Logical foundations for reasoning about transformations of knowledge bases

Martin Strecker
joint work with Mohamed Chaabani and Rachid Echahed

Université de Toulouse/IRIT

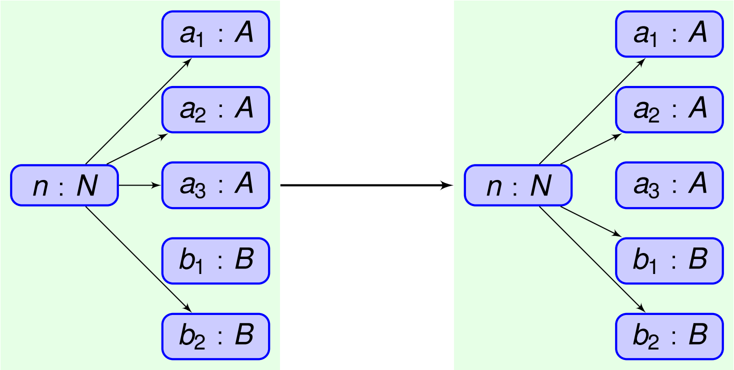
DynRes / CLIMT, 30/5/2013

Plan

Avant-Propos

The slides are in a very preliminary state.
For the accompanying paper and (eventually also) the formal Isabelle development, visit:
`http://www.irit.fr/~Martin.Strecker/Publications/dl_transfo2013.html`

Example: Load balancing (1)



Setup: Routers of categories A and B, communication node $n : N$
Initially: Node n connected to too many nodes of type A
Purpose: Swap some of these connections to nodes of type B

Example: Load balancing (2)

Program transformation:

```
vars n, a, b;

/* Pre: n : (≥ 3 r A) ⊓ (≤ 1 r B) */

while ( n: (> 2 r A) ) do {
        /* Inv: n: (≥ 2 r A) ⊓ (∀ r B) */
        select a sth a : A ∧ (n r a);
        delete(n r a);
        select b sth b : B ;
        add(n r b)
        }
    }
/* Post: n : (= 2 r A) ⊓ (∀ r B) */
```

Approach

Programming language:

- Basis: Imperative programming language
- Conditions: Description logic (DL) formulae
- Generalized assignment statement: `select`

Computing weakest preconditions:

- Yields a formula not directly representable as DL formula
- Therefore: extend DL syntax with new constructor: *explicit substitution*

Deciding weakest preconditions: Tableau calculus interleaving

- traditional DL rules
- “pushing down” explicit substitutions

Plan

Description logics

Traditionally: Family of logics (usually decidable) that are

- sub-languages of FO logic
- variants of modal logics
- cheap forms of set theory
- Distinction between:
  - TBOX (for "terminological" reasoning): involving *concepts* and *roles*
  - ABOX (for "assertional" reasoning): adding individuals

Here: Three levels:

- Concepts ($\approx$ TBOX)
- Facts ($\approx$ ABOX)
- Formulas (Boolean combination of facts, limited quantification)

Substitutions and Concepts

Substitutions:

σ	::=	$[x := y]$	(variable replacement)
	$\mid$	$[r := r - (x, y)]$	(relation substraction)
	$\mid$	$[r := r + (x, y)]$	(relation addition)

Concepts:

C	::=	c	(atomic concept)
	$\mid$	$\neg\ C$	(negation)
	$\mid$	$C \sqcap C$	(conjunction)
	$\mid$	$C \sqcup C$	(disjunction)
	$\mid$	$(\geq n\ r\ C)$	(at least)
	$\mid$	$(< n\ r\ C)$	(no more than)
	$\mid$	$C\ \sigma$	(explicit substitution)

Facts

$$
\begin{array}{llll}
\mathit{fact} & ::= & i : C & \text{(instance of concept)} \\
 & \mid & i\ r\ i & \text{(instance of role)} \\
 & \mid & i\ (\neg r)\ i & \text{(instance of role complement)} \\
 & \mid & i = i & \text{(equality of instances)} \\
 & \mid & i \neq i & \text{(inequality of instances)}
\end{array}
$$

Formulas

$$
\begin{array}{lcll}
\textit{form} & ::= & \bot & \\
& \mid & \textit{fact} & \\
& \mid & \neg \textit{form} & \\
& \mid & \textit{form} \wedge \textit{form} & \mid \quad \textit{form} \vee \textit{form} \\
& \mid & \forall i.\textit{form} & \mid \quad \exists i.\textit{form} \\
& \mid & \textit{form}\ \sigma &
\end{array}
$$

The logic ALC: Syntax

Roles: Here only atomic roles

Concepts:

C, D	$::=$	A	(atomic concept)
	$\mid$	$\top$	(universal concept Top)
	$\mid$	$\bot$	(empty concept Bottom)
	$\mid$	$\neg\ C$	(negation)
	$\mid$	$C \sqcap D$	(conjunction)
	$\mid$	$C \sqcup D$	(disjunction)
	$\mid$	$\forall\ R\ C$	(for all in relation)
	$\mid$	$\exists\ R\ C$	(there are some in relation)

Attention, $\forall$ and $\exists$ are not quantifiers, R not bound in C !

