

Special Chapters in Artificial Intelligence

Module 2 (*L. Ciortuz*):

Optimising Parsing for Unification-Based Grammars

Administrative issues:

Please refer to “Fişa disciplinei” on the FII site

Tentative schedule:

7 courses in October (29.09, and 02.10, 06.10, 09.10, 20.10, 23.10, 27.10)

+ 2 written short exams (16.10 and 30.10)

5 seminars in November and December (06.11, 13.11, 27.11, 4.12, 11.12) project work + evaluation

1. From first-order terms towards feature structures

Courses 1-2 (29.09 and 02.10)

- A. first-order terms
 - substitutions
 - unification of first-order terms
 - most general unifiers; algorithms for the computation of mgu's
- B. Generalised Modus Ponens;
 - inference in first-order logic:
 - a forward chaining algorithm
 - a backward chaining algorithm;
 - the resolution principle
- C. Generalising first-order terms:
 - using attributes – free order of subterms; incomplete specification
 - tags vs variables
 - partial order relation between sorts

Recommended readings:

Russell and Norvig, *Artificial Intelligence: A Modern Approach* (the so called AIMA book), 2nd ed., ch. 8.2, 9.1–9.3 (optionally: 9.4–9.5)

Kevin Knight, “Unification: A Multidisciplinary Survey”, *ACM Computing Surveys*, 1989

Krzysztof Apt, “The Logic Programming Paradigm and Prolog”, ch. 15 in “*Concepts in Programming Languages*”, John Mitchell, Stanford University, 2002

Optional readings:

Hassan Ait-Kaci, Roger Nasr, “LOGIN: A Logic Programming with Built-in Inheritance”, *Journal of Logic Programming*, 1986

2. From feature structures towards unification-based grammars

Courses 3-5 (06.10, 09.10 and 20.10):

- A. Feature Structures:
 - subsumption,
 - normalisation,
 - unification (algorithms: Hassan Aït-Kaci and Roger Nasr; Aït-Kaci and Roberto Di Cosmo)
- B. The Cocke-Younger-Kasami parsing algorithm and its deductive (active) form

Recommended readings:

“Logic and Inheritance”, Hassan Aït-Kaci, Roger Nasr, ACM, 1985

“Towards a Meaning of LIFE”, Hassan Aït-Kaci, Andreas Podelski, Journal of Logic Programming, 1993

“Compiling Order-Sorted Feature Term Unification”, Hassan Aït-Kaci, Roberto Di Cosmo, PRL Technical Report, 1993

“Principles and Implementation of Deductive Parsing”, Stuart Shieber, Yves Schabes, Fernando Pereira, Journal of Logic Programming, 1995

Optional readings:

“Speech and Language Processing”, Daniel Jurafsky and James Martin, 2000, ch. 10-11

3. Exemplifying unification-based grammars

Courses 6-7 (23.10 and 27.10):

Recommended readings:

“Parsing Scemata”, Klaas Sikkels, Springer, 1997, ch. 7, and eventually ch. 8-9

In-depth readings:

“An introduction to unification-based approaches to grammars”, Stuart Shieber, CSLI Publications, Stanford, 1986

“Paradigms of Artificial Intelligence”, Peter Norvig, 1992, ch. 19-20, and eventually ch. 21

1.A. Unification algorithms for terms in first-order logic

A version of Robinson's unification algorithm (1965)

by Kevin Knight, borrowed from [Corbin & Bidoit, 1983]

```
function UNIFY( $t_1, t_2$ )  $\Rightarrow$  (unifiable: Boolean,  $\sigma$ : substitution)
begin
  if  $t_1$  or  $t_2$  is a variable then
    begin
      let  $x$  be the variable, and let  $t$  be the other term
      if  $x = t$ , then (unifiable,  $\sigma$ )  $\leftarrow$  (true,  $\emptyset$ )
      else if occur( $x, t$ ) then unifiable  $\leftarrow$  false
      else (unifiable,  $\sigma$ )  $\leftarrow$  (true,  $\{x/t\}$ )
    end
  else
    begin
      assume  $t_1 = f(x_1, \dots, x_n)$  and  $t_2 = g(y_1, \dots, y_m)$ 
      if  $f \neq g$  or  $n \neq m$  then unifiable  $\leftarrow$  false
      else
        begin
           $k \leftarrow 0$ 
          unifiable  $\leftarrow$  true
           $\sigma \leftarrow$  nil
          while  $k < m$  and unifiable do
            begin
               $k \leftarrow k+1$ 
              (unifiable,  $\tau$ )  $\leftarrow$  UNIFY( $\sigma(x_k), \sigma(y_k)$ )
              if unifiable then  $\sigma$  compose( $\tau, \sigma$ )
            end
          end
        end
      end
    end
  return (unifiable,  $\sigma$ )
end
```

Complexity: exponential, both in time and space

A version of Huet's unification algorithm (1976)

by Kevin Knight

```
function UNIFY( $t_1, t_2$ )  $\Rightarrow$  (unifiable: Boolean,  $\sigma$ : substitution)
begin
  pairs-to-unify  $\leftarrow \{(t_1, t_2)\}$ 
  for each node  $z$  in  $t_1$  and  $t_2$ 
     $z.class \leftarrow z$ 
  while pairs-to-unify  $\neq \emptyset$  do
    begin
      ( $x, y$ )  $\leftarrow$  pop(pairs-to-unify)
       $u \leftarrow$  FIND( $x$ )
       $v \leftarrow$  FIND( $y$ )
      if  $u \neq v$  then
        if  $u$  and  $v$  are not variables, and  $u.symbol \neq v.symbol$  or
          numberof( $u.subnodes$ )  $\neq$  numberof( $v.subnodes$ ) then
            return (false, nil)
        else
          begin
             $w \leftarrow$  UNION( $u, v$ )
            if  $w = v$  and  $u$  is a variable then
               $u.class \leftarrow v$ 
            if neither  $v$  nor  $u$  is a variable then
              begin
                let ( $u_1, \dots, u_n$ ) be  $u.subnodes$ 
                let ( $v_1, \dots, v_n$ ) be  $v.subnodes$ 
                for  $i \leftarrow 1$  to  $n$  do
                  push ( $(u_i, v_i)$ , pairs-to-unify)
              end
            end
          end
      end
    end
  end
  Form a new graph composed of the root nodes of the equivalence classes.
  This graph is the result of the unification.
  If the graph has a cycle, return (false, nil),
  but the terms are infinitely unifiable.
  If the graph is acyclic, return (true,  $\sigma$ ), where
   $\sigma$  is a substitution in which any variable  $x$  is mapped on to
  the root of its equivalence class, that is, FIND( $x$ ).
end
```

Notes:

0. Huet's idea consists in the computation of the closure of a (least) congruence/equivalence relation. It is a variation on the algorithm deciding the equivalence of two finite-state automata (see [Aho, Hopcroft, Ulman, "The design and analysis of computer algorithms", 1974], pag. 129-145).

