E-matching for Fun and Profit

Michał Moskal, Jakub Łopuszański, Joseph R. Kiniry

University of Wrocław, Poland
University College, Dublin, Ireland

SMT Workshop, Berlin
July 1st, A.D. 0x7D7

Instantiation

Let's take the formula:

$$P(f(42)) \land \forall x. (P(f(x)) \Rightarrow x < 0)$$

Instantiation

Let's take the formula:

$$P(f(42)) \wedge \forall x. \, (P(f(x)) \Rightarrow x < 0)$$

Once the solver figures out the implication:

$$(\forall x. \, (P(f(x)) \Rightarrow x < 0)) \Rightarrow (P(f(42)) \Rightarrow 42 < 0)$$

Instantiation

Let's take the formula:

$$P(f(42)) \wedge \forall x. (P(f(x)) \Rightarrow x < 0)$$

Once the solver figures out the implication:

$$(\forall x. (P(f(x)) \Rightarrow x < 0)) \Rightarrow (P(f(42)) \Rightarrow 42 < 0)$$

it can deduce the query to be unsatisfiable using only ground reasoning.

Instantiation

Let's take the formula:

$$P(f(42)) \wedge \forall x. (P(f(x)) \Rightarrow x < 0)$$

Once the solver figures out the implication:

$$(\forall x. (P(f(x)) \Rightarrow x < 0)) \Rightarrow (P(f(42)) \Rightarrow 42 < 0)$$

it can deduce the query to be unsatisfiable using only ground reasoning.

How do we figure the correct instantiation out?

Triggers (an idea borrowed from Simplify)

- triggers are subterms present in the quantified formula, which can be automatically generated, or come with the input
- we only consider instances, for which the trigger after substitution is present somewhere in the model returned by the SAT solver

Triggers (an idea borrowed from Simplify)

- triggers are subterms present in the quantified formula, which can be automatically generated, or come with the input
- we only consider instances, for which the trigger after substitution is present somewhere in the model returned by the SAT solver
- for example if we have a formula

$$\psi \equiv \forall x, y.\, F(x, y) \Rightarrow g(x) = h(y)$$

for which $F(x, y)$ is a trigger and the current model returned by the SAT solver is:

$$\psi \wedge F(1, c) \wedge g(7) = h(c)$$

we only consider instance where $x \to 1, y \to c$, because $F(1, c)$ is present in the monome.

E–matching

- this picture with the triggers is slightly more involved, because we are interested in the trigger being present in the monome *up to equivalence*
- for example trigger $F(x, c)$ does not syntactically match in $F(1, d) \ \wedge \ c = d$, but we would like it to

E–matching: definition

Input

- a finite set $\mathcal{A}$ of *active* ground terms,
- a relation $\cong_g \subseteq \mathcal{A} \times \mathcal{A}$,
- a finite set of non-variable, non-constant *triggers* $p_1, \ldots, p_n$.

Definitions

- let $\cong \subseteq \mathcal{T} \times \mathcal{T}$ be the smallest congruence relation containing $\cong_g$,
- let $\mathtt{root}(t)$ denote a canonical representative of equivalence class containing t.

The solution to the E-matching problem is the set:

$$T = \left\{ \sigma \;\middle|\; \begin{array}{l} \exists t_1, \ldots, t_n \in \mathcal{A}.\, \sigma(p_1) \cong t_1 \wedge \cdots \wedge \sigma(p_n) \cong t_n, \\ \forall x \in \mathcal{V}.\, \sigma(x) = \mathtt{root}(\sigma(x)) \end{array} \right\}$$

E–matching: complexity

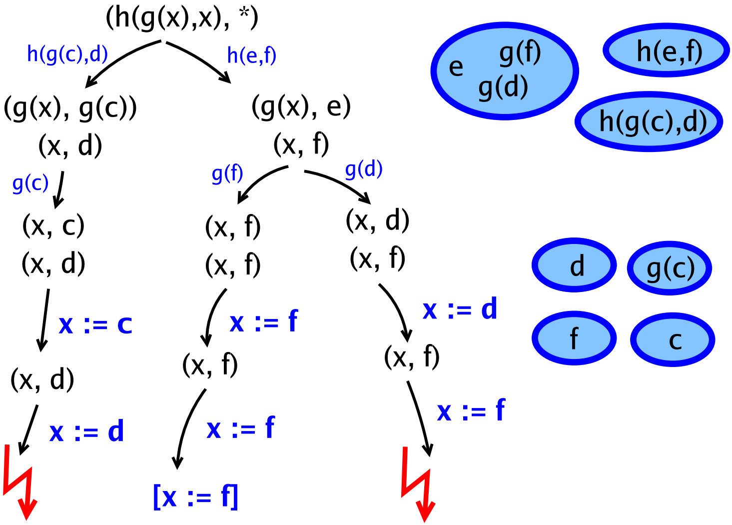
Michał Moskal, Jakub Łopuszański, Joseph R. Kiniry