The logic ALC: Semantics (1)

Interpretation $\mathcal{I}$ composed of basic interpretations

- I_c : *conceptname* $\Rightarrow \Delta$ *set*
- I_r : *rolename* $\Rightarrow (\Delta \times \Delta)$ *set*
- I_i : *indivname* $\Rightarrow \Delta$

Interpretation of concepts

$$
\begin{aligned}
\mathcal{I}(A) &= I_c(A) \\
\mathcal{I}(\top) &= \Delta_{\mathcal{I}} \\
\mathcal{I}(\bot) &= \emptyset \\
\mathcal{I}(C \sqcap D) &= \mathcal{I}(C) \cap \mathcal{I}(D) \\
\mathcal{I}(C \sqcup D) &= \mathcal{I}(C) \cup \mathcal{I}(D) \\
\mathcal{I}(\neg C) &= \Delta_{\mathcal{I}} - \mathcal{I}(C) \\
\mathcal{I}(\geq n\ r\ C) &= \{x \mid card\{y \mid (x, y) \in \mathcal{I}(r) \wedge y \in \mathcal{I}(C)\} \geq n\}
\end{aligned}
$$

The logic ALC: Semantics (2)

Interpretation of roles: $R^{\mathcal{I}} = I_r(R)$
Interpretation of substitutions:

$$\mathcal{I}([r := r + (x, y)]) \quad = \quad \mathcal{I} I_r(r) := I_r(r) \cup \{(I_c(x), I_c(y))\}$$

Interpretation of facts:

$$\mathcal{I}(x : C) \quad = \quad I_i(x) \in \mathcal{I}(C)$$

The logic ALC: ABOXes

Idea of ABOXes: Introduce individuals

Syntax: ABOX is finite set of assertions of the form:

- $x : C$, where x is the name of an individual and C a concept
- xRy, where x, y are the names individuals and R is a role

Semantics: evident

Plan

Programs and their semantics (1)

(Big-step) operational semantics: defines transition relation

$$(c, s) \Rightarrow s'$$

between:

- command c
- initial state s
- end state s'

Notion of state:

- Arithmetic programs: $state \equiv var \Rightarrow int$
- (pure) OO programs: $state \equiv addr \Rightarrow obj\ option$, where $obj \equiv field\ list$ and $field \equiv ident \times addr$
- graph programs: *to be discussed*

Programs and their semantics (2)

Programs / commands c usually defined by an abstract syntax / inductive type:

$$
\begin{array}{lcll}
c & ::= & x = e & (x \text{ variable, } e \text{ expression}) \\
& | & c_1 ; c_2 & \\
& | & \texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2 & \\
& | & \texttt{while } e \texttt{ do } c &
\end{array}
$$

Typical rules of the semantics (for arithmetic programs):

$$
\frac{eval(e, s) = v}{(x = e, s) \Rightarrow s(x := v)}
$$

$$
\frac{eval(e, s) \neq 0 \qquad (c, s) \Rightarrow s'' \qquad (\texttt{while } e \texttt{ do } c, s'') \Rightarrow s'}{(\texttt{while } e \texttt{ do } c, s) \Rightarrow s'}
$$

Programs (1)

Basic programs:

basic	::=	$x = \mathtt{new}\ C$	(create new node of C, assign to x)
	|	$\mathtt{delete}(x)$	(delete node)
	|	$\mathtt{delete}(x\ R\ y)$	(delete arc)
	|	$\mathtt{add}(x\ R\ y)$	(add arc)

To be discussed:

- $\mathtt{new}\ C$ for "empty" concept C?
- $\mathtt{delete}(x)$ for linked x?

Programs (2)

Composite programs:

```
prog ::= basic
      |  prog; prog
      |  if form then prog else prog
      |  while form prog
      |  select var sth form in prog
```

Notes:

- `select` *v* `sth` *f* `in` *p* binds *v* in *f* and *p*
- Computation of weakest precondtion is standard for sequence, `if`, `while`
- Needs to be explored for `select` and basic statements.