1. Terms are represented as graphs whose nodes have the following structure:

```
type = node
  symbol:    a function, variable, or constant symbol
  subnodes:  a list of nodes that are children of this node
  class:     a node that represents this node's equivalence class
end
```

2. The algorithm maintains a set of equivalence classes of nodes. Each class has a unique representative, given by the FIND operation. The UNION x, y operation computes the union of two – disjoint – classes represented by x and y ; it returns the representative of the newly formed class.

Complexity: $O(n\alpha(n))$ where $\alpha(n)$ is an extremely slow-growing function.

An elegant unification algorithm (Martelli and Montanari, 1982)

Input: a finite set of term equations $\{s_1 = t_1, \dots, s_n = t_n\}$.

Output: an mgu of them, or 'failure' if they are not unifiable.

Procedure:

Nondeterministically choose from the set of equations an equation of a form below and perform the associated action:

- (1) $f(s_1, \dots, s_n) = f(t_1, \dots, t_n)$
replace by the equations $s_1 = t_1, \dots, s_n = t_n$
- (2) $f(s_1, \dots, s_n) = g(t_1, \dots, t_m)$ where $f \neq g$
halt with failure
- (3) $x = x$
delete the equation
- (4) $t = x$ where t is not a variable
replace by the equation $x = t$
- (5) $x = t$ where x does not occur in t and x occurs elsewhere
apply the substitution x/t to all other equations
- (6) $x = t$ where x occurs in t and x differs from t
halt with failure.

The algorithm terminates when no action can be performed or when failure arises.

In case of success, by changing in the final set of equations all occurrences of '=' to '/', we obtain the desired mgu.

Notes:

- (1) includes the case $c = c$ for every constant c which leads to deletion of such an equation
- action (2) includes the case of two different constants
- no action is performed when the selected equation is of the form $x = t$ where x does not occur elsewhere (so a fortiori does not occur in t). In fact, in case of success all equations will be of such a form.

Complexity: (from Kevin Knight, 1989):

$O(n + m \log m)$, where m is the number of distinct variables in the terms.

Example 1:

Reconsider the equation $f(x, a) = f(b, y)$.

By using action (1), it rewrites to the set of two equations, $x = b, a = y$.

By action (4) we now get the set $x = b, y = a$.

At this moment the algorithm terminates and we obtain the mgu $x/b, y/a$.

Example 2:

Consider the sequence $f(x, a) = f(g(z), y), h(x, z) = h(d, u)$. It yields a failure. Indeed, after executing the first equation the variable x is bound to $g(z)$, so the evaluation of the second equation yields $h(g(z), z) = h(d, u)$ and no substitution makes equal (unifies) the terms $h(g(z), z)$ and $h(d, u)$.

2.A. Feature Structures

by Hassan-Aït-Kaci and Andreas Podelski (1993)

Definition:

A **feature structure** (or, OSF-term) is an expression of the form:

$$\psi = X : s(l_1 \Rightarrow \psi_1, \dots, l_n \Rightarrow \psi_n)$$

where X is a variable (in \mathcal{V}), s is a sort in \mathcal{S} , l_1, \dots, l_n are features in \mathcal{F} , and ψ_1, \dots, ψ_n ($n \geq 0$) are feature structures.

Note that the equation above includes $n = 0$ as a base case. Thus, the simplest FSs are of the form $X : s$. We call the variable X in the above FS the *root* of ψ (denoted $Root(\psi)$), and say that X is “sorted” by the sort s and “has attributes/features” l_1, \dots, l_n .

Example 1:

$$\begin{aligned} X:person(name \Rightarrow N:\top (first \Rightarrow F:string), \\ name \Rightarrow M:id(last \Rightarrow S:string), \\ spouse \Rightarrow P:person(name \Rightarrow I:id(last \Rightarrow S:\top), \\ spouse \Rightarrow X:\top)) \end{aligned}$$

Note that, in general, an OSF-term may have redundant attributes (e.g., *name* above), or the same variable sorted by different sorts (e.g., X and S above).

We shall usually use a *lightened notation* for FSs whereby any variable occurring without a sort is implicitly sorted with \top and all variables which do not occur more than once are not given explicitly.

Example 1':

Using this light notation, the FS of Example 1 becomes:

$$\begin{aligned} X:person(name \Rightarrow \top (first \Rightarrow string), \\ name \Rightarrow id(last \Rightarrow S:string), \\ spouse \Rightarrow person(name \Rightarrow id(last \Rightarrow S), \\ spouse \Rightarrow X)) \end{aligned}$$

Definition:

A **normal FS** ψ is of the form $\psi = X : s(l_1 \Rightarrow \psi_1, \dots, l_n \Rightarrow \psi_n)$ where:

- there is at most one occurrence of a variable Y in ψ such that Y is the root variable of a non-trivial FS (i.e., different than $Y : \top$);
- s is a non-bottom sort in \mathcal{S} ;
- l_1, \dots, l_n are pairwise distinct features in \mathcal{F} , $n \geq 0$,
- ψ_1, \dots, ψ_n are normal FSs.

Example 1'':

One could verify easily that the FS

$$\begin{aligned} X:person(name \Rightarrow id(first \Rightarrow string, \\ last \Rightarrow S:string), \\ spouse \Rightarrow person(name \Rightarrow id(last \Rightarrow S), \\ spouse \Rightarrow X)) \end{aligned}$$

is a normal FS and it correspondes exactly to the FS in Example 1.

Remark:

We will later present a rewriting rule-based algorithm for the normalisation of a given FS.

FS Normalisation:

Definition:

A FS **atomic constraint** (or, OSF-constraint) is an expression of either of the forms:

$$\begin{array}{ll} X : s & \text{read as: } \text{“}X \text{ lies in sort } s\text{”} \\ X \doteq Y & \text{“}X \text{ is equal to } Y\text{”} \\ X.l \doteq Y & \text{“}Y \text{ is the feature } l \text{ of } X\text{”} \end{array}$$

where X and Y are variables in \mathcal{V} , s is a sort in S , and l is a feature in \mathcal{F} .