- the problem of checking if $T \neq \emptyset$ for a fixed $\mathcal{A}$ and $\cong_g$ is NP-hard
- there can be exponential number of instances of a trigger

E–matching: complexity

- the problem of checking if $T \neq \emptyset$ for a fixed $\mathcal{A}$ and $\cong_g$ is NP-hard
- there can be exponential number of instances of a trigger
- the practical problem is, however, that there are often millions of matching problems to solve during solving of a single SMT query

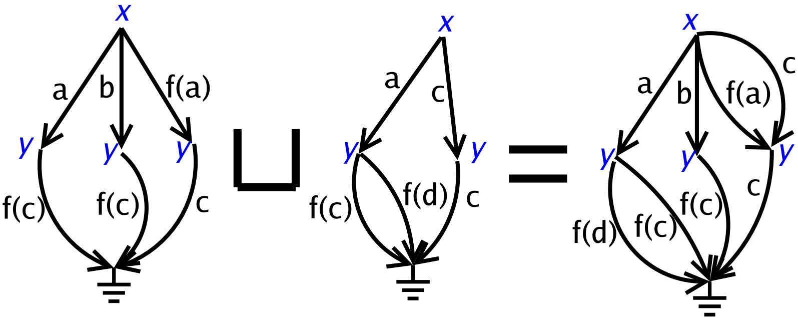
Simplify's matcher example



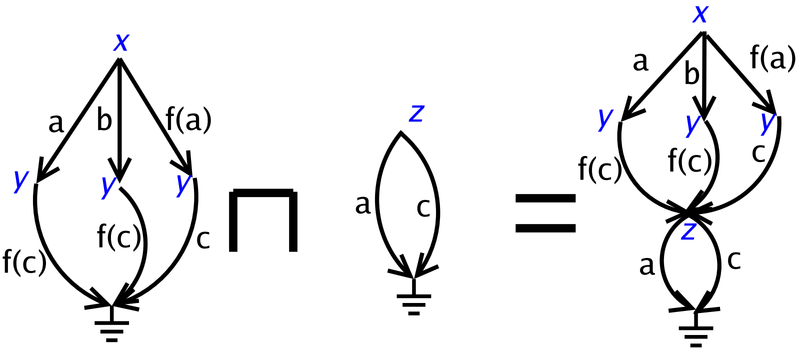
Simplify's algorithm

$$
\begin{array}{l}
\textbf{fun } \mathit{simplify\_match}([p_1, \ldots, p_n]) \\
\quad R := \emptyset \\
\quad \textbf{proc } \mathit{match}(\sigma, j) \\
\qquad \textbf{if } j = \mathit{nil} \textbf{ then } R := R \cup \{\sigma\} \\
\qquad \textbf{else case } \mathit{hd}(j) \textbf{ of} \\
\qquad\quad (c, t) \Rightarrow \qquad /\!*\ 1 *\!/ \\
\qquad\qquad \textbf{if } c \cong t \textbf{ then } \mathit{match}(\sigma, \mathit{tl}(j)) \\
\qquad\qquad \textbf{else skip} \\
\qquad\quad (x, t) \Rightarrow \qquad /\!*\ 2 *\!/ \\
\qquad\qquad \textbf{if } \sigma(x) = x \textbf{ then } \mathit{match}(\sigma[x := \mathtt{root}(t)], \mathit{tl}(j)) \\
\qquad\qquad \textbf{else if } \sigma(x) = \mathtt{root}(t) \textbf{ then } \mathit{match}(\sigma, \mathit{tl}(j)) \\
\qquad\qquad \textbf{else skip} \\
\qquad\quad (f(p_1, \ldots, p_n), t) \Rightarrow \qquad /\!*\ 3 *\!/ \\
\qquad\qquad \textbf{foreach } f(t_1, \ldots, t_n) \textbf{ in } \mathcal{A} \textbf{ do} \\
\qquad\qquad\quad \textbf{if } t = * \vee \mathtt{root}(f(t_1, \ldots, t_n)) = t \textbf{ then} \\
\qquad\qquad\qquad \mathit{match}(\sigma, (p_1, \mathtt{root}(t_1)) :: \cdots :: (p_n, \mathtt{root}(t_n)) :: \mathit{tl}(j)) \\
\quad \mathit{match}([], (p_1, *) :: \cdots :: (p_n, *) :: \mathit{nil}) \qquad /\!*\ 4 *\!/ \\
\quad \textbf{return } R
\end{array}
$$