Syntax

stmt	::=	`Skip`	(empty statement)
	|	`select` *i* `sth` *form*	(assignment)
	|	`delrel(`*i r i*`)`	(delete arc in relation)
	|	`insrel(`*i r i*`)`	(insert arc in relation)
	|	*stmt* `;` *stmt*	(sequence)
	|	`if` *form* `then` *stmt* `else` *stmt*	
	|	`while` *form* `do` *stmt*	

Semantics

$$\frac{}{(\texttt{Skip}, \sigma) \Rightarrow \sigma}\ (\mathit{Skip}) \qquad \frac{(c_1, \sigma) \Rightarrow \sigma'' \quad (c_2, \sigma'') \Rightarrow \sigma'}{(c_1 ; c_2, \sigma) \Rightarrow \sigma'}\ (\mathit{Seq})$$

Semantics

$$\frac{\sigma' = \mathit{delete\_edge}\ v_1\ r\ v_2\ \sigma}{(\texttt{delrel}(v_1\ r\ v_2), \sigma) \Rightarrow \sigma'}\ (\mathit{EDel})$$

$$\frac{\sigma' = \mathit{generate\_edge}\ v_1\ r\ v_2\ \sigma}{(\texttt{insrel}(v_1\ r\ v_2), \sigma) \Rightarrow \sigma'}\ (\mathit{EGen})$$

$$\frac{\exists vi.(\sigma' = \sigma^{[v:=vi]} \wedge \sigma'(b))}{(\texttt{select}\ v\ \texttt{sth}\ b, \sigma) \Rightarrow \sigma'}\ (\mathit{SelAssT})$$

Semantics

$$\frac{\sigma(b) \quad (c_1, \sigma) \Rightarrow \sigma'}{(\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma) \Rightarrow \sigma'} \; (IfT)$$

$$\frac{\neg\sigma(b) \quad (c_2, \sigma) \Rightarrow \sigma'}{(\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma) \Rightarrow \sigma'} \; (IfF)$$

$$\frac{\sigma(b) \quad (c, \sigma) \Rightarrow \sigma'' \quad (\texttt{while } b \texttt{ do } c, \sigma'') \Rightarrow \sigma'}{(\texttt{while } b \texttt{ do } c, \sigma) \Rightarrow \sigma'} \; (WT)$$

$$\frac{\neg\sigma(b)}{(\texttt{while } b \texttt{ do } c, \sigma) \Rightarrow \sigma} \; (WF)$$

Plan

Hoare logics (1)

Reasoning about programs:

1. assertions in a given "background logic" ("shallow embedding")
   $\rightsquigarrow$ might be too expressive (undecidable reasoning)
2. for a dedicated logic ("deep embedding")
   - does this logic attain a sufficiently high precision?
   - is it closed under programming language ops?

Approach 1: Assertion-style reasoning

An assertion is a state predicate (i.e., a set of states):

$assn \equiv (state \Rightarrow bool)$

Example:

$\{x \geq y\}$ `x = x + 2; y = y + 1;` $\{x > y\}$

Here, $\{x \geq y\}$ describes the state set $\{s.(s\ \mathtt{x}) \geq (s\ \mathtt{y})\}$

Hoare logics (2)

Typical Hoare rules:

$$\frac{}{\{Q[x := e]\} x = e \{Q\}}$$

$$\frac{\{P\} c_1 \{R\} \qquad \{R\} c_2 \{Q\}}{\{P\} c_1 ; c_2 \{Q\}}$$

"Weakening":

$$\frac{\{P'\} c \{Q'\} \quad P' \longrightarrow P \quad Q \longrightarrow Q'}{\{P\} c \{Q\}}$$

Shorthand: $Q[x := e] = \lambda s. Q(s(x := eval(e, s)))$
codable in Lambda-calculus. But in less expressive logics?

