                                    0.999425451364613  (obj:0)
z                                     999.999656552061  (obj:1)
```

SCIP log

$$\min z \quad \text{s.t. } \exp(\ln(1000) + 1 + x\,y) \leq z,\ x^2 + y^2 \leq 2$$

```
presolving (5 rounds: 5 fast, 1 medium, 1 exhaustive):
 0 deleted vars, 0 deleted constraints, 1 added constraints, 6 tightened bounds, 0 added holes, 0 changed sides, 0 changed coefficients
 0 implications, 0 cliques
presolved problem has 4 variables (0 bin, 0 int, 0 impl, 4 cont) and 3 constraints
      2 constraints of type <quadratic>
      1 constraints of type <nonlinear>
Presolving Time: 0.00

 time | node  | left  |LP iter|LP it/n| mem |mdpt |extbr|vars |cons |cols |rows |cuts |confs|strbr|  dualbound   | primalbound  |  gap
  0.0s|     1 |     0 |     1 |     - | 565k|   0 |   0 |   4 |   3 |   4 |   8 |   0 |   0 |   0 | 3.678794e+02 | 1.000000e+05 |  Larg

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

q 0.0s|     1 |     0 |     1 |     - | 569k|   0 |   0 |   4 |   3 |   4 |   8 |   0 |   0 |   0 | 3.678794e+02 | 9.999999e+02 | 171.8
...
  0.1s|     1 |     2 |    39 |     - | 629k|   0 |   2 |   4 |   3 |   4 |  21 |  14 |   0 |   0 | 4.367357e+02 | 9.999999e+02 | 128.9
* 0.1s|     2 |     1 |    56 |  42.0 | 651k|   1 |   0 |   4 |   3 |   4 |  18 |  28 |   0 |   0 | 4.367357e+02 | 9.999997e+02 | 128.9
  0.1s|     3 |     2 |    66 |  26.0 | 653k|   1 |   2 |   4 |   3 |   4 |  12 |  35 |   0 |   0 | 8.291193e+02 | 9.999997e+02 |  20.6
  0.1s|     4 |     1 |    66 |  17.3 | 653k|   2 |   0 |   4 |   3 |   0 |   0 |  35 |   0 |   0 | 8.291193e+02 | 9.999997e+02 |  20.6
  0.1s|     5 |     0 |    72 |  14.5 | 653k|   2 |   0 |   4 |   3 |   4 |  11 |  38 |   0 |   0 | 9.999997e+02 | 9.999997e+02 |   0.0

SCIP Status        : problem is solved [optimal solution found]
Solving Time (sec) : 0.08
Solving Nodes      : 5
Primal Bound       : +9.99999656552062e+02 (3 solutions)
Dual Bound         : +9.99999656552062e+02
Gap                : 0.00 %
  [nonlinear] <e1>: exp((7.9077552789821368151 +1 (<x> * <y>)))-1<z>[C]  <= 0;
violation: right hand side is violated by 0.000673453314561812 (scaled: 0.000673453314561812)
```

Reformulation takes apart $\exp(\ln(1000) + 1 + x\,y)$, thus SCIP actually solves

$$\begin{aligned} \min\ & z \\ \text{s.t.}\ & \exp(w) \leq z \\ & \ln(1000) + 1 + x\,y = w \\ & x^2 + y^2 \leq 2 \end{aligned}$$

Reformulation takes apart $\exp(\ln(1000) + 1 + x\,y)$, thus SCIP actually solves

		Violation	
min	z		
s.t.	$\exp(w) \leq z$	$0.4659 \cdot 10^{-6} \leq$	`numerics/feastol` ✓
	$\ln(1000) + 1 + x\,y = w$	$0.6731 \cdot 10^{-6} \leq$	`numerics/feastol` ✓
	$x^2 + y^2 \leq 2$	$0.6602 \cdot 10^{-6} \leq$	`numerics/feastol` ✓

Solution (`found by <relaxation>`):

x = -1.000574549
y = 0.999425451
z = 999.999656552
w = 6.907754936

Reformulation takes apart $\exp(\ln(1000) + 1 + x\,y)$, thus SCIP actually solves

		Violation	
min	z		
s.t.	$\exp(w) \leq z$	$0.4659 \cdot 10^{-6} \leq$	`numerics/feastol` ✓
	$\ln(1000) + 1 + x\,y = w$	$0.6731 \cdot 10^{-6} \leq$	`numerics/feastol` ✓
	$x^2 + y^2 \leq 2$	$0.6602 \cdot 10^{-6} \leq$	`numerics/feastol` ✓

Solution (`found by <relaxation>`):

x = -1.000574549
y = 0.999425451
z = 999.999656552
w= 6.907754936

⇒ Explicit reformulation of constraints ...

- ... looses the connection to the original problem.

display transproblem

Reformulation takes apart $\exp(\ln(1000) + 1 + x\,y)$, thus SCIP actually solves

$$\begin{aligned} \min\ & z && \text{Violation} \\ \text{s.t. } & \exp(w) \leq z && 0.4659 \cdot 10^{-6} \leq \texttt{numerics/feastol} \ \checkmark \\ & \ln(1000) + 1 + x\,y = w && 0.6731 \cdot 10^{-6} \leq \texttt{numerics/feastol} \ \checkmark \\ & x^2 + y^2 \leq 2 && 0.6602 \cdot 10^{-6} \leq \texttt{numerics/feastol} \ \checkmark \end{aligned}$$

Solution (`found by <relaxation>`):

```
x =   -1.000574549
y =    0.999425451
z = 999.999656552
w=     6.907754936
```

$\Rightarrow$ Explicit reformulation of constraints ...

- ... looses the connection to the original problem.
- ... looses distinction between original (i.e., transformed) and auxiliary variables.
  Thus, we may branch on auxiliary variables.
- ... prevents simultaneous exploitation of overlapping structures.

WIP: A new framework for NLP in SCIP

Main Ideas

Everything is an expression.

- **ONE** constraint handler: `cons_expr`
- represent all nonlinear constraints in one expression graph (DAG)

```
struct SCIP_ConsData
{
   SCIP_CONSEXPR_EXPR*   expr;   /**< expression that represents this constraint */
   SCIP_Real             lhs;    /**< left-hand side */
   SCIP_Real             rhs;    /**< right-hand side */

   ...                           /**< auxiliary data that does not define constraint */
};
```

- all algorithms (check, separation, propagation, etc.) work on the expression graph (no upgrades to specialized nonlinear constraints)

Main Ideas

Everything is an expression.

- **ONE** constraint handler: `cons_expr`
- represent all nonlinear constraints in one expression graph (DAG)

