Progress Report
Term Dags Using Stobjs

Ruiz-Reina J.L., Alonso, J.A., Hidalgo, M.J., Martín, F.J.

Dpto. de Ciencias de la Computación e Inteligencia Artificial

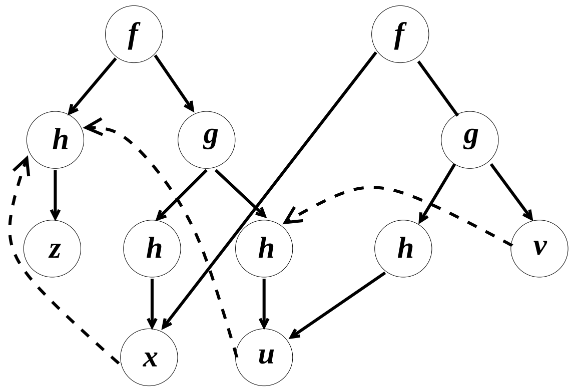
UNIVERSIDAD DE SEVILLA

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:**



- **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 $\Rightarrow_u^{dag}$, the transformation $\Rightarrow_u$ on term dags:

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

- To obtain a most general unifier of two terms
  - we store both terms as graphs in the stobj
  - and iteratively apply $\Rightarrow_u^{dag}$, 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

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:

S          = ((1 . 9))  initial indices system to be solved
U          = nil        initial computed substitution
terms-dag  = #((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))
```

```
Iteratively applying dag-transform-mm, we obtain:

S'        = nil
U'        = ((V . 7) (U . 2) (X . 2))
terms-dag = #((EQU 1 9) (F 2 4) (H 3) (Z . T)
                 (G 5 7) (H 6) 2 (H 8) 2
                 (F 10 11) 6 (G 12 14) (H 13) 8 7)
```

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-soundeness
  (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 $\Rightarrow_u^{dag}$:**

```
(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 multivalue 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 $S; U;$`terms-dag` $\Rightarrow_u^{dag}$ $S'; U';$`terms-dag`, then $\alpha_{\texttt{terms-dag}}(S;U) \Rightarrow_u \alpha_{\texttt{terms-dag}}(S';U')$ where $\alpha_{\texttt{terms-dag}}$ transforms indices into the corresponding terms in list/prefix representation
  - **One of the main proof efforts: prove that $\Rightarrow_u^{dag}$ 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 S0 is an indices system, let
[U,bool,terms-dag] = (dag-mgs-st S0 terms-dag),
$S = \alpha_{\texttt{terms-dag}}(\texttt{S0})$ and $\sigma = \alpha_{\texttt{terms-dag}}(\texttt{U})$. Then:
  - S has a solution if and only if bool$\neq$nil.
  - If bool$\neq$nil, σ 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:**

```
(defthm well-formed-upl-st-preserved-by-dag-transform-mm-st
  (implies (and (well-formed-upl-st S U terms-dag)
                (consp S))
           (mv-let (S1 U1 bool1 terms-dag)
                   (dag-transform-mm-st S U terms-dag)
                   (well-formed-upl-st S1 U1 terms-dag))))
```

- **The *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**