Definition:

An **OSF-clause** $\phi_1 \& \dots \& \phi_n$ is a finite, possibly empty conjunction of atomic constraints ϕ_1, \dots, ϕ_n ($n \geq 0$).

Example 2:

The OSF-clause corresponding to the FS of Example 1 is the following:

$$\begin{aligned} X:\textit{person} \& X.\textit{name} \doteq N \& N:\top \& N:\textit{first} \doteq F \& F:\textit{string} \& \\ X.\textit{name} \doteq M \& M:\textit{id} \& M:\textit{last} \doteq S \& S:\textit{string} \& \\ X:\textit{spouse} \doteq P \& P:\textit{person} \& P.\textit{name} \doteq \& I:\textit{id} \& \\ & I.\textit{last} \doteq S \& S:\top \& \\ & P.\textit{spouse} \doteq X \& X:\top \end{aligned}$$

Definition:

A **rooted OSF-clause** ϕ_X is an OSF-clause together with a distinguished variable X (called its *root*) such that every variable Y occurring in ϕ_X is explicitly sorted (possibly as $Y : \perp$), and reachable from X .

Definition:

An **OSF-clause** ϕ is called **solved** if for every variable X , ϕ contains:

- at most one sort constraint of the form $X : s$, with $\perp < s$
- at most one feature constraint of the form $X.l \doteq Y$ for each l , and
- no equality constraint of the form $X \doteq Y$.

OSF-Clause Normalisation Rules:

Given an OSF-clause ϕ , it can be normalized by choosing non-deterministically and applying any applicable rule among the four transformations rules shown below, until none applies.

$$\begin{array}{ll} (\textit{Inconsistent Sort}) & \frac{\phi \& X:\perp}{X:\perp} \\ (\textit{Sort Intersection}) & \frac{\phi \& X:s \& X:s'}{\phi \& X:s \wedge s'} \\ (\textit{Feature Decomposition}) & \frac{\phi \& X.l \doteq Y \& X.l \doteq Y'}{\phi \& X.l \doteq Y \& Y \doteq Y'} \\ (\textit{Variable Elimination}) & \frac{\phi \& X \doteq Y}{\phi[X/Y] \& X \doteq Y} \text{ if } X \in \textit{Var}(\phi) \end{array}$$

Note: The expression $\phi[X/Y]$ stands for the formula obtained from ϕ after replacing all occurrences of Y by X . We also refer to any clause of the form $X : \perp$ as the *fail* clause.

Theorem:

The above normalisation rules are solution-preserving, finite terminating, and confluent (modulo variable renaming). Furthermore, they always result in a normal form that is either the fail (inconsistent) clause or an OSF-clause in solved form together with a conjunction of equality constraints.

Example 3:

The normalization of the OSF-clause given in Example 2 leads to the solved OSF-clause which is the conjunction of the equality constraint $N \doteq M$ and the following solved OSF-clause:

$$\begin{aligned} X:person \ \& \ X.name \doteq N \ \& \ N:id \ \& \ N:first \doteq F \ \& \ F:string \ \& \\ & N:last \doteq S \ \& \ S:string \ \& \\ X:spouse \doteq P \ \& \ P:person \ \& \ P.name \doteq \ \& \ I:id \ \& \\ & I.last \doteq S \ \& \\ & P.spouse \doteq X \end{aligned}$$

Proposition (Syntactic Bijection):

There is a one-one correspondence between normal FSs and rooted solved OSF-clauses.

FS Subsumption:

Generalizing first-order terms, 2 FSs are ordered up to variable renaming:

Given that the set of sorts \mathcal{S} is partially-ordered (with a greatest element \top and a lowest element \perp), its partial ordering is extended to the set of FSs. Informally, a FS t_1 is subsumed by a FS t_2 (and this will be denoted as $t_1 \sqsubseteq t_2$) if

- (1) the root sort of t_1 is a subsort in \mathcal{S} of the root sort of t_2 ;
- (2) all features of t_2 are also features of t_1 ,
having as values FSs which subsume their homologues in t_1 ; and,
- (3) all coreference constraints binding in t_2 must also be binding in t_1 .

Example 4:

If $student < person$ and $paris < cityname$ in the sort signature \mathcal{S} then the following FS

$$\begin{aligned} &student(id \Rightarrow name(first \Rightarrow string, \\ &\qquad\qquad\qquad last \Rightarrow X:string), \\ &\quad lives_at \Rightarrow Y:address(city \Rightarrow paris), \\ &\quad father \Rightarrow person(id \Rightarrow name(last \Rightarrow X), \\ &\qquad\qquad\qquad lives_at \Rightarrow Y)) \end{aligned}$$

is subsumed by the feature structure

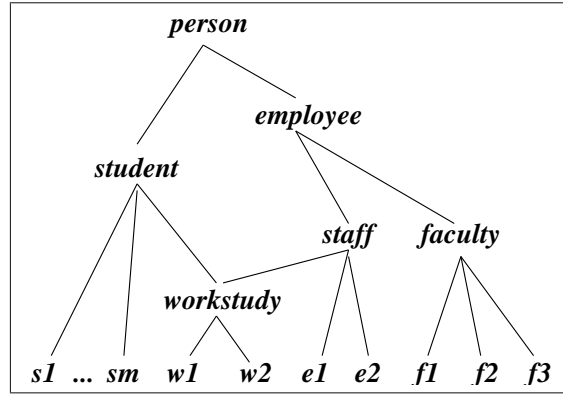
$$\begin{aligned} &person(id \Rightarrow name(last \Rightarrow X:string), \\ &\quad lives_at \Rightarrow address(city \Rightarrow cityname), \\ &\quad father \Rightarrow person(id \Rightarrow name(last \Rightarrow X))) \end{aligned}$$

FS Unification:

If the sort signature \mathcal{S} is such that greatest lower bounds (GLBs) exist for any pair of type symbols (that is, \mathcal{S} is a lower semi-lattice of sorts), then the subsumption ordering on FSs is also such that GLBs exist. The *unification* of two FSs is defined in this way.