```
struct SCIP_ConsData
{
   SCIP_CONSEXPR_EXPR*   expr;   /**< expression that represents this constraint */
   SCIP_Real             lhs;    /**< left-hand side */
   SCIP_Real             rhs;    /**< right-hand side */

   ...                           /**< auxiliary data that does not define constraint */
};
```

- all algorithms (check, separation, propagation, etc.) work on the expression graph (no upgrades to specialized nonlinear constraints)
- avoid redundancy / ambiguity of expression types
  (classic framework: EXPR_PLUS, EXPR_SUM, EXPR_LINEAR, EXPR_QUADRATIC, ...)
- separate expression operators ($+$, $\times$) and high-level structures (quadratic, etc.)

Main Ideas

Everything is an expression.

- **ONE** constraint handler: `cons_expr`
- represent all nonlinear constraints in one expression graph (DAG)

```
struct SCIP_ConsData
{
   SCIP_CONSEXPR_EXPR*   expr;   /**< expression that represents this constraint */
   SCIP_Real             lhs;    /**< left-hand side */
   SCIP_Real             rhs;    /**< right-hand side */

   ...                           /**< auxiliary data that does not define constraint */
};
```

- all algorithms (check, separation, propagation, etc.) work on the expression graph (no upgrades to specialized nonlinear constraints)
- avoid redundancy / ambiguity of expression types
  (classic framework: EXPR_PLUS, EXPR_SUM, EXPR_LINEAR, EXPR_QUADRATIC, ...)
- separate expression operators ($+$, $\times$) and high-level structures (quadratic, etc.)

Do not reformulate constraints.

- introduce auxiliary variables for the relaxation only

The new expressions

Classic framework:

```
typedef enum  SCIP_ExprOp   SCIP_EXPROP;      /**< expression operand */
struct SCIP_Expr      /* and similar for struct SCIP_ExprGraphNode */
{
   SCIP_EXPROP           op;                  /**< operator of the node */
   SCIP_EXPROPDATA       data;                /**< operator data */
   int                   nchildren;           /**< number of children */
   SCIP_EXPR**           children;            /**< children nodes */
};
```

New framework:

```
struct SCIP_ConsExpr_Expr
{
   SCIP_CONSEXPR_EXPRHDLR* exprhdlr;      /**< expression type (as pointer to its handler) *
   SCIP_CONSEXPR_EXPRDATA* exprdata;      /**< expression data */
   int                     nchildren;     /**< number of children */
   SCIP_CONSEXPR_EXPR**    children;      /**< children expressions */
   ...
};
```

- expression handler behave like SCIP plugins (e.g., `SCIPincludeExprHdlr()`)

→ expressions require SCIP pointer! → NLPI require SCIP pointer!

- maybe more flexible regarding extensibility than using `SCIP_EXPR_USER`

Expression handler

Classic framework:

```
/** element in table of expression operands */
struct exprOpTableElement
{
   const char*           name;           /**< name of operand (used for printing) */
   int                   nargs;          /**< number of arguments (negative if not fixed) */
   SCIP_DECL_EXPREVAL    ((*eval));      /**< evaluation function */
   SCIP_DECL_EXPRINTEVAL ((*inteval));   /**< interval evaluation function */
   SCIP_DECL_EXPRCURV    ((*curv));      /**< curvature check function */
   SCIP_DECL_EXPRCOPYDATA((*copydata));  /**< expression data copy function, or NULL to only
   SCIP_DECL_EXPRFREEDATA((*freedata));  /**< expression data free function, or NULL if not
};
```

... and a number of `switch` statements in code (e.g., simplify, reformulate, propagation)

Expression handler

Classic framework:

```
/** element in table of expression operands */
struct exprOpTableElement
{
   const char*            name;           /**< name of operand (used for printing) */
   int                    nargs;          /**< number of arguments (negative if not fixed) */
   SCIP_DECL_EXPREVAL     ((*eval));      /**< evaluation function */
   SCIP_DECL_EXPRINTEVAL  ((*inteval));   /**< interval evaluation function */
   SCIP_DECL_EXPRCURV     ((*curv));      /**< curvature check function */
   SCIP_DECL_EXPRCOPYDATA((*copydata));   /**< expression data copy function, or NULL to only
   SCIP_DECL_EXPRFREEDATA((*freedata));   /**< expression data free function, or NULL if not
};
```

... and a number of `switch` statements in code (e.g., simplify, reformulate, propagation)

Reminder: ... *"nonlinearities are distinguished into*

- *convex and concave (cons_nonlinear)*
- *...*

"

Expression handler

New framework:

```
struct SCIP_ConsExpr_ExprHdlr
{
   char*                         name;     /**< expression handler name */
   char*                         desc;     /**< expression handler description (can be NULL)
   SCIP_CONSEXPR_EXPRHDLRDATA*   data;     /**< data of handler */

