Iterate Logic

Peter H. Schmitt

Universität Karlsruhe, Institute for Logic, Complexity and Deduction Systems, D-76128 Karlsruhe, Germany. Web: i12www.ira.uka.de/

Abstract We introduce a new logic for finite first-order structures with a linear odering. We study its expressive power. In particular we show that it is strictly stronger than first-order logic on finite structures. We close with a list of open problems.

1 Introduction

The Unified Modeling Language (UML) is a language for visualizing, specifying, constructing and documenting object-oriented software systems. It has been widely accepted as a standard for modeling software systems and is supported by a great number of CASE tools (Computer Aided Software Engineering tools). The standard document on UML is [12]. The Object Constraint Language (OCL) is part of UML [11]. The only available introduction at the time is [18]. The Object Contraint Language (OCL) is used to describe the semantics of UML with greater precision. It is extensively used in the UML meta model. The available descriptions of OCL fall short of giving a rigorous semantics, and even some syntactic details are not clear. Deficiencies have been pointed out e.g. in [14,4,5,10].

One way to remedy this situation and provide OCL with a precise semantics is via a translation into a known, well understood specification language. This approach has been pursued e.g. in [6,3,2,8,9] . As an additional advantage of this approach the translated expressions are frequently used as input to existing tools. The disadvantage is that those not familiar with the target language will gain nothing.

This paper grew out of an effort to give a systematic definition of syntax and semantics of the OCL. This will be pursued elsewhere. Here we aim at a different goal, we try to formulate the logical essence behind OCL. Syntax and semantics of an OCL expression depends stongly on an associated UML class diagram. The work presented here is at a level of abstraction where this dependence is no longer directly visible. We arrive at a new kind of logic and study it in its own right.

2 Syntax and Semantics of L_{it}

In this section we introduce Iterate logic $\mathcal{L}_{it}$ by a series of definitions following the usual pattern. We will define the possible signatures Σ, the set of well-formed terms $\mathcal{T}_{it}(\Sigma)$ and formulas $\mathcal{L}_{it}(\Sigma)$ over Σ, the structures $\mathcal{M}$ and the interpretation of terms and formulas in a given $\mathcal{M}$.

Definition 1
The signature Σ for an Iterate logic consists of

1. *A non-empty set $\mathcal{S}$ of sorts. $Bool$ is always an element of $\mathcal{S}$*
2. *A set C of sorted constant symbols.*
   We use $c : S$ to indicate that c is a constant symbol of sort S.
   The constants $true$ and $false$ of sort $Bool$ are assumed to be available in any signature Σ.
3. *A set F of sorted function symbols.*
   we use $f : S_1 \times \ldots \times S_k \to S$ to indicate that f takes arguments of sort S_1 to S_k, respectively, and output values of sort S. We stipulate further that $S, S_i \in \mathcal{S}$.
4. *Function symbols $r : S_1 \times \ldots \times S_k \to Bool$ of target sort $Bool$ will be called predicate symbols.*
5. *For every sort $S \in \mathcal{S}$ there is a predicate symbol $\doteq_S : S \times S \to Bool$.*
   We will usually just write $\doteq$ instead of $\doteq_S$.
6. *The will always be function symbols*
   - $\wedge : Bool \times Bool \to Bool$
   - $\vee : Bool \times Bool \to Bool$
   - $\Rightarrow : Bool \times Bool \to Bool$
   - $\neg : Bool \to Bool$

   in Σ
7. *There will always be a binary predicate symbol $<$ in Σ.*
   We will later insist that $<$ is always interpreted as a linear order.

There is for every sort S an infinite collection of variables Var_S of sort S.

Definition 2
Terms and their sorts over Σ are defined as follows. Terms of sort $Bool$ will be called formulas.