Example 5:

Assuming the sort signature graphically presented below,



the two feature structures:

$$\psi_1 = X:\text{student}(\text{advisor} \Rightarrow \text{faculty}(\text{secretary} \Rightarrow Y:\text{staff}, \\ \text{assistant} \Rightarrow X), \\ \text{room-mate} \Rightarrow \text{employee}(\text{representative} \Rightarrow Y))$$

and

$$\psi_2 = \text{employee}(\text{advisor} \Rightarrow f1 (\text{secretary} \Rightarrow \text{employee}, \\ \text{assistant} \Rightarrow U:\text{person}), \\ \text{room-mate} \Rightarrow V:\text{student}(\text{representative} \Rightarrow V), \\ \text{helper} \Rightarrow w1 (\text{spouse} \Rightarrow U))$$

have as their unification the term:

$$\psi = W:\text{workstudy}(\text{advisor} \Rightarrow f1(\text{secretary} \Rightarrow Z:\text{workstudy}(\text{representative} \Rightarrow Z), \\ \text{assistant} \Rightarrow W), \\ \text{room-mate} \Rightarrow Z, \\ \text{helper} \Rightarrow w1(\text{spouse} \Rightarrow W))$$

Note: The two FSs initially given above would not unify if in the sort signature the line (denoting the is-a relationship) *workstudy* — *stuff* would have been replaced by *workstudy* — *employee*.

Note: Ait-Kaci's algorithm for unification of two FSs (see the next page) uses the same idea as Huet's algorithm for unification of first-order terms:

- follow through all possible attribute/feature paths in both FSs;
- pairs of nodes/variables that are reachable following the same path of features are merged into coreference classes.

Theorem (FS Unification):

Let ψ_1 and ψ_2 be two FSs.

Let ϕ be the normal form of the OSF-clause $\phi(\psi_1) \& \phi(\psi_2) \& \text{Root}(\psi_1) \doteq \text{Root}(\psi_2)$.

Then, ϕ is the inconsistent clause iff the GLB of ψ_1 and ψ_2 with respect to the subsumption relation is \perp .

If ϕ is not the inconsistent clause, then the GLB of $\psi_1 \wedge \psi_2$ (modulo variable renaming) is given by the normal feature structure $\psi(\text{Solved}(\phi))$.

Theorem (Semantic Transparency of Orderings):

If the normal OSF-terms ψ , ψ' , and the rooted solved OSF-clauses ϕ_X , ϕ'_X respectively correspond to one another through the syntactic mappings, then the following are equivalent statements:

- $\psi' \sqsubseteq \psi$, i.e. ψ' is more specific than (i.e. it is subsumed by) ψ ,
- $\phi'_X \models \phi_X$ i.e. ϕ is true of X whenever ϕ' is true of X .

Example 6:

The unification of the two FSs ϕ_1 and ϕ_2 in Example 5 can be done via normalisation as follows:

Note that we must introduce fresh variables wherever needed, in order to reverse from the *lightened* notation to the one stated in original definition for FSs. Then the two FSs get translated as OSF-clauses:

$$\begin{aligned}\phi(\psi_1) = & X:\textit{student} \ \& \ X.\textit{advisor} \doteq V_1 \ \& \ V_1:\textit{faculty} \ \& \ V_1.\textit{secretary} \doteq Y \ \& \ Y:\textit{staff} \ \& \\ & V_1.\textit{assistant} \doteq X \ \& \\ X.\textit{room-mate} \doteq & V_2 \ \& \ V_2:\textit{employee} \ \& \ V_2.\textit{representative} \doteq Y\end{aligned}$$

and

$$\begin{aligned}\phi(\psi_2) = & V_3:\textit{employee} \ \& \ V_3.\textit{advisor} \doteq V_4 \ \& \ V_4:\textit{f1} \ \& \ V_4.\textit{secretary} \doteq V_5 \ \& \ V_5:\textit{employee} \ \& \\ & V_4.\textit{assistant} \doteq U \ \& \ U:\textit{person} \ \& \\ V_3.\textit{room-mate} \doteq & V \ \& \ V:\textit{student} \ \& \ V.\textit{representative} \doteq V \ \& \\ V_3.\textit{helper} \doteq & V_6 \ \& \ V_6:\textit{w1} \ \& \ V_6.\textit{spouse} \doteq U\end{aligned}$$

According to the above theorem (FS Unification), we will normalise the clause $\phi(\psi_1) \wedge \phi(\psi_2) \wedge X \doteq V_3$: First, the rule *Variable Elimination* is applied, due to the atomic constraint $X \doteq V_3$. This would eventually lead to

$$\begin{aligned}\phi'(\psi_2) = & X:\textit{employee} \ \& \ X.\textit{advisor} \doteq V_4 \ \& \ V_4:\textit{f1} \ \& \ V_4.\textit{secretary} \doteq V_5 \ \& \ V_5:\textit{employee} \ \& \\ & V_4.\textit{assistant} \doteq U \ \& \ U:\textit{person} \ \& \\ X.\textit{room-mate} \doteq & V \ \& \ V:\textit{student} \ \& \ V.\textit{representative} \doteq V \ \& \\ X.\textit{helper} \doteq & V_6 \ \& \ V_6:\textit{w1} \ \& \ V_6.\textit{spouse} \doteq U\end{aligned}$$

The rule *Sort Intersection* can now be applied to $X:\textit{student} \wedge X:\textit{employee}$, leading to replace these constraints with $X:\textit{workstudy}$ in both $\phi(\psi_1)$ and $\phi'(\psi_2)$.

Further on, *Feature Decomposition* applies for $\phi(\psi_1) \wedge \phi'(\psi_2) \wedge X \doteq V_3$ (more exactly for $X.\textit{advisor}$ and $X.\textit{room-mate}$) leading to adding to this clause the equations $V_1 \doteq V_4$ and $V_2 \doteq V$.

Next, applying *Variable Elimination* leads to propagating these two equations in $\phi'(\psi_2)$:

$$\begin{aligned}\phi''(\psi_2) = & X:\textit{workstudy} \ \& \ X.\textit{advisor} \doteq V_1 \ \& \ V_1:\textit{f1} \ \& \ V_1.\textit{secretary} \doteq V_5 \ \& \ V_5:\textit{employee} \ \& \\ & V_1.\textit{assistant} \doteq U \ \& \ U:\textit{person} \ \& \\ X.\textit{room-mate} \doteq & V_2 \ \& \ V_2:\textit{student} \ \& \ V_2.\textit{representative} \doteq V_2 \ \& \\ X.\textit{helper} \doteq & V_6 \ \& \ V_6:\textit{w1} \ \& \ V_6.\textit{spouse} \doteq U\end{aligned}$$

It follows that *Sort Intersection* must be applied once for V_1 and then for V_2 . Therefore $V_1:\textit{faculty} \wedge V_1:\textit{f1}$ will be substituted with $V_1:\textit{f1}$ (in $\phi(\psi_1)$), and $V_2:\textit{employee} \wedge V_2:\textit{student}$ will be replaced with $V_2:\textit{workstudy}$ (in both $\phi(\psi_1)$ and $\phi''(\psi_2)$).