   SCIP_DECL_CONSEXPR_EXPREVAL((*eval));  /**< point evaluation callback (can never be NULL)
   SCIP_DECL_CONSEXPR_EXPRCOPYHDLR((*copyhdlr));  /**< handler copy callback (can be NULL) *
   SCIP_DECL_CONSEXPR_EXPRFREEHDLR((*freehdlr));  /**< handler free callback (can be NULL) *
   SCIP_DECL_CONSEXPR_EXPRCOPYDATA((*copydata));  /**< data copy callback, or NULL for expre
   SCIP_DECL_CONSEXPR_EXPRFREEDATA((*freedata));  /**< data free callback, or NULL for expre
   SCIP_DECL_CONSEXPR_EXPRSIMPLIFY((*simplify));  /**< simplify callback (can be NULL) */
   SCIP_DECL_CONSEXPR_EXPRCMP((*compare));        /**< compare callback (can be NULL) */
   SCIP_DECL_CONSEXPR_EXPRPRINT((*print));        /**< print callback (can be NULL) */
   SCIP_DECL_CONSEXPR_EXPRPARSE((*parse));        /**< parse callback (can be NULL) */
   SCIP_DECL_CONSEXPR_EXPRBWDIFF((*bwdiff));      /**< derivative evaluation callback (can b
   SCIP_DECL_CONSEXPR_EXPRINTEVAL((*inteval));    /**< interval evaluation callback (can be
   SCIP_DECL_CONSEXPR_EXPRINITSEPA((*initsepa));  /**< separation initialization callback (
   SCIP_DECL_CONSEXPR_EXPREXITSEPA((*exitsepa));  /**< separation deinitialization callback
   SCIP_DECL_CONSEXPR_EXPRSEPA((*sepa));          /**< separation callback (can be NULL) */
   SCIP_DECL_CONSEXPR_REVERSEPROP((*reverseprop));/**< reverse propagation callback (can be
   SCIP_DECL_CONSEXPR_EXPRHASH((*hash));          /**< hash callback (can be NULL) */
   SCIP_DECL_CONSEXPR_EXPRBRANCHSCORE((*brscore));/**< branching score callback (can be NULL
   SCIP_DECL_CONSEXPR_EXPRCURVATURE((*curvature));/**< curvature detection callback (can be
   SCIP_DECL_CONSEXPR_EXPRMONOTONICITY((*monotonicity)); /**< monotonicity detection callbac
   SCIP_DECL_CONSEXPR_EXPRINTEGRALITY((*integrality));   /**< integrality detection callback
};
```


Yes, they look similar to a constraint handler...

Expression types

classic (Σ: 27)	new (Σ: 11)
SCIP_EXPR_VARIDX	var
SCIP_EXPR_CONST	val
SCIP_EXPR_PARAM	–
SCIP_EXPR_PLUS	sum
SCIP_EXPR_MINUS	sum
SCIP_EXPR_MUL	prod
SCIP_EXPR_DIV	prod + pow
SCIP_EXPR_SQUARE	pow
SCIP_EXPR_SQRT	pow
SCIP_EXPR_REALPOWER	pow
SCIP_EXPR_INTPOWER	pow
SCIP_EXPR_SIGNPOWER	pow (todo)
SCIP_EXPR_EXP	exp
SCIP_EXPR_LOG	log
SCIP_EXPR_SIN	sin
SCIP_EXPR_COS	cos
SCIP_EXPR_TAN	–
SCIP_EXPR_MIN	– (todo)
SCIP_EXPR_MAX	– (todo)
SCIP_EXPR_ABS	abs
SCIP_EXPR_SIGN	–
SCIP_EXPR_SUM	sum
SCIP_EXPR_PRODUCT	prod
SCIP_EXPR_LINEAR	sum
SCIP_EXPR_QUADRATIC	sum + prod + pow
SCIP_EXPR_POLYNOMIAL	sum + prod + pow
SCIP_EXPR_USER	n/a
–	entropy: $x \mapsto \begin{cases} 0, & x = 0, \\ -x \log(x), & x > 0 \end{cases}$

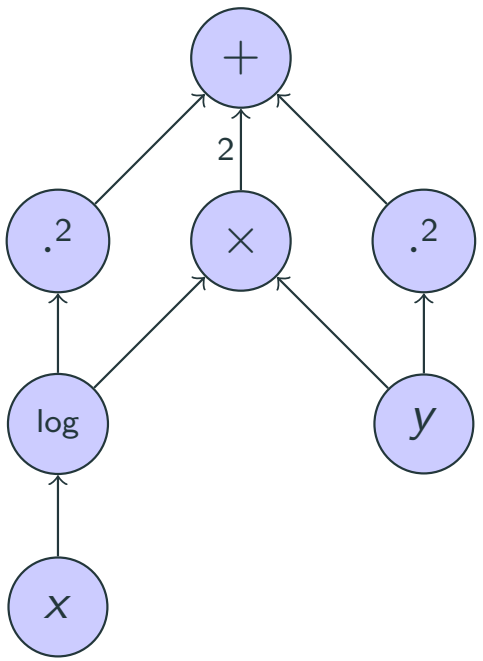
Enforcement

Constraint:

$$\log(x)^2 + 2\log(x)y + y^2 \leq 4$$

Used to

- check feasibility (CONSCHECK),
- presolve (CONSPRESOL),
- propagate domains (CONSPROP), ...



Enforcement

Constraint:

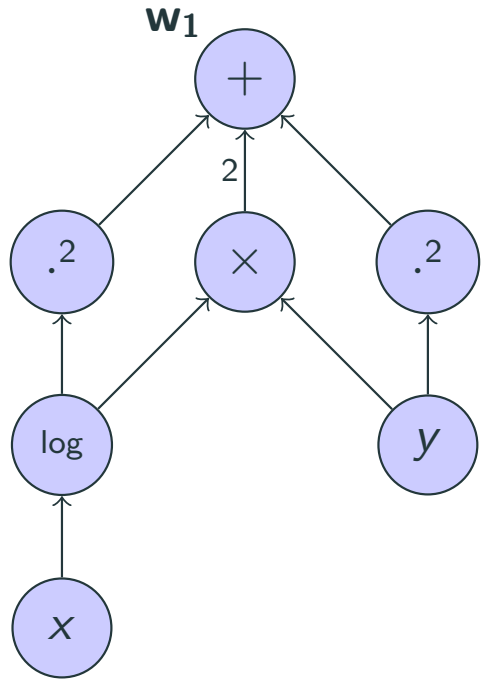
$$\log(x)^2 + 2\log(x)y + y^2 \leq 4$$

Used to

- check feasibility (CONSCHECK),
- presolve (CONSPRESOL),
- propagate domains (CONSPROP), ...

(Implicit) Reformulation:

$$w_1 \leq 4$$
$$\log(x)^2 + 2\log(x)y + y^2 = w_1$$



Enforcement

Constraint:

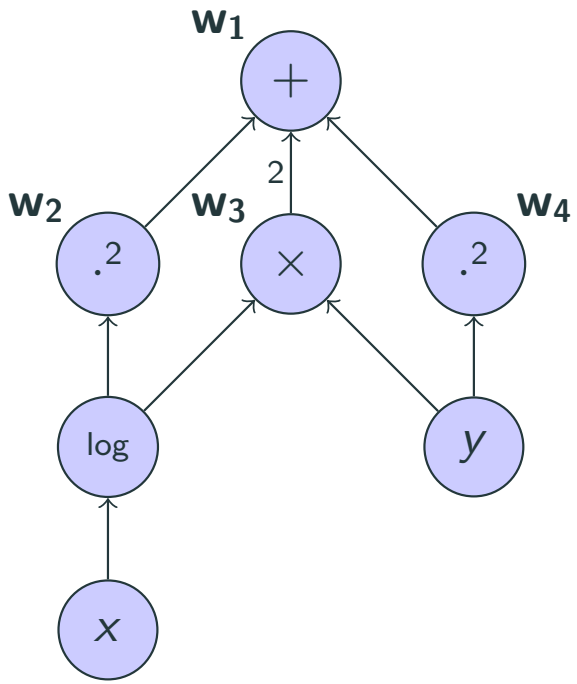
$$\log(x)^2 + 2\log(x)y + y^2 \leq 4$$

Used to

- check feasibility (CONSCHECK),
- presolve (CONSPRESOL),
- propagate domains (CONSPROP), ...

(Implicit) Reformulation:

$$\begin{aligned} w_1 &\leq 4 \\ w_2 + 2w_3 + w_4 &= w_1 \\ \log(x)^2 &= w_2 \\ \log(x)y &= w_3 \\ y^2 &= w_4 \end{aligned}$$



