Progress Report
Term Dags Using Stobjs

Dpto. de Ciencias de la Computación e Inteligencia Artificial
UNIVERSIDAD DE SEVILLA
Introduction

- We are currently exploring the use of efficient data structures to implement operations on first-order terms
- Our idea is to use a single-threaded object (stobj) to store terms as directed acyclic graphs (dags)
  - Thus, operations never build new terms but merely update pointers
  - Application of substitutions needs no reconstruction of terms
- As a first attempt: implementation and verification of a unification algorithm on term dags
- The work is not finished yet
  - But we think that there are some interesting points that can be discussed
**Representation of term dags**

- $f(h(z), g(h(x), h(u))) \approx f(x, g(h(u), v))$, as a term dag:

  ![Diagram of term dags]

- **A stobj used to store term dags:**
  
  (defstobj terms-dag
   (dag :type (array t (1000)) :resizable t))

- **Every graph node is represented by a cell. Depending on the type of a node $i$, (dagi i terms-dag) stores the following:**
  
  - $(f . l)$: node $i$ is the root node of a term $f(t_1, \ldots, t_n)$ where $l$ is the list of indices corresponding to $t_1, \ldots, t_n$.
  - $(x . t)$: node $i$ stores the unbound variable $x$.
  - $n$: node $i$ stores a bound variable pointing to node $n$.

- **Example (before solving):**
  
  #((EQU 1 9) (F 2 4) (H 3) (Z . T) (G 5 7) (H 6) (X . T) (H 8) (U . T)
   (F 10 11) 6 (G 12 14) (H 13) 8 (V . T))

- **Some terminology:**
  
  - we can view an array index as a term
  - lists of pair of indices as a system of equations
  - lists of pairs of the form $(x . N)$ as substitutions
  - *indices systems and indices substitutions*
An unification algorithm

- The following function applies one step of $\Rightarrow^\text{dag}_u$, the transformation $\Rightarrow_u$ on term dags:

\[
\text{(defun dag-transform-mm} (S U \text{terms-dag})
\text{(declare (xargs :stobjs terms-dag :mode :program))}
\text{(let* ((ecu (car S)) (R (cdr S)))}
\text{(t1 (dag-deref (car ecu) terms-dag))}
\text{(t2 (dag-deref (cdr ecu) terms-dag))}
\text{(p1 (dagi t1 terms-dag)) (p2 (dagi t2 terms-dag)))}
\text{(cond}
\text{((= t1 t2) (mv R U t terms-dag))} \quad \text{;; DELETE}
\text{((dag-variable-p p1)}
\text{([if (occur-check t t1 t2 terms-dag)} \quad \text{;; CHECK}
\text{((mv nil nil terms-dag))}
\text{(let ((terms-dag (update-dagi t1 t2 terms-dag)))} \quad \text{;; ELIMINATE}
\text{((mv R (cons (dag-symbol p1) t2) U t terms-dag))})}
\text{((dag-variable-p p2)})
\text{((mv (cons (cons t2 t1) R) U t terms-dag))} \quad \text{;; ORIENT}
\text{((not (eq (dag-symbol p1) (dag-symbol p2)))} \quad \text{;; CLASH}
\text{((mv nil nil nil terms-dag))}
\text{(t (mv-let (pair-args bool)}
\text{(pair-args (dag-args p1) (dag-args p2))}
\text{(if bool)} \quad \text{;; DECOMPOSE}
\text{((mv (append pair-args R) U t terms-dag))}
\text{(mv nil nil nil terms-dag))})\text{)))} \quad \text{;; CLASH}
\text{))}
\]

- To obtain a most general unifier of two terms
  - we store both terms as graphs in the stobj
  - and iteratively apply $\Rightarrow^\text{dag}_u$, starting with the indices of the input terms and with the empty substitution
  - until the system is empty or unsolvability is found

- Remarks:
  - S and U do not contain terms but pointers
  - Syntactic restrictions enforced by stobjs are naturally ensured