At this point in the normalisation process, the clause $\phi(\psi_1) \wedge \phi(\psi_2) \wedge X \doteq V_3$ would have reached the form

$$\begin{aligned}X \doteq V_3 \wedge V_1 \doteq V_4 \wedge V_2 \doteq V_6 \wedge \\ X:\textit{workstudy} \ \& \ X.\textit{advisor} \doteq V_1 \ \& \ V_1:\textit{f1} \ \& \ V_1.\textit{secretary} \doteq Y \ \& \ Y:\textit{staff} \ \& \\ & V_1.\textit{assistant} \doteq X \ \& \\ X.\textit{room-mate} \doteq & V_2 \ \& \ V_2:\textit{workstudy} \ \& \ V_2.\textit{representative} \doteq Y \ \& \\ X:\textit{workstudy} \ \& \ X.\textit{advisor} \doteq & V_1 \ \& \ V_1:\textit{f1} \ \& \ V_1.\textit{secretary} \doteq V_5 \ \& \ V_5:\textit{employee} \ \& \\ & V_1.\textit{assistant} \doteq U \ \& \ U:\textit{person} \ \& \\ X.\textit{room-mate} \doteq & V_2 \ \& \ V_2:\textit{workstudy} \ \& \ V_2.\textit{representative} \doteq V_2 \ \& \\ X.\textit{helper} \doteq & V_6 \ \& \ V_6:\textit{w1} \ \& \ V_6.\textit{spouse} \doteq U\end{aligned}$$

Be using the idempotency of the logical conjunction (\wedge), the above clause gets simplified to the form

$$\begin{aligned}
X \doteq V_3 \wedge V_1 \doteq V_4 \wedge V_2 \doteq V_6 \wedge \\
X:\text{workstudy} \ \& \ X:\text{advisor} \doteq V_1 \ \& \ V_1:f1 \ \& \ V_1:\text{secretary} \doteq Y \ \& \ Y:\text{staff} \ \& \\
& \ V_1:\text{assistant} \doteq X \ \& \\
X:\text{room-mate} \doteq V_2 \ \& \ V_2:\text{workstudy} \ \& \ V_2:\text{representative} \doteq Y \ \& \\
& \ V_1:\text{secretary} \doteq V_5 \ \& \ V_5:\text{employee} \ \& \\
& \ V_1:\text{assistant} \doteq U \ \& \ U:\text{person} \ \& \\
& \ V_2:\text{representative} \doteq V_2 \ \& \\
X:\text{helper} \doteq V_6 \ \& \ V_6:w1 \ \& \ V_6:\text{spouse} \doteq U
\end{aligned}$$

Feature Decomposition applied to $V_1.\text{secretary}$ (and $V_1.\text{assistant}$) will add the equation(s) $Y \doteq V_5$ (and respectively $X \doteq U$), after which, *Variable Elimination* will produce:

$$\begin{aligned}
X \doteq V_3 \wedge V_1 \doteq V_4 \wedge V_2 \doteq V_6 \wedge \\
X:\text{workstudy} \ \& \ X:\text{advisor} \doteq V_1 \ \& \ V_1:f1 \ \& \ V_1:\text{secretary} \doteq Y \ \& \ Y:\text{staff} \ \& \\
& \ V_1:\text{assistant} \doteq X \ \& \\
X:\text{room-mate} \doteq V_2 \ \& \ V_2:\text{workstudy} \ \& \ V_2:\text{representative} \doteq Y \ \& \\
& \ Y:\text{employee} \ \& \\
& \ X:\text{person} \ \& \\
& \ V_2:\text{representative} \doteq V_2 \ \& \\
X:\text{helper} \doteq V_6 \ \& \ V_6:w1 \ \& \ V_6:\text{spouse} \doteq X
\end{aligned}$$

Then the constraints $Y:\text{employee}$ and $X:\text{person}$ will be eliminated through application of *Sort Intersection*.

Finally, *Feature Decomposition* applied to $V_2.\text{representative}$ will lead to teh equation $Y \doteq V_2$, thus producing (through *Variable Elimination*):

$$\begin{aligned}
X \doteq V_3 \ \& \ V_1 \doteq V_4 \ \& \ V_2 \doteq V_6 \ \& \ Y \doteq V_5 \ \& \ X \doteq U \ \& \ Y \doteq V_2 \ \& \\
X:\text{workstudy} \ \& \ X:\text{advisor} \doteq V_1 \ \& \ V_1:f1 \ \& \ V_1:\text{secretary} \doteq Y \ \& \ Y:\text{staff} \ \& \\
& \ V_1:\text{assistant} \doteq X \ \& \\
X:\text{room-mate} \doteq Y \ \& \ Y:\text{workstudy} \ \& \ Y:\text{representative} \doteq Y \ \& \\
X:\text{helper} \doteq V_6 \ \& \ V_6:w1 \ \& \ V_6:\text{spouse} \doteq X
\end{aligned}$$

and *Sort Intersection* applied on $Y:\text{staff}$ & $Y:\text{workstudy}$ will eliminate the first one of these two constraints to produce:

$$\begin{aligned}
X \doteq V_3 \ \& \ V_1 \doteq V_4 \ \& \ V_2 \doteq V_6 \ \& \ Y \doteq V_5 \ \& \ X \doteq U \ \& \ Y \doteq V_2 \ \& \\
X:\text{workstudy} \ \& \ X:\text{advisor} \doteq V_1 \ \& \ V_1:f1 \ \& \ V_1:\text{secretary} \doteq Y \ \& \ Y:\text{workstudy} \ \& \\
& \ Y:\text{representative} \doteq Y \ \& \\
& \ V_1:\text{assistant} \doteq X \ \& \\
X:\text{room-mate} \doteq Y \ \& \\
X:\text{helper} \doteq V_6 \ \& \ V_6:w1 \ \& \ V_6:\text{spouse} \doteq X
\end{aligned}$$

By putting asside the equations $X \doteq V_3$, $V_1 \doteq V_4$, $V_2 \doteq V_6$, $Y \doteq V_5$, $X \doteq U$, $Y \doteq V_2$, the remaining solved clause will be translated into the following FS, which is the unification result of ψ_1 and ψ_2 .