Enforcement

Constraint:

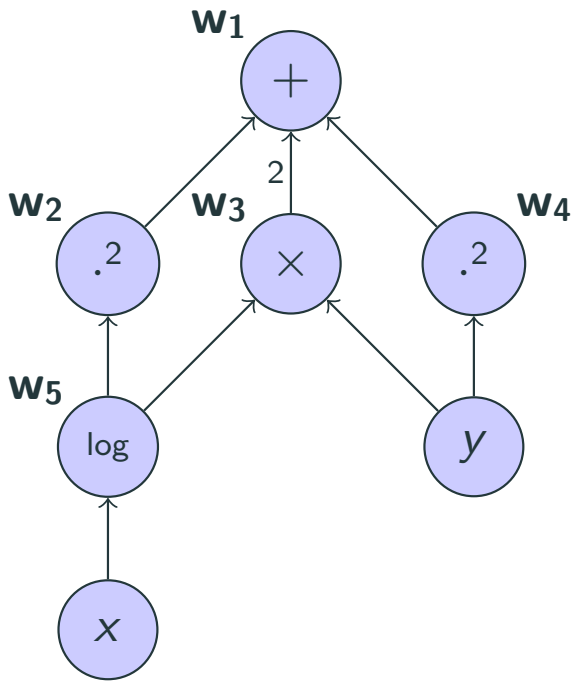
$$\log(x)^2 + 2\log(x)y + y^2 \leq 4$$

Used to

- check feasibility (CONSCHECK),
- presolve (CONSPRESOL),
- propagate domains (CONSPROP), ...

(Implicit) Reformulation:

$$\begin{aligned} w_1 &\leq 4 \\ w_2 + 2w_3 + w_4 &= w_1 \\ w_5^2 &= w_2 \\ w_5 y &= w_3 \\ y^2 &= w_4 \\ \log(x) &= w_5 \end{aligned}$$



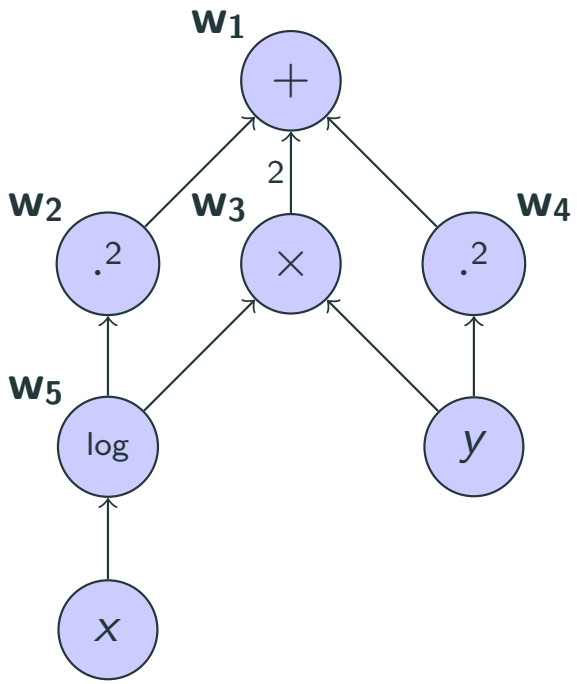
Enforcement

Constraint:

$$\log(x)^2 + 2\log(x)y + y^2 \leq 4$$

Used to

- check feasibility (CONSCHECK),
- presolve (CONSPRESOL),
- propagate domains (CONSPROP), ...



(Implicit) Reformulation:

$$\begin{aligned} w_1 &\leq 4 \\ w_2 + 2w_3 + w_4 &= w_1 \\ w_5^2 &= w_2 \\ w_5 y &= w_3 \\ y^2 &= w_4 \\ \log(x) &= w_5 \end{aligned}$$

Used to construct LP relaxation (CONSSEPALP).

Note: Auxiliary variables w_i are added during solve (CONSINITLP) without a pricer!

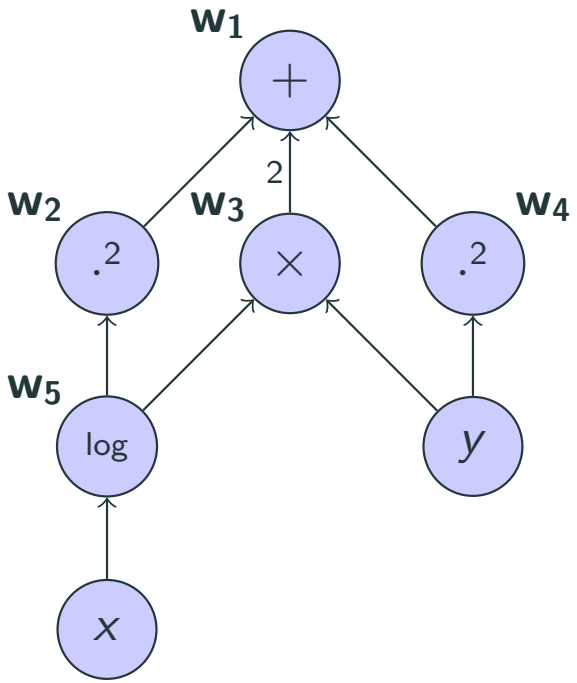
“Semi-auxiliaries”

Constraint:

$$\log(x)^2 + 2\log(x)y + y^2 \leq 4$$

(Implicit) Reformulation:

$$\begin{aligned} w_1 &\leq 4 \\ w_2 + 2w_3 + w_4 &= w_1 \\ w_5^2 &= w_2 \\ w_5 y &= w_3 \\ y^2 &= w_4 \\ \log(x) &= w_5 \end{aligned}$$



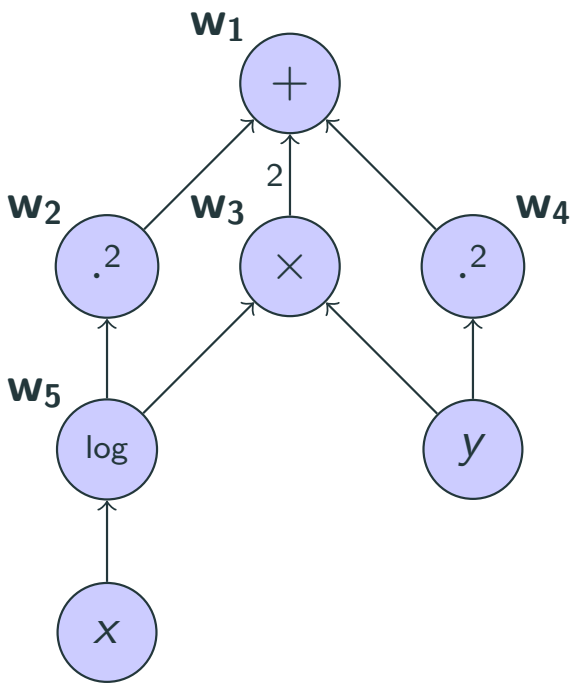
“Semi-auxiliaries”

Constraint:

$$\log(x)^2 + 2\log(x)y + y^2 \leq 4$$

(Implicit) Reformulation taking monotonicity (or locks) into account:

$$\begin{aligned} w_1 &\leq 4 \\ w_2 + 2w_3 + w_4 &\leq w_1 \\ w_5^2 &\leq w_2 \\ w_5 y &\leq w_3 \\ y^2 &\leq w_4 \\ \log(x) &= w_5 \end{aligned}$$



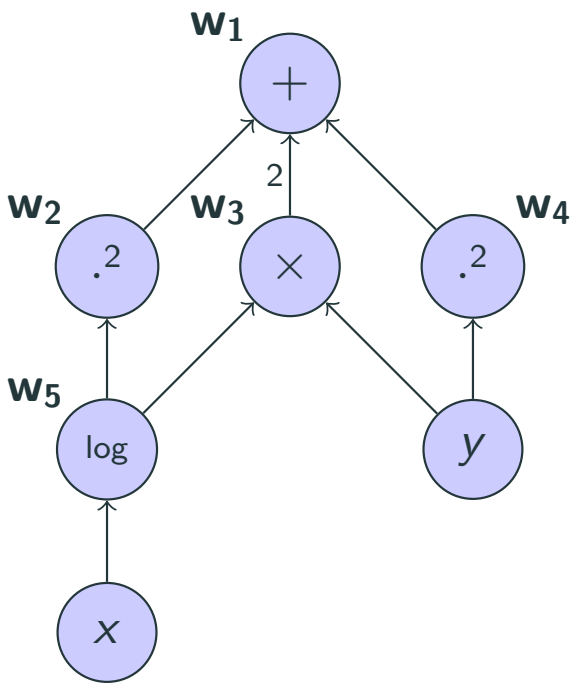
"Semi-auxiliaries"

Constraint:

$$\log(x)^2 + 2\log(x)y + y^2 \leq 4$$

(Implicit) Reformulation taking monotonicity (or locks) into account:

$$\begin{aligned} w_1 &\leq 4 \\ w_2 + 2w_3 + w_4 &\leq w_1 \\ w_5^2 &\leq w_2 \\ w_5 y &\leq w_3 \\ y^2 &\leq w_4 \\ \log(x) &= w_5 \end{aligned}$$



If $x \geq 1$ and $y \geq 0$, then $\log(x) = w_5$ can be relaxed to $\log(x) \leq w_5$.

If $x \leq 1$ and $y \leq 0$, then $\log(x) = w_5$ can be relaxed to $\log(x) \geq w_5$.

Exploiting structure

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$

Smarter reformulation:

- Recognize that $\log(x)^2 + 2\log(x)y + y^2$ is convex in $(\log(x), y)$.

Exploiting structure

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$

Smarter reformulation:

- Recognize that $\log(x)^2 + 2\log(x)y + y^2$ is convex in $(\log(x), y)$.

$\Rightarrow$ Introduce auxiliary variable for $\log(x)$ only.

$$w^2 + 2wy + y^2 \leq 4$$
$$\log(x) = w$$

Separate $w^2 + 2wy + y^2 \leq 4$ by linearization of $w^2 + 2wy + y^2$.

Exploiting structure

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$

Smarter reformulation:

- Recognize that $\log(x)^2 + 2\log(x)y + y^2$ is convex in $(\log(x), y)$.

$\Rightarrow$ Introduce auxiliary variable for $\log(x)$ only.

$$w^2 + 2wy + y^2 \leq 4$$
$$\log(x) = w$$

Separate $w^2 + 2wy + y^2 \leq 4$ by linearization of $w^2 + 2wy + y^2$.

Nonlinearity Handler (any suggestion for a better name?):

- Adds additional separation and propagation algorithms for structures that can be identified in the expression graph.
- Attached to nodes in expression graph, but does not *define* expressions or constraints.
- Rather similar to SCIP separators (SEPA) and propagators (PROP).
- Examples: quadratics, convex subexpressions, vertex-polyhedral

Nonlinearity Handler (Nlhdlr)