Hoare logics (3)

Avoid "weakening": To show $\{P\}c\{Q\}$

1. compute *weakest precondition* $wp(c, Q)$
2. show $P \longrightarrow wp(c, Q)$

wp progressively eliminates all program statements:

- $wp(x = e,\ Q) = Q[x := e]$
- $wp(c_1; c_2,\ Q) = wp(c_1,\ wp(c_2,\ Q))$

Example:

- $wp(\mathtt{x} = \mathtt{x} + 2; \mathtt{y} = \mathtt{y} + 1, x > y)$
- $= wp(\mathtt{x} = \mathtt{x} + 2, wp(\mathtt{y} = \mathtt{y} + 1, x > y))$
- $= wp(\mathtt{x} = \mathtt{x} + 2, x > y + 1)$
- $= x + 2 > y + 1$

Now, show $x \geq y \longrightarrow x + 2 > y + 1$

Hoare logics

Approach 2: Deep embedding for dedicated logic

Instead of using an expressive logic: use a restricted (decidable) logic

Questions:

- Is it closed under conditions of `if` and `while`?
- Is the logic closed under basic operations (e.g. assignment)?

Illustration: Assume the propositions of the logic are formed according to the grammar:

e	::=	x	variable
	|	$e + n$	addition of natural number constant n
p	::=	$x = e$	basic propositions

Computing $wp(\mathtt{x = x + 5},\ x = y + 2) = (x + 5 = y + 2)$

which is not well-formed according to the grammar of propositions.

Weakest preconditions

$$
\begin{aligned}
&wp(\texttt{Skip},\ Q) = Q\\
&wp(\texttt{delrel}(v_1\ r\ v2),\ Q) = Q[r := r - (v_1, v_2)]\\
&wp(\texttt{insrel}(v_1\ r\ v2),\ Q) = Q[r := r + (v_1, v_2)]\\
&wp(\texttt{select}\ v\ \texttt{sth}\ b,\ Q) = \forall v.(b \longrightarrow Q)\\
&wp(c_1 ; c_2,\ Q) = wp(c_1, wp(c_2,\ Q))\\
&wp(\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2,\ Q) = ite(b, wp(c_1, Q), wp(c_2, Q))\\
&wp(\texttt{while}\{iv\}\ b\ \texttt{do}\ c,\ Q) = iv
\end{aligned}
$$

Verification conditions

$vc(\texttt{Skip},\ Q) = \top$
$vc(\texttt{delrel}(v_1\ r\ v2),\ Q) = \top$
$vc(\texttt{insrel}(v_1\ r\ v2),\ Q) = \top$
$vc(\texttt{select}\ v\ \texttt{sth}\ b,\ Q) = \top$
$vc(c_1 ; c_2,\ Q) = vc(c_1, wp(c_2, Q)) \wedge vc(c_2, Q)$
$vc(\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2,\ Q) = vc(c_1, Q) \wedge vc(c_2, Q)$
$vc(\texttt{while}\{iv\}\ b\ \texttt{do}\ c,\ Q) = (iv \wedge \neg b \longrightarrow Q) \wedge (iv \wedge b \longrightarrow wp(c, iv)) \wedge v$

Plan

The logic ALC: Tableau calculus

Related inferences:

- Subsumption: $C \sqsubseteq D$, equivalent to $C \sqcap \neg D = \bot$
- Emptyness: $C = \bot$, equivalent to $C \sqsubseteq \bot$

usually reduced to: check satisfiability of ABOX $x : C$, for fresh x

Typical tableau rules: After conversion to negation normal form:

- $x : (C \sqcup D) \rightsquigarrow x : C$ or $x : (C \sqcup D) \rightsquigarrow x : D$
- $x : (C \sqcap D) \rightsquigarrow x : C, x : D$
- $x : (\forall\ R\ C), (xRy) \rightsquigarrow y : C$
- $x : (\exists\ R\ C) \rightsquigarrow (xRy), y : C$
  for fresh y; *provided* these two facts do not yet exist on the branch
- remove contradictory branches: $x : C, x : \neg C$

until model found

Variants of DLs

Number restrictions: concept constructors

- $(\geq n\ R\ C)$ means $\{x.card(\{y.(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}) \geq n\}$
  "the set of all x connected to more than n C-nodes via R"
- $(< n\ R\ C)$ (analogous)

Allow to define the constructors $(\forall\ R\ C)$ and $(\exists\ R\ C)$, for example:

- $(\exists\ R\ C) = (\geq 1\ R\ C)$
- $(\forall\ R\ C) = (< 1\ R\ (\neg C))$

Take some liberty with DL

DL Concepts C: As outlined before: $\top \ldots (< n\ R\ C)$
DL Roles R: Atomic roles: $(x\ r\ y)$ and role negation $(x\ \bar{r}\ y)$
DL Facts *Fact*

fact	::=	$x : C$	(instance of concept)
	|	$x\ R\ y$	(instance of role)
	|	$x = y$	(equality of instances)
	|	$x \neq y$	(inequality of instances)

Note: Facts closed by negation
DL Forms *Form*: Boolean combinations of facts
Examples:

- $x : A \wedge x : B$ is a *Form* equivalent to the fact $x : A \sqcap B$
- $a : A \wedge (n\ r\ a)$ is a *Form* that does not correspond to a DL concept

Weakest preconditions: Relation deletion

What one would like to do:

$\{Q[r := r - (v_1, v_2)]\}$ `delete(` $v_1 \quad r \quad v_2$ `)` $\{Q\}$

But what is $Q[r := r - (v_1, v_2)]$? Is it a DL-formula after all?