$$\begin{aligned}
\psi = X:\text{workstudy}(\text{advisor} \Rightarrow V_1:f1(\text{secretary} \Rightarrow Y:\text{workstudy}(\text{representative} \Rightarrow Y), \\
& \ \text{assistant} \Rightarrow X), \\
& \ \text{room-mate} \Rightarrow Y, \\
& \ \text{helper} \Rightarrow V_6:w1(\text{spouse} \Rightarrow X))
\end{aligned}$$

or, in lightened notation:

$$\begin{aligned}
\psi = X:\text{workstudy}(\text{advisor} \Rightarrow f1(\text{secretary} \Rightarrow Y:\text{workstudy}(\text{representative} \Rightarrow Y), \\
& \ \text{assistant} \Rightarrow X), \\
& \ \text{room-mate} \Rightarrow Y, \\
& \ \text{helper} \Rightarrow w1(\text{spouse} \Rightarrow X))
\end{aligned}$$

[LC:] A simple recursive FS unification algorithm

% unification may be seen as proceeding recursively:

```
start with the root nodes  $X_0$  and  $Y_0$ ;  
for the current pair of nodes  $X$  and  $Y$ ,  
begin  
  compute the GLB of the two sorts  $X.type$  and  $Y.type$   
  if this is  $\perp$  then return fail;  
  else  
    for each common feature  $l$  of  $X$  and  $Y$ ,  
      the respective values ( $X.l$  and  $Y.l$ ) should be unified  
      % ( $X.l \doteq X1$  and  $Y.l \doteq Y1$ )  
      % here comes recursion  
    each un-common feature from either  $X$  or  $Y$   
      should be carried onto the resulting FS  
end
```

A Feature Structure Unification Algorithm

Hassan Aït-Kaci, 1984

```

tagnode = record
    id      : tag symbol;
    type    : constructor symbol;
    subnodes : set of pairs <label, tagnode>;
    correferance : tag node;
end

procedure UNIFY(s, t);
begin
    PAIRS ← {< X0, Y0 >}; // X0, Y0 are the roots of s and t
    while PAIRS ≠ ∅ do
        begin
            remove < x, y > from PAIRS;
            u ← FIND(x);
            v ← FIND(y);
            if u ≠ v then
                begin
                    σ ← u.type ∧ v.type;
                    if σ = ⊥ then return(⊥)
                    else
                        begin
                            UNION(u,v,w);
                            w.type ← σ;
                            for each l in labels(u) ∪ labels(v) do
                                begin
                                    if w = v
                                        then CARRY_LABEL(l,u,v);
                                        else CARRY_LABEL(l,v,u);
                                    if l ∈ labels(u) ∩ labels(v) then
                                        PAIRS ← PAIRS ∪ {< subterm(u,l), subterm(v,l) >}
                                    end
                                end
                            end
                        end
                    end
                end
            return(REBUILD(tags(s) ∪ tags(t)))
        end
end

procedure CARRY_LABEL(l,u,v);
begin
    if l ∉ labels(v) then
        v.subnodes ← v.subnodes ∪ {< l, FIND(subterm(u,l)) >}
    end
end

procedure REBUILD(tagset);
begin
    CLASSES ← ∪x∈tagset {FIND(x)};
    for each x ∈ CLASSES do ID[x] ← NewTagSymbol;
    for each x ∈ CLASSES do
        begin
            NODE ← NewTagNode;
            with NODE do
                begin
                    id ← ID[x];
                    type ← x.type;
                    subnodes ← {< l, ID[ FIND(y) ]> | < l,y> ∈ x.subnodes >};
                    correferance ← nil
                end
            end
        end
    return(ID[ FIND(X0) ])
end

```

Another Feature Structure Unification Algorithm

Hassan Ait-Kaci and Roberto Di Cosmo, 1993

```
boolean osf_unify( int a1, int a2 )
{
  boolean fail = FALSE;
  push_PDL( &PDL, a1 ); push_PDL( &PDL, a2 );
  while non_empty( &PDL ) ^¬fail {
    d1 = deref( pop( &PDL ) ), d2 = deref( pop( &PDL ) );
    if d1 ≠ d2 {
      new_sort = heap[d1].SORT ^ heap[d2].SORT;
      if new_sort = BOT
        fail = TRUE;
      else {
        bind_refine( d1, d2, new_sort )
        if deref( d1 ) = d2
          carry_features( d1, d2 );
        else
          carry_features( d2, d1 ); } } }
  return ¬fail;
}

bind_refine( int d1, int d2, sort s )
{
  heap[ d1 ].CREF = d2;
  heap[ d2 ].SORT = s;
}

carry_features( int d1, int d2 )
{
  FEAT_frame *frame1 = heap[ d1 ].FTAB, *frame2 = heap[ d2 ].FTAB;
  FHEAP_cell *feats1 = frame1 -> feats, *feats2 = frame2 -> feats;
  int feat, nf = frame1 -> nf;
  for (feat = 0; feat < nf; ++feat) {
    int f, f1 = feats1[ feat ].FEAT, v1 = feats1[ feat ].TERM, v2;
    if ((f = get_feature( d2, f1 )) ≠ FAIL) {
      v2 = feats2[ f ].TERM;
      push_PDL( &PDL, v2 );
      push_PDL( &PDL, v1 ); }
    else
      add_feature( d2, f1, v1 ); }
}

int deref( int a )
{
  int b;
  for (b = heap[ a ].CREF;
       a != b;
       a = b, b = heap[ a ].CREF)
    ;
  return a;
}
```

Example 7:

Let us consider two FSs and a sort signature in which $b \wedge c = d$ and the symbol $+$ is a subsort of the sort *bool*.