```
struct SCIP_ConsExpr_Nlhdlr
{
   char*                         name;       /**< nonlinearity handler name */
   char*                         desc;       /**< nonlinearity handler description (can be N
   SCIP_CONSEXPR_NLHDLRDATA*     data;       /**< data of handler */
   unsigned int                  priority;   /**< priority of nonlinearity handler */

   SCIP_DECL_CONSEXPR_NLHDLRFREEHDLRDATA((*freehdlrdata)); /**< callback to free data of han
   SCIP_DECL_CONSEXPR_NLHDLRFREEEXPRDATA((*freeexprdata)); /**< callback to free expression
   SCIP_DECL_CONSEXPR_NLHDLRCOPYHDLR((*copyhdlr));       /**< callback to copy nonlinear han
   SCIP_DECL_CONSEXPR_NLHDLRINIT((*init));               /**< initialization callback (can b
   SCIP_DECL_CONSEXPR_NLHDLREXIT((*exit));               /**< deinitialization callback (can

   SCIP_DECL_CONSEXPR_NLHDLRDETECT((*detect));           /**< structure detection callback *

   SCIP_DECL_CONSEXPR_NLHDLRINITSEPA((*initsepa));       /**< separation initialization call
   SCIP_DECL_CONSEXPR_NLHDLRSEPA((*sepa));               /**< separation callback (can be NU
   SCIP_DECL_CONSEXPR_NLHDLREXITSEPA((*exitsepa));       /**< separation deinitialization ca

   SCIP_DECL_CONSEXPR_NLHDLRINTEVAL((*inteval));         /**< interval evaluation callback (
   SCIP_DECL_CONSEXPR_NLHDLRREVERSEPROP((*reverseprop)); /**< reverse propagation callback (