Definition by recursion over Q:

- $(P \wedge Q)[r := r - (v_1, v_2)] = P[r := r - (v_1, v_2)] \wedge Q[r := r - (v_1, v_2)]$
- Assuming C does not contain r, and $x \neq v_1$:
  $(x : (< n\ r\ C))[r := r - (v_1, v_2)] = (x : (< n\ r\ C))$
- Assuming C does not contain r, and $x = v_1$ and $v_2 : C$ and $v_1 \ r \ v_2$:
  $(x : (< n\ r\ C))[r := r - (v_1, v_2)] = (x : (< (n + 1)\ r\ C))$

And what if C contains r? Intertwine tableau construction and *wp*-calculus?

Elimination of substitutions (1)

Equality / Inequality:

- $(x = y)[r := re]$ reduces to $(x = y)$
- $(x \neq y)[r := re]$ reduces to $(x \neq y)$

Roles:

- $(x\ r\ y)[r := r - (v_1, v_2)]$ reduces to $(\neg((x = v_1) \wedge (y = v_2))) \wedge (x\ r\ y)$
- $(x\ (\neg r)\ y)[r := r - (v_1, v_2)]$ reduces to $((x = v_1) \wedge (y = v_2)) \vee (x\ (\neg r)\ y)$
- $(x\ r\ y)[r := r + (v_1, v_2)]$ reduces to $((x = v_1) \wedge (y = v_2)) \vee (x\ r\ y)$
- $(x\ (\neg r)\ y)[r := r + (v_1, v_2)]$ reduces to $(\neg((x = v_1) \wedge (y = v_2))) \wedge (x\ (\neg r)\ y)$

Elimination of substitutions (2)

- $(x : \neg C)[r := re]$ reduces to $x : (\neg C[r := re])$
- $(x : (\geq n\ r\ C))[r := r - (v_1, v_2)]$ reduces to

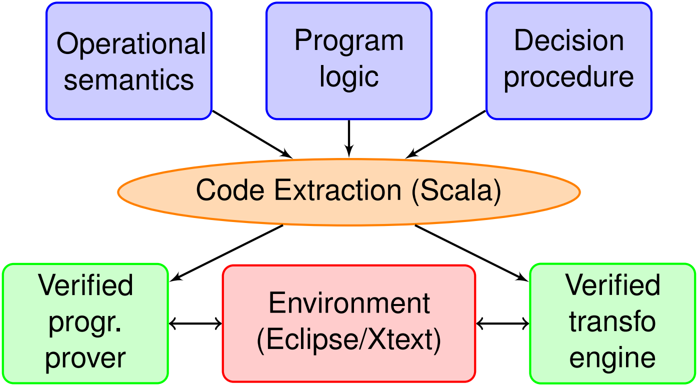
$$\begin{array}{ll} ite & ((x = v_1) \land (v_2 : (C[r := r - (v_1, v_2)])) \land (v_1\ r\ v_2), \\ & (x : (\geq\ (n+1)\ r\ (C[r := r - (v_1, v_2)]))), \\ & (x : (\geq\ n\ r\ (C[r := r - (v_1, v_2)])))) \end{array}$$

and similarly when replacing $\geq$ by $<$

Plan

Pragmatics (1)

Extract a verified transformation engine and program proof environment



Pragmatics (2)

Applications in:

- Model transformations (UML-style): preservation of cardinality restrictions
- Schema updates for expressive data bases
- Transformation of ontologies ($\leadsto$ CIMI working group)

Fundamental questions

Extension of the programming language

- Generalized iterators (map / reduce)
- Procedures (currently only: statements)
- Allow creation and deletion of nodes $\rightsquigarrow$ modeling a heap

Facilitating program proofs:

- Generation of counter-examples out of failed proofs
- Automatic inference of loop invariants
- Automatic derivation of programs out of specifications

More expressive logics

- More expressive role operations: union/ intersection; transitive closure
- Radical departure: realization of MSO-definable transductions?