S-tree sum

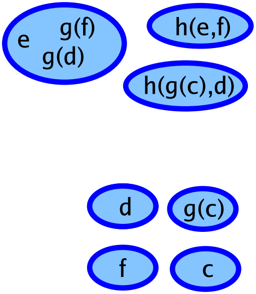


S-tree merge



Subtrigger matcher example, part I

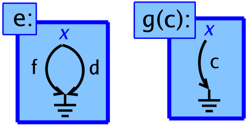
h(g(x),x) ->

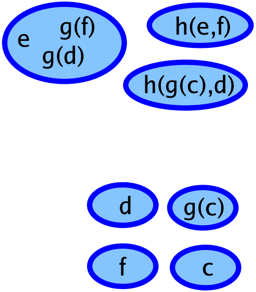


Subtrigger matcher example, part II

h(g(x),x) ->

g(x) ->

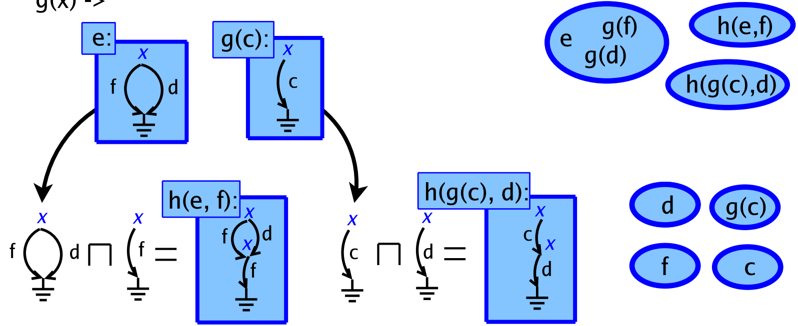




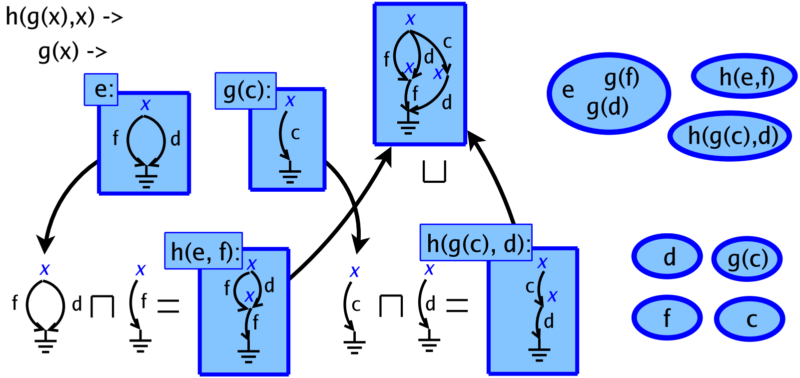
Subtrigger matcher example, part III

h(g(x),x) ->

g(x) ->



Subtrigger matcher example, part IV



Subtrigger matcher

fun *fetch*(S, t, p)
    if $S = \top$ **then return** $\{[p := \mathtt{root}(t)]\}$
    else if $S = \times \wedge t \cong p$ **then return** $\{[]\}$
    else if $S = \times$ **then return** $\emptyset$
    else return $S(\mathtt{root}(t))$
fun *match*(p)
    case p **of**
        $x \Rightarrow$ **return** $\top$
        $c \Rightarrow$ **return** $\times$
        $f(p_1, \ldots, p_n) \Rightarrow$
            foreach i **in** $1 \ldots n$ **do** $S_i = match(p_i)$ $/* \; 1 \; */$
            if $\exists i.\, S_i = \bot$ **then return** $\bot$ $/* \; 2 \; */$
            if $\forall i.\, S_i = \times$ **then return** $\times$ $/* \; 3 \; */$
            $S := \{t \mapsto \emptyset \mid t \in \mathcal{A}\}$
            foreach $f(t_1, \ldots, t_n)$ **in** $\mathcal{A}$ **do** $/* \; 4 \; */$
                $t := \mathtt{root}(f(t_1, \ldots, t_n))$
                $S := S[t \mapsto S(t) \sqcup (fetch(S_1, t_1, p_1) \sqcap \cdots \sqcap fetch(S_n, t_n, p_n))]$
          if $\forall t.\, S(t) = \bot$ **then return** $\bot$
          else return S