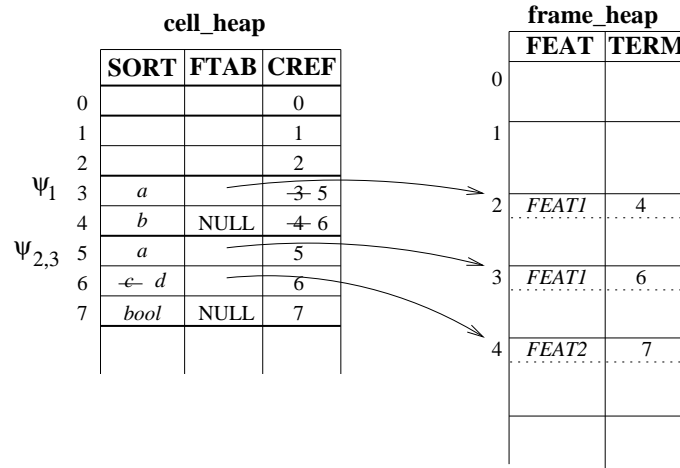
The *glb* (i.e. unification result) of ψ_1 and ψ_2

$$\begin{aligned} \psi_1 &= a(\text{FEAT1} \Rightarrow b), \\ \psi_2 &= a(\text{FEAT1} \Rightarrow c(\text{FEAT2} \Rightarrow \text{bool})), \end{aligned}$$

is

$$\psi_3 = a(\text{FEAT1} \rightarrow d(\text{FEAT2} \rightarrow \text{bool})),$$

The effect of typed (OSF-theory) unification on ψ_1 and ψ_2



Compiled FS unification

Hassan Aït-Kaci and Roberto Di Cosmo, 1993

The ‘query’ abstract code
for ψ_1 (left) and ψ_2 (right)

push_cell 0	push_cell 0
set_sort 0, a	set_sort 0, a
push_cell 1	push_cell 1
set_feature 0, FEAT1, 1	set_feature 0, FEAT1, 1
set_sort 1, b	set_sort 1, c
	push_cell 2
	set_feature 1, FEAT2, 2
	set_sort 2, bool

‘WRITE’ abstract instructions

push_cell i:int \equiv
if $i+Q \geq \text{MAX_HEAP} \vee H \geq \text{MAX_HEAP}$
 error("heap allocated size exceeded\n");
else {
 heap[H].SORT = TOP;
 heap[H].FTAB = FTAB.DEF_VALUE;
 heap[H].CREF = H;
 setX(i+Q, H++); }

set_sort i:int, s:sort \equiv
 heap[X[i+Q]].SORT = s;

set_feature i:int, f:feat, j:int \equiv
 int addr = deref(X[i+Q]);
 FEAT_frame *frame = heap[addr].FTAB
 add_feature(addr, f, X[j+Q]);

write_test level:int, l:label \equiv
 if $D \geq \text{level}$
 goto R_l;

Abstract ‘program’ code for the term ψ_1

```

R0: intersect_sort 0, a
    test_feature 0, FEAT1, 1, 1, W1, a
    intersect_sort 1, b
R1: goto W2;

W1: push_cell 1
    set_feature 0, FEAT1, 1
    set_sort 1, b

W2:
```

‘READ’ abstract instructions

intersect_sort $i:\text{int}$, $s:\text{sort} \equiv$

```

int addr = deref( X[ i+Q ] ), p;
sort new_sort = glb( s, heap[ addr ].SORT );
if new_sort =  $\perp$ 
  fail = TRUE;
else {
  heap[ addr ].SORT = new_sort; }
```

test_feature $i:\text{int}$, $f:\text{feat}$, $j:\text{int}$, $\text{level}:\text{int}$, $l:\text{label} \equiv$

```

int addr = deref( X[ i+Q ] ), p;
int k = get_feature( addr, f );
if k  $\neq$  FAIL
  X[ j+Q ] = heap[ addr ].FTAB.features[ k ].VAL;
else
  { D = level; goto Wl; }
```

unify_feature $i:\text{int}$, $f:\text{feat}$, $j:\text{int} \equiv$

```

int addr = deref( X[ i+Q ] ), k;
FEAT_frame *frame = heap[ addr ].FTAB;
if (k = (get_feature( addr, f ))  $\neq$  FAIL)
  fail = osf_unify( heap[ addr ].FTAB.feats[ k ].TERM, X[ j+Q ] );
else {
  add_feature( addr, f, X[ j+Q ] ); }
```

3. A simple unification grammar

inspired from Klass Sikkil, “Parsing Schemata”, Springer, 1997, ch. 7

$S \rightarrow NP VP$ $S.head \doteq VP.head$ $VP.subject \doteq NP$ $VP \rightarrow *v NP$ $VP.head \doteq *v.head$ $VP.subject \doteq *v.subject$ $*v.object \doteq NP$ $NP \rightarrow *det *n$ $NP.head \doteq *n.head$ $*n.head.trans \doteq *det.head.trans$

$a \mapsto [cat \Rightarrow *det,$ $head.trans.det \Rightarrow +]$
$the \mapsto [cat \Rightarrow *det,$ $head.trans.det \Rightarrow -]$
$cat \mapsto [cat \Rightarrow *n,$ $head \Rightarrow [agr \Rightarrow [number \Rightarrow singular,$ $person \Rightarrow third],$ $trans.pred \Rightarrow cat]]$
$mouse \mapsto [cat \Rightarrow *n,$ $head \Rightarrow [agr \Rightarrow [number \Rightarrow singular,$ $person \Rightarrow third],$ $trans.pred \Rightarrow mouse]]$
$catches \mapsto [tense \Rightarrow present,$ $head \Rightarrow [agr \Rightarrow [1][number \Rightarrow singular,$ $person \Rightarrow third],$ $trans \Rightarrow [pred \Rightarrow catch,$ $arg1 \Rightarrow [2],$ $arg2 \Rightarrow [3]],$ $subject \Rightarrow [agr \Rightarrow [1],$ $trans \Rightarrow [2],$ $object.head.trans \Rightarrow [3]]$

Translated into OSF syntax, and preparing for the application of Generalised Modus Ponens:

$s(head \Rightarrow V1) \leftarrow$ $V2:np,$ $vp(head \Rightarrow V1,$ $subject \Rightarrow V2)$ $vp(head \Rightarrow V1, subject \Rightarrow V2) \leftarrow$ $v(head \Rightarrow V1,$ $subject \Rightarrow V2,$ $object \Rightarrow V3),$ $V3:np$ $np(head \Rightarrow V1) \leftarrow$ $det(head \Rightarrow top(trans \Rightarrow V2)),$ $n(head \Rightarrow V1(trans \Rightarrow V2))$
--

$det(phon \Rightarrow "the",$ $head \Rightarrow top(trans \Rightarrow top(det \Rightarrow +)))$
$det(phon \Rightarrow "a",$ $head \Rightarrow top(trans \Rightarrow top(det \Rightarrow -)))$
$n(phon \Rightarrow "cat",$ $head \Rightarrow top(agr \Rightarrow top(number \Rightarrow singular,$ $person \Rightarrow third),$ $trans \Rightarrow top(pred \Rightarrow cat)))$
$n(phon \Rightarrow "mouse",$ $head \Rightarrow top(agr \Rightarrow top(number \Rightarrow singular,$ $person \Rightarrow third),$ $trans \Rightarrow top(pred \Rightarrow mouse)))$
$v(phon \Rightarrow "catches",$ $tense \Rightarrow present,$ $head \Rightarrow top(agr \Rightarrow V1(number \Rightarrow singular,$ $person \Rightarrow third),$ $trans \Rightarrow top(pred \Rightarrow catch,$ $arg1 \Rightarrow V2,$ $arg2 \Rightarrow V3)),$ $subject \Rightarrow top(agr \Rightarrow V1,$ $trans \Rightarrow V2),$ $object \Rightarrow top(head \Rightarrow top(trans \Rightarrow V3)))$

Proiect: Implementarea unui parser pentru gramatici de unificare

Task1:

Obținerea reprezentării interne pentru o structură pe trăsături dată sub formă de program abstract ‘query’.