Ruiz-Reina et al. CCIACL2 Workshop 2002 4
Example

Unification of \( f(h(z), g(h(x), h(u))) \approx f(x, g(h(u), v)) \)
Both terms are stored in the stobj terms-dag

Starting with the following unification problem:

\[
\begin{align*}
S & = ((1 . 9)) \text{ initial indices system to be solved} \\
U & = \text{nil} \quad \text{initial computed substitution} \\
\text{terms-dag} & = \#((\text{EQU} 1 9) (F 2 4) (H 3) (Z . T) \\
 & \quad (G 5 7) (H 6) (X . T) (H 8) (U . T) \\
 & \quad (F 10 11) 6 (G 12 14) (H 13) 8 (V . T))
\end{align*}
\]

---

Iteratively applying dag-transform-mm, we obtain:

\[
\begin{align*}
S' & = \text{nil} \\
U' & = ((V . 7) (U . 2) (X . 2)) \\
\text{terms-dag} & = \#((\text{EQU} 1 9) (F 2 4) (H 3) (Z . T) \\
 & \quad (G 5 7) (H 6) 2 (H 8) 2 \\
 & \quad (F 10 11) 6 (G 12 14) (H 13) 8 7)
\end{align*}
\]

---

Following the pointers of \( U' \) in terms-dag, we obtain the following most general unifier of the input terms:

\[
\{v \mapsto h(h(z)), \ u \mapsto h(z), \ x \mapsto h(z)\}
\]
Termination properties

- The previous functions are in :program mode
  - they are not terminating in general

- Problem: the graph stored in terms-dag could contain cycles

- Sources of non-termination:
  - Traversing the graph: for example (occur-check flg x h terms-dag) may not terminate
  - Even if occur-check is never applied, iterative applications of dag-transform-mm may not terminate

- We defined conditions that ensure termination
  - Directed acyclic graphs, dag-p
  - Main properties:
    (defthm dag-p-soundness
     (implies (not (dag-p g))
             (cycle-p (one-cyclic-path g) g)))

    (defthm dag-p-completeness
     (implies (cycle-p p g) (not (dag-p g))))

- This function allows us to define:
  * (dag-p-st terms-dag)
  * (well-formed-term-dag-st terms-dag)
  * (well-formed-upl-st S U terms-dag)

- These are expensive “type” checks
Functions in logic mode

- **Occur check:**
  
  (defun occur-check-st (flg x h terms-dag)
   (declare (xargs :measure ... :stobjs terms-dag))
   (if (dag-p-st terms-dag)
        (body)
        'undef))

- **Iterative application of** $\Rightarrow_{dag}^{u}$:

  (defun dag-solve-system-st (S U bool terms-dag)
   (declare (xargs :measure ... :stobjs terms-dag))
   (if (well-formed-upl-st S U terms-dag)
        (body)
        (mv 'undef 'undef 'undef terms-dag)))

- The measure functions are not trivial

- **Now we can define a function in logic mode** (dag-mgs-st S terms-dag), such that:
  
  - given a unification problem stored in terms-dag
  - and an indices system
  
  - returns a multvalue with a boolean (solvability), a most general solution in the form of indices substitution (in case of solvability) and terms-dag
Verification of dag-mgs-st

- Key point: if the graph stored in terms-dag is a dag, we can associate with each index of the array a term represented in the standard (list/prefix) notation

- Compositional reasoning
  - We first proved the properties of $\Rightarrow_u$ acting on the standard representation
  - Then we prove:
    If $S;U;\text{terms-dag} \Rightarrow_u^{\text{dag}} S';U';\text{terms-dag}$, then $\alpha_{\text{terms-dag}}(S;U) \Rightarrow_u^{\text{dag}} \alpha_{\text{terms-dag}}(S';U')$ where $\alpha_{\text{terms-dag}}$ transforms indices into the corresponding terms in list/prefix representation
  - One of the main proof efforts: prove that $\Rightarrow_u^{\text{dag}}$ preserves the dag-p property