Subtrigger matcher, cont.

fun $\mathit{topmatch}(p)$ /* 5 */
 $S := \mathit{match}(p)$
 return $\bigsqcup_{t \in \mathcal{A}} S(t)$
fun $\mathit{subtrigger\_match}([p_1, \ldots, p_n])$
 return $\mathit{topmatch}(p_1) \sqcap \cdots \sqcap \mathit{topmatch}(p_n)$

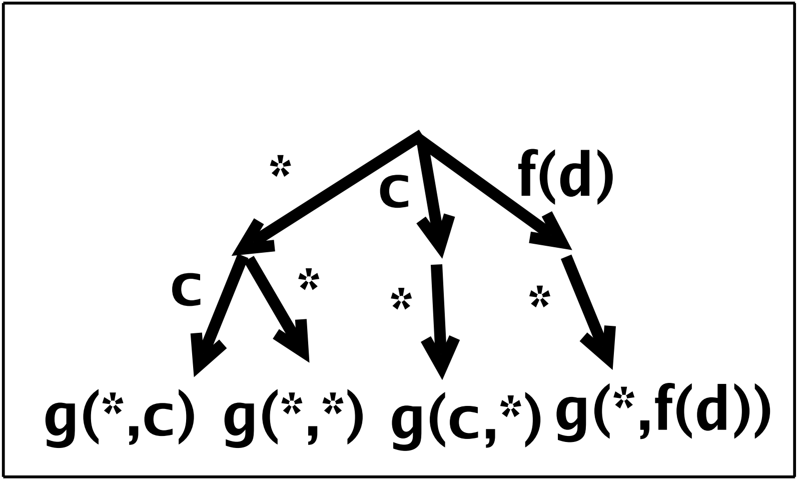
Even simpler triggers

- even this algorithm was taking up a lot of time, mainly because the loop over all terms with given head is performed for each trigger

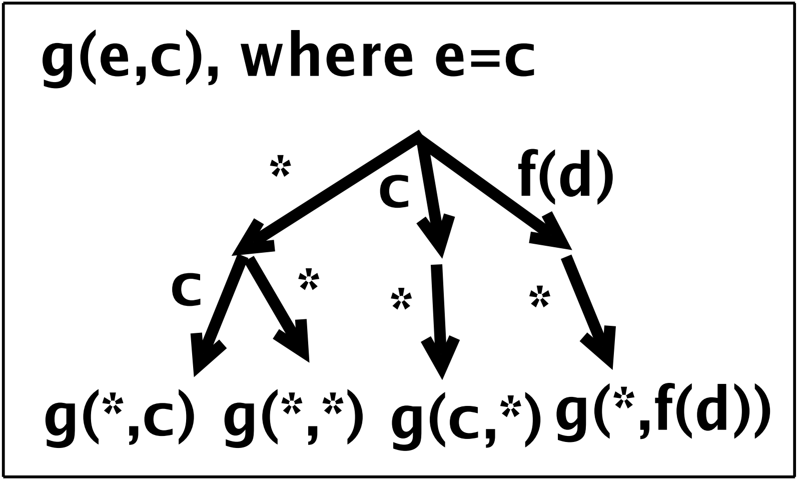
Even simpler triggers

- even this algorithm was taking up a lot of time, mainly because the loop over all terms with given head is performed for each trigger
- but it seems that a lot (most?) of the triggers are even simpler – they have variables only at depth one: $f(X, Y)$, $f(c, X)$, $f(X, c, Y, g(g(d)), Z)$
- this means one can put all such *flat* triggers with head f in an indexing tree and match them all at once during one loop over terms with head f