   SCIP_DECL_CONSEXPR_NLHDLRBRANCHSCORE((*branchscore)); /**< branching scoring callback (ca
};
```

Nonlinearity Handler in Expression Graph

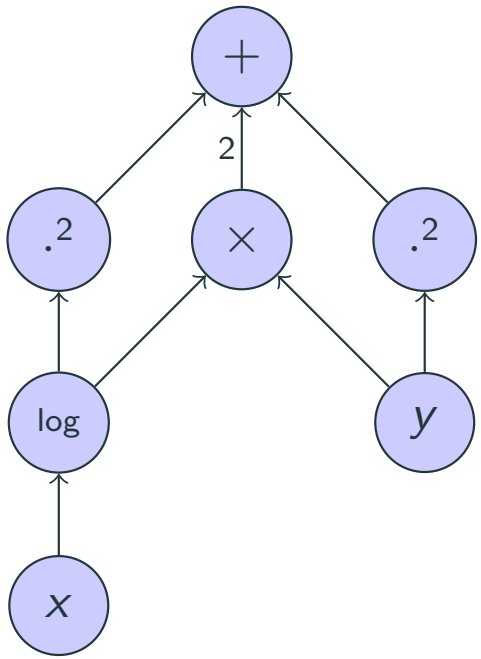
- Nodes in the expression graph can have one or several nlhdlrs attached.
- At beginning of solve (currently INITSOL), detection callbacks are run **only** for nodes that have auxiliary variable. Detection callback may add auxiliary variables.

.

Nonlinearity Handler in Expression Graph

- Nodes in the expression graph can have one or several nlhdlrs attached.
- At beginning of solve (currently INITSOL), detection callbacks are run **only** for nodes that have auxiliary variable. Detection callback may add auxiliary variables.

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$



.

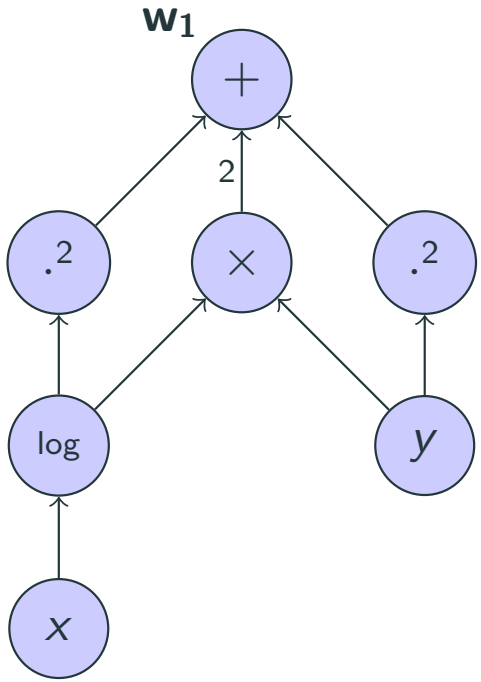
Nonlinearity Handler in Expression Graph

- Nodes in the expression graph can have one or several nlhdlrs attached.
- At beginning of solve (currently INITSOL), detection callbacks are run **only** for nodes that have auxiliary variable. Detection callback may add auxiliary variables.

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$

$$w_1 \leq 4$$

1. Add auxiliary variable w_1 for root.



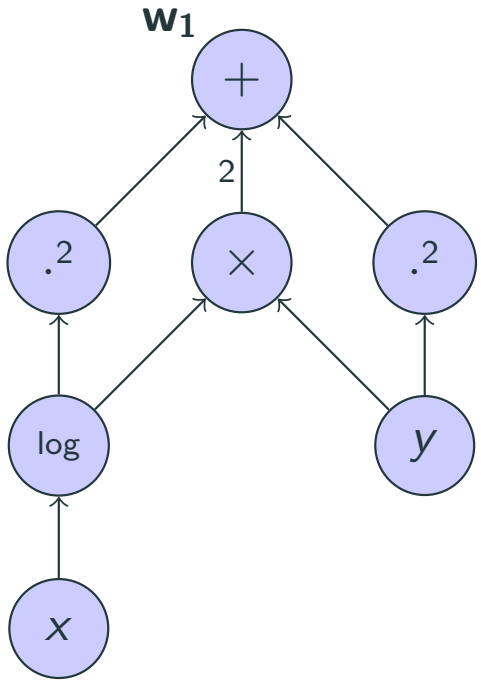
.

Nonlinearity Handler in Expression Graph

- Nodes in the expression graph can have one or several nlhdlrs attached.
- At beginning of solve (currently INITSOL), detection callbacks are run **only** for nodes that have auxiliary variable. Detection callback may add auxiliary variables.

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$

$$w_1 \leq 4$$



1. Add auxiliary variable w_1 for root.
2. Run `detect` of all nlhdlrs on $+$ node.

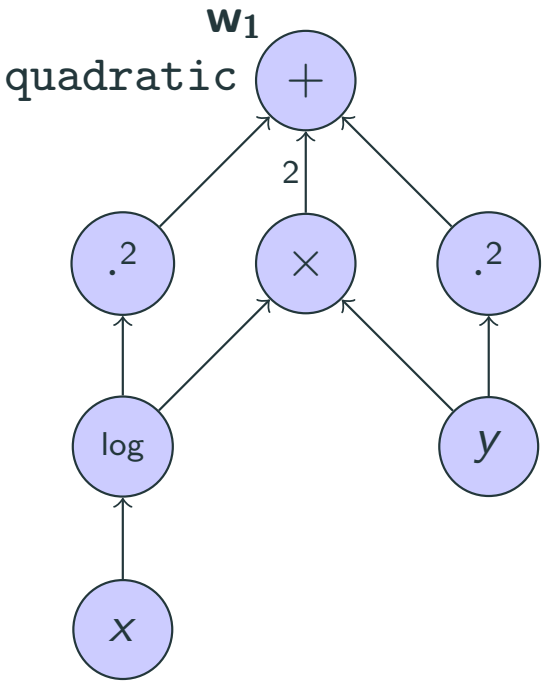
.

Nonlinearity Handler in Expression Graph

- Nodes in the expression graph can have one or several nlhdlrs attached.
- At beginning of solve (currently INITSOL), detection callbacks are run **only** for nodes that have auxiliary variable. Detection callback may add auxiliary variables.

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$

$$w_1 \leq 4$$



1. Add auxiliary variable w_1 for root.
2. Run `detect` of all nlhdlrs on $+$ node.
   - `nlhdlr_quadratic` detects a convex quadratic structure and signals success.

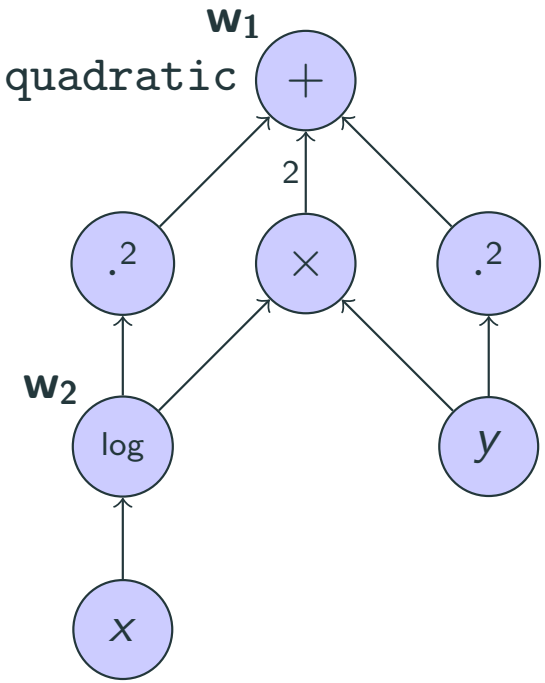
.

Nonlinearity Handler in Expression Graph

- Nodes in the expression graph can have one or several nlhdlrs attached.
- At beginning of solve (currently INITSOL), detection callbacks are run **only** for nodes that have auxiliary variable. Detection callback may add auxiliary variables.

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$

$$w_1 \leq 4$$
$$w_2^2 + 2w_2 y + y^2 \leq w_1 \quad [\texttt{nlhdlr\_quadratic}]$$



1. Add auxiliary variable w_1 for root.
2. Run `detect` of all nlhdlrs on $+$ node.
   - `nlhdlr_quadratic` detects a convex quadratic structure and signals success.
   - `nlhdlr_quadratic` adds an auxiliary variable w_2 for log node.

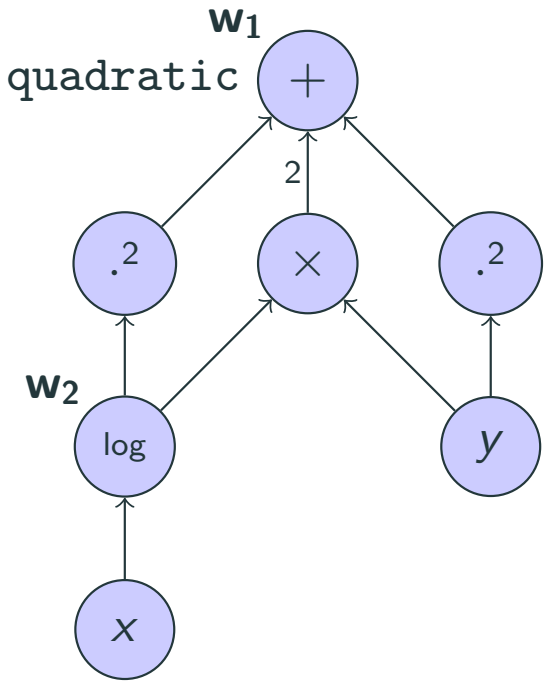
.

Nonlinearity Handler in Expression Graph

- Nodes in the expression graph can have one or several nlhdlrs attached.
- At beginning of solve (currently INITSOL), detection callbacks are run **only** for nodes that have auxiliary variable. Detection callback may add auxiliary variables.

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$

$$w_1 \leq 4$$