1. (06.11.2009, ora 15)

Să se implementeze în C instrucțiunile abstracte ‘WRITE’ — pentru construirea reprezentării interne a structurilor pe trăsături — conform raportului tehnic al lui Hassan-Aït-Kaci și Roberto Di Cosmo din 1993.

2. (06.11.2009, ora 15)

Să se implementeze în C un program care să ia ca input un fișier ce conține o secvență ‘query’ formată din apeluri de instrucțiuni abstracte (‘WRITE’). Se presupune că această secvență corespunde unei anumite structuri pe trăsături. (A se vedea exemplificarea făcută pentru ψ_1 și ψ_2 la pag. 18.) Programul va executa instrucțiunile în ordinea apariției lor, generând astfel reprezentarea respectivei structuri pe stivele `cell_heap` și `frame_heap`.

Obținere executabil (obligatoriu Linux): `make task1`.

Execuție: `./task1 file-name`.

Task2:

Unificarea a două structuri pe trăsături.

Se consideră un fișier cu structura următoare:

```
declarații de genul  $s_1 : s_2$  (câte una pe linie)
cel puțin o linie blank
cod ‘query’ (pentru FS1)
cel puțin o linie blank
cod ‘query’ (pentru FS2)
```

3. (13.11.2009, ora 15)

Să se implementeze o funcție care pornind de la declarațiile din prima secțiune a unui fișier de genul celui dat mai sus calculează sortul glb (cea mai mare margine inferioară) pentru fiecare pereche de sorturi s_1 și s_2 (inclusiv cazul în care unul dintre acestea este sortul *top* sau sortul *bottom*).

4. (13.11.2009, ora 15)

Implementați funcția `osf_unify` din raportul tehnic al lui Hassan Aït-Kaci și Roberto di Cosmo (1993). (Alternativ, vezi pag. 16 a acestui document.)

Aplicați această funcție pe reprezentările interne obținute pentru termii FS1 și FS2 specificați în format abstract ‘query’ în fișierul `dat`.

5. (13.11.2009, ora 15)

Afișați rezultatul unificării folosind o funcție `pretty_print` (pe care o veți implementa în prealabil). Această funcție ia ca argument un întreg care reprezintă indicele celulei rădăcină a FS de imprimat.

6. (27.11.2009, ora 15)

Implementați o funcție numită `copy_fs(a)` care are un argument (`a`) de tip integer și efectuează copierea (reprezentării pe heap-uri `a`) structurii pe trăsături cu rădăcina dată de celula de indice `a`.

Obținere executabil (obligatoriu Linux): `make task2`.

Execuție: `./task2 file-name [-nc]`.

Observație:

Pentru unificare nedestructivă folosiți funcția `copy_fs` pentru a duplica structurile pe trăsături înainte de unificare; corespunzător, se va folosi opțiunea `-nd` în linia de comandă.

Task3: (27.11.2009, ora 15)

Implementare parser CYK pentru gramatici de unificare.

Obținere executabil (obligatoriu Linux): `make task3`.

Execuție: `./task3 grammar-file-name sentences-file-name`.

Observație:

Pentru test folosiți gramatica dată la pagina 20, în sintaxa OSF.

Notă: În locul simbolului '→' introduceți o trăsătură numită `ARGS` care ia ca valoare o listă formată din argumentele din dreapta regulii (vezi exemplele din slide-urile #10 și #13 despre sistemul `LIGHT`, de pe site-ul meu). Codul 'query' corespunzător regulilor și intrărilor lexicale îl puteți scrie singuri, bazat pe exemplele date anterior. (A se vedea și taskurile 1 și 2.)

Formatul fișierului pe care se dă gramatica va fi:

```
declarații de genul  $s_1 : s_2$  (câte una pe linie)
cel puțin o linie blank
rules: (cuvânt rezervat)
cel puțin o linie blank
cod 'query' pentru regula1, regula2, ... (separate de câte cel puțin o linie blank)
cel puțin o linie blank
lex: (cuvânt rezervat)
cel puțin o linie blank
cod 'query' pentru intr. lexicale 1, 2, ... (separate de câte cel puțin o linie blank)
```

Propozițiile de parsat se vor scrie câte una pe linie în fișierul `sentences-file-name`.

La ieșire se vor furniza parsările fiecărei propoziții (sau mesajul *no parse*) atât în format retrâns (vezi explicațiile de la seminarul din 13.11) cât și în format extins. În formatul extins se vor afișa în sintaxa OSF (folosind funcția `pretty_print` implementată anterior) structurile pe trăsături corespunzătoare simbolului de start, atunci când acesta subîntinde întreaga propoziție de parsat.

Observații generale:

1. Se recomandă să lucrați în echipe de câte 3 persoane. Echipele trebuie să fie stabile (adică să nu își modifice componența) pe perioada întregului proiect. Fiecare membru al echipei va specifica care este task-ul de care este responsabil. Echipele pot dialoga la nivel de idei, dar se interzice cu desăvârșire schimbarea de cod între echipe.
2. Termenul de trimitere a proiectelor (sub forma unei arhive obținută cu `gzip`) este menționat în dreptul fiecărui task. Dacă termenul indicat va fi depășit, punctajul care va fi acordat (de drept) se va înmulți cu factorul 0.75 în prima săptămână de depășire și cu factorul 0 după aceea. **Obiectivele** urmărite sunt: efectuarea corectă, cât mai eficientă și **la termenele fixate** a diferitelor task-uri ale proiectului.
3. Pentru ultima fază a proiectului (termenul de predare: 27.11.2009) nu se mai acordă posibilitatea trimiterii implementărilor după trecerea deadline-ului fixat.