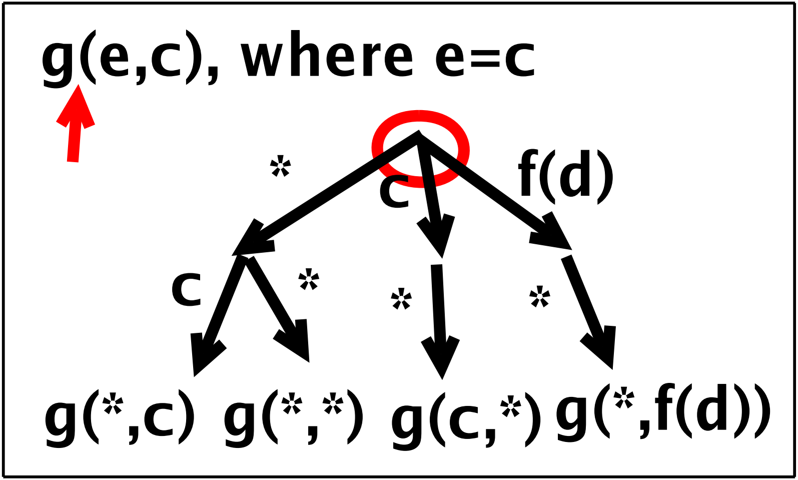
Flat trigger index



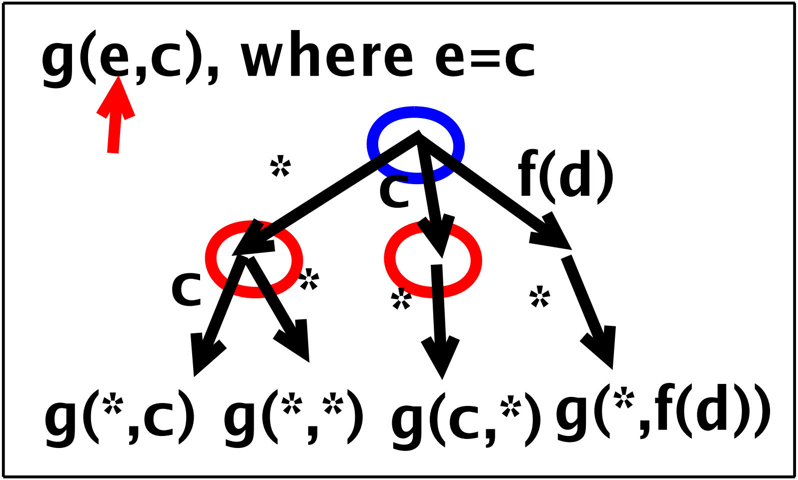
Let's match something!



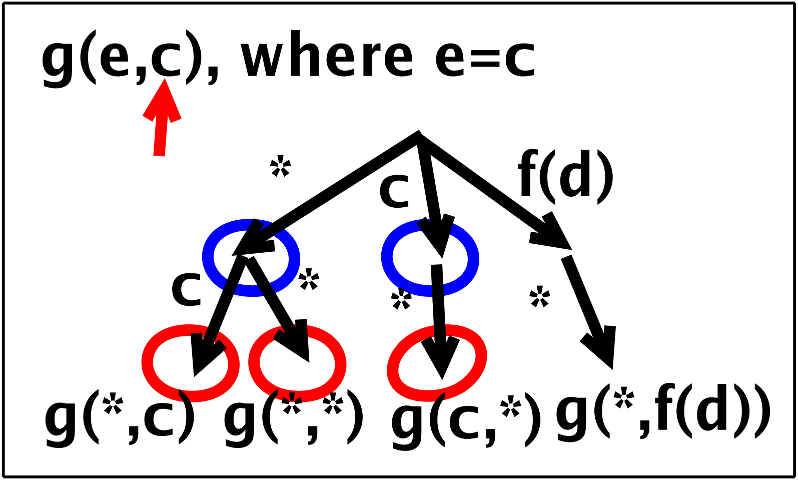
At the root



Child #1



Child #2



Memoization

- maximal sharing in terms and s-trees
- $\sqcup$, $\sqcap$ memoize results
- s-tree subtraction to remove previously returned results
- mapping of all the variables to $*$, to maximize sharing in subtrigger matcher
- mixed effects with *mod-time*

Explanation

- consider trigger: $f(g_1(x_1), \ldots, g_n(x_n))$
- $g_1(x_1)$ through $g_{n-1}(x_{n-1})$ return two matches each
- $g_n(x_n)$ does not match anything

Explanation

- consider trigger: $f(g_1(x_1), \ldots, g_n(x_n))$
- $g_1(x_1)$ through $g_{n-1}(x_{n-1})$ return two matches each
- $g_n(x_n)$ does not match anything
- Simplify's matcher: $O(2^n)$ steps, subtrigger matcher: $O(n)$ steps

Explanation

- consider trigger: $f(g_1(x_1), \ldots, g_n(x_n))$
- $g_1(x_1)$ through $g_{n-1}(x_{n-1})$ return two matches each
- $g_n(x_n)$ does not match anything
- Simplify's matcher: $O(2^n)$ steps, subtrigger matcher: $O(n)$ steps
- even if $g_n(x_n)$ matches something, subtrigger will still do $O(n)$ steps to match, and only $O(2^n)$ much cheaper steps to walk s-tree

Thank you!

Questions?