$$w_2^2 + 2w_2 y + y^2 \leq w_1 \quad [\texttt{nlhdlr\_quadratic}]$$



1. Add auxiliary variable w_1 for root.
2. Run `detect` of all nlhdlrs on $+$ node.
   - `nlhdlr_quadratic` detects a convex quadratic structure and signals success.
   - `nlhdlr_quadratic` adds an auxiliary variable w_2 for log node.
3. Run `detect` of all nlhdlrs on log node.

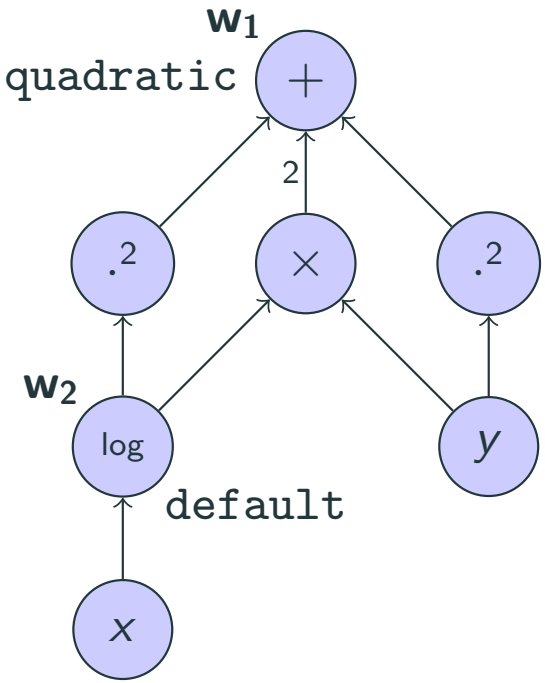
.

Nonlinearity Handler in Expression Graph

- Nodes in the expression graph can have one or several nlhdlrs attached.
- At beginning of solve (currently INITSOL), detection callbacks are run **only** for nodes that have auxiliary variable. Detection callback may add auxiliary variables.

Constraint: $\log(x)^2 + 2\log(x)y + y^2 \leq 4$

$$
\begin{aligned}
w_1 &\leq 4 \\
w_2^2 + 2w_2 y + y^2 &\leq w_1 \quad [\texttt{nlhdlr\_quadratic}] \\
\log(x) &= w_2 \quad [\texttt{nlhdlr\_default} \rightarrow \texttt{expr\_log}]
\end{aligned}
$$



1. Add auxiliary variable w_1 for root.
2. Run `detect` of all nlhdlrs on $+$ node.
   - `nlhdlr_quadratic` detects a convex quadratic structure and signals success.
   - `nlhdlr_quadratic` adds an auxiliary variable w_2 for log node.
3. Run `detect` of all nlhdlrs on log node.
   - `nlhdlr_default` (lowest priority) signals success.

`nlhdlr_default` forwards to separation and propagation of expression handler.

Done / WIP / TODO

What exists so far

Constraint Handler (`cons_expr`):

- upgrades from `cons_quadratic` and `cons_nonlinear`
- fundamental callbacks + parse, write, copy, separate, propagate, presolve
- canonicalization: simplify, common subexpressions
- add to NLP relaxation (conversion to classic expressions)

What exists so far

Constraint Handler (`cons_expr`):

- upgrades from `cons_quadratic` and `cons_nonlinear`
- fundamental callbacks + parse, write, copy, separate, propagate, presolve
- canonicalization: simplify, common subexpressions
- add to NLP relaxation (conversion to classic expressions)

Expression Handler:

- sum, var, value, abs, exp, log, cos, sin, entropy, pow, product
- product: convex hull for moderate number of factors (2–13)

What exists so far

Constraint Handler (`cons_expr`):

- upgrades from `cons_quadratic` and `cons_nonlinear`
- fundamental callbacks + parse, write, copy, separate, propagate, presolve
- canonicalization: simplify, common subexpressions
- add to NLP relaxation (conversion to classic expressions)

Expression Handler:

- sum, var, value, abs, exp, log, cos, sin, entropy, pow, product
- product: convex hull for moderate number of factors (2–13)

Nonlinarity Handler:

- default: wrap around expression handler
- quadratic: recognize and separate convex quadratic
- convex: recognize some simple general convexities, separate by linearization

What exists so far

Constraint Handler (`cons_expr`):

- upgrades from `cons_quadratic` and `cons_nonlinear`
- fundamental callbacks + parse, write, copy, separate, propagate, presolve
- canonicalization: simplify, common subexpressions
- add to NLP relaxation (conversion to classic expressions)

Expression Handler:

- sum, var, value, abs, exp, log, cos, sin, entropy, pow, product
- product: convex hull for moderate number of factors (2–13)

Nonlinearity Handler:

- default: wrap around expression handler
- quadratic: recognize and separate convex quadratic
- convex: recognize some simple general convexities, separate by linearization

Unittests:

- plenty