- The dag-p property is essential:
  - for termination
  - for compositional reasoning (for example, structural induction on term dags)

- The main theorem we have proved:

  If (well-formed-term-dag-st terms-dag) and $S_0$ is an indices system, let $[U,\text{bool},\text{terms-dag}] = (\text{dag-mgs-st } S_0 \text{ terms-dag})$, $S = \alpha_{\text{terms-dag}}(S_0)$ and $\sigma = \alpha_{\text{terms-dag}}(U)$. Then:
  - $S$ has a solution if and only if $\text{bool} \neq \text{nil}$.
  - If $\text{bool} \neq \text{nil}$, $\sigma$ is a most general solution of $S$. 
Verification of dag-mgs-st

- Main properties proved:

  (defthm dag-mgs-st-completeness
   (let ((S (tbs-as-system-st S-dag terms-dag)))
     (implies
      (and (well-formed-dag-system-st S-dag terms-dag)
           (solution sigma S))
      (second (dag-mgs-st S-dag terms-dag))))

  (defthm dag-mgs-st-soundness
   (let* ((S (tbs-as-system-st S-dag terms-dag))
          (dag-mgs-st (dag-mgs-st S-dag terms-dag))
          (unifiable (second dag-mgs-st))
          (sol (solved-as-system-st (first dag-mgs-st)
                                    (third dag-mgs-st))))
     (implies
      (and (well-formed-dag-system-st S-dag terms-dag)
           unifiable)
      (solution sol S))))

  (defthm dag-mgs-st-most-general-solution
   (let* ((S (tbs-as-system-st S-dag terms-dag))
          (dag-mgs-st (dag-mgs-st S-dag terms-dag))
          (sol (solved-as-system-st (first dag-mgs-st)
                                     (third dag-mgs-st))))
     (implies
      (and (well-formed-dag-system-st S-dag terms-dag)
           (solution sigma S))
      (subs-subst sol sigma))))
To be done

• Integrate dag-mgs-st with a function that stores terms in the stobj
  • using the new functionalities in version 2.6 (with-local-stobj and resizable arrays)

• The algorithm is still exponential
  • we think it is not difficult to refine it in order to obtain a quadratic algorithm

• Possible future work:
  • Extensions: term rewriting, automated deduction
  • Reasoning about complexity

• But our current major problem is execution.
  • The dag-p check makes execution impractical

• One standard approach that could work:
  • A counter decremented in each recursive call: the dag check can be replaced by simpler integer tests
  • Equivalence of both versions have to be proved (for well-formed term dags)
  • As for the functions traversing dags, a suitable value for the counter is the number of total nodes

• We are exploring an alternative
Execution

- Use for execution similar functions in program mode, removing the expensive checks

- To be confident about this:
  - the functions have to be called only on term dags
  - recursion have to be closed on term dags
  - we can use the prover to ensure those conditions
  - for example, we have proved:

    \[
    \begin{align*}
    \text{(defthm well-formed-upl-st-preserved-by-dag-transform-mm-st)} \\
    \quad (& \text{implies (and (well-formed-upl-st S U terms-dag)}} \\
    \quad \quad (\text{consp S)}) \\
    \quad (\text{mv-let (S1 U1 bool1 terms-dag)}} \\
    \quad \quad (\text{dag-transform-mm-st S U terms-dag)}} \\
    \quad \quad (\text{well-formed-upl-st S1 U1 terms-dag)))))
    \end{align*}
    \]

- The \textit{guarded domain} idea of defpun (Manolios and Moore, ACL2 Workshop 2000):
  - The domain of a partial function is its guard
  - The guard verification mechanism provides built-in support for ensuring that recursion is closed
  - Drawback: termination conditions are mixed with Common Lisp compliant conditions

- We would like more built-in support for this kind of situations