```
wc -l src/scip/*cons_expr*    total:  24052
wc -l tests/src/cons/expr/*   total:  10012
```

Performance

- 1367 instances from MINLPLib
- 1800s timelimit, 0.0001 gaplimit, no permutations, default seed

	# instances[1]	classic	new	classic → new
# instances handled[2]	1367	1311	1361	-1+51

[1] number of instances considered in this row (1367 = all; 1310 = all that both frameworks can handle (no sin/cos/min/max/erf); 634 = instances solved by both frameworks

[2] handled = only supported expression operators

[4] number of instances with $\geq$ 10% increase/decrease in time

Performance

- 1367 instances from MINLPLib
- 1800s timelimit, 0.0001 gaplimit, no permutations, default seed

	# instances[1]	classic	new	classic → new
# instances handled[2]	1367	1311	1361	-1+51
# fail/abort/no-result[3]	1310	28	35	-21+28
# sol. infeas. $> 10^{-5}$	1310	39	2	-39+ 2

[1] number of instances considered in this row (1367 = all; 1310 = all that both frameworks can handle (no sin/cos/min/max/erf); 634 = instances solved by both frameworks
[2] handled = only supported expression operators
[3] no-result = run did not return from cluster
[4] number of instances with $\geq$ 10% increase/decrease in time

Performance

- 1367 instances from MINLPLib
- 1800s timelimit, 0.0001 gaplimit, no permutations, default seed

	# instances[1]	classic	new	classic → new
# instances handled[2]	1367	1311	1361	-1+51
# fail/abort/no-result[3]	1310	28	35	-21+28
# sol. infeas. $> 10^{-5}$	1310	39	2	-39+ 2
# solved	1310	737	704	-102+69
time (sh.geom.mean) [s]	634	8.3	13.0	-233+95[4]

[1] number of instances considered in this row (1367 = all; 1310 = all that both frameworks can handle (no sin/cos/min/max/erf); 634 = instances solved by both frameworks

[2] handled = only supported expression operators

[3] no-result = run did not return from cluster

[4] number of instances with $\geq$ 10% increase/decrease in time

Open merge-requests (WIP)

Propagation of quadratics:

- stronger propagation for quadratic terms by reducing interval overestimation
- essentially as in `cons_quadratic` for now

Open merge-requests (WIP)

Propagation of quadratics:

- stronger propagation for quadratic terms by reducing interval overestimation
- essentially as in `cons_quadratic` for now

Separation of bilinear terms:

- convex hull and propagation of $x \cdot y$ over polytopes
- recall previous talk

Open merge-requests (WIP)

Propagation of quadratics:

- stronger propagation for quadratic terms by reducing interval overestimation
- essentially as in `cons_quadratic` for now

Separation of bilinear terms:

- convex hull and propagation of $x \cdot y$ over polytopes
- recall previous talk

Reformulation-Linearization Technique (RLT) [Adams & Sherali]:

- stronger relaxations for problems with quadratic terms
- Example:
  - Assume $x\,y$ and x^2 appear somewhere in the problem.
  - Assume auxiliary variables are added: $w_{xy} = xy$, $w_{xx} = x^2$.
  - Assume linear constraint $\alpha x + \beta y = \gamma$.

Open merge-requests (WIP)

Propagation of quadratics:

- stronger propagation for quadratic terms by reducing interval overestimation
- essentially as in `cons_quadratic` for now

Separation of bilinear terms:

- convex hull and propagation of $x \cdot y$ over polytopes
- recall previous talk

Reformulation-Linearization Technique (RLT) [Adams & Sherali]:

- stronger relaxations for problems with quadratic terms
- Example:
  - Assume $x\,y$ and x^2 appear somewhere in the problem.
  - Assume auxiliary variables are added: $w_{xy} = xy$, $w_{xx} = x^2$.
  - Assume linear constraint $\alpha x + \beta y = \gamma$.
  - Multiply with variable: $(\alpha x + \beta y)x = \gamma x \Rightarrow \alpha w_{xx} + \beta w_{xy} - \gamma x = 0$.

Open merge-requests (WIP)

Propagation of quadratics:

- stronger propagation for quadratic terms by reducing interval overestimation
- essentially as in `cons_quadratic` for now

Separation of bilinear terms:

- convex hull and propagation of $x \cdot y$ over polytopes
- recall previous talk

Reformulation-Linearization Technique (RLT) [Adams & Sherali]:

- stronger relaxations for problems with quadratic terms
- Example:
  - Assume $x\,y$ and x^2 appear somewhere in the problem.
  - Assume auxiliary variables are added: $w_{xy} = xy$, $w_{xx} = x^2$.
  - Assume linear constraint $\alpha x + \beta y = \gamma$.
  - Multiply with variable: $(\alpha x + \beta y)x = \gamma x \Rightarrow \alpha w_{xx} + \beta w_{xy} - \gamma x = 0$.
  - Similar for products of linear inequalities with variable bounds.

more TODO for v1 (SCIP 7 ???)

- `expr_pow`: separation for exponent $!= 2$
- `expr_pow`: support for absolute power
- `nlhdlr_convex`: better detection (e.g., xe^x)
- `nlhdlr_convex`: convexity in auxiliary variables

  $(f(g(x), y),\ f(\cdot,\cdot)\ \text{convex} \Rightarrow f(w, y),\ w = g(x))$
- `nlhdlr_convex` (?): separation for concave functions (=vertex-polyhedral)
- `nlhdlr_soc`: handling of second-order cone constraints (!= expressions)
- split-up sums
- nonlinearity handler during presolve (for propagation)
- replace classic framework with new framework: adapt readers, NLP, NLPIs, primal heuristics, separators, propagators
- performance tuning and debugging

End.