1. *If c is a constant symbol of sort S then c is a term of sort S,*
2. *For every variable $x \in Var_S$ the expression x is a term of sort S,*
3. *If $f \in F$ is a function symbol with signature $f : S_1 \times \ldots \times S_k \to S$ and $t_1, \ldots , t_k$ are terms of sort $S_1, \ldots , S_k$ respectively, then $f(t_1, \ldots , t_k)$ is a term of sort S.*
4. *If*
   - *sorts $S, T \in \mathcal{S}$,*
   - *$x \in Var_S$, $y \in Var_T$, y different from x,*
   - *t is a formula not containing y,*
   - *t_0 a term of sort T not containing x or y,*
   - *u a term of sort T,*

   then
   $\mathbf{iterate}(x : S; y : T = t_0 \mid u)t$
   is a term of sort T.

5. *If t_1, t_2 are terms of the same sort then we write $t_1 \doteq t_2$ instead of $\doteq(t_1, t_2)$. Also the propositional connectives will be written in infix notation, i.e. $t_1 \wedge t_2$ instead of $\wedge(t_1, t_2)$.*

Since the syntax of iterator terms involves five constituent entities it pays to agree on some terminology. Using the notation from clause 4 above, the variable x is called the *iterator variable*, y the *accumulator variable*, t is termed the *range formula*, u the *step term* and t_0 the *initial term*.

Definition 3
A structure $\mathcal{M}$ of Σ consists of a non-empty domain M and an interpretation function I such that

1. *$I(S)$ is a non-empty subset of M for every $S \in \mathcal{S}$. The universe M of $\mathcal{M}$ is the disjoint union of all $I(S)$, $S \in \mathcal{S}$.*
2. *$I(c) \in I(S)$ for every $(c : S) \in C$.*
3. *for every $(f : S_1 \times \ldots \times S_k \to S) \in F$ the interpretation $I(f)$ is a function from $I(S_1) \times \ldots \times I(S_k)$ into $I(S)$.*
4. *$I(\doteq)$ is the equality relation and the Boolean connectives will have their usual meaning.*
5. *$I(<)$ is a total linear order on M. Instead of $I(<)$ we will by abuse of notation write $<$ or $<_{\mathcal{M}}$ if need arises.*

Usually we will just write I for the interpretation function. If necessary we will write, more precisely, $I_{\mathcal{M}}$.

We note that Definition 3 is for the greatest part the usual definition of many-sorted structures. The only particularities are the insistence on having a linear order available, a phenomenon familiar to those working in the area of characterizing complexity classes via logical definability, and the fact that we will only consider finite structures.

Definition 4
Let $\mathcal{M} = (M, I)$ be a finite structure.

The interpretation function I is extended to terms and formulas. As usual we need valuations $\beta : Var_S \to I(S)$ of free variables for all $S \in \mathcal{S}$.

For a term t of sort $S \in \mathcal{S}$ $I_\beta(t)$ will be an element of $I(S)$.

1. *if $t \in Var_S$ then $I_\beta(t) = \beta(t)$,*
2. *$I_\beta(f(t_1, \ldots, t_k)) = I(f)(I_\beta(t_1), \ldots, I_\beta(t_k))$.*
3. *the interpretation $m = I_\beta(\mathbf{iterate}(x : S; y : T = t_0 \mid u)t)$ is computed as follows.*
   Let $\{a_1, \ldots, a_n\} = \{a \in I(S) \mid I_{\beta_x^a}(t) = true\}$ with $a_1 < \ldots < a_n$.
   We define m_i for $0 \leq i < n$ by:
   (a) $m_0 = I_\beta(t_0)$
   (b) $m_{i+1} = I_{\beta^{i+1}}(u)$
   where

$$\beta^{i+1}(z) = \begin{cases} \beta(z) \text{ if } z \neq x, y \\ a_{i+1} \text{ if } z = x \\ m_i \quad \text{if } z = y \end{cases}$$

(Observe that $m_i \in I(T)$ for all i.)

Now we set $m = m_n$.

4. *$I_\beta(t_1 \doteq t_2) = true$ iff $I_\beta(t_1) = I_\beta(t_2)$.*
5. *$I_\beta(A)$ for A a propositional composition of smaller formulas is defined as usual.*

Remark: If in the computation of $I_\beta(\mathbf{iterate}(x : S_1; y : T = t_0 \mid u)t)$ the set $\{a \in I(S_1) \mid I_{\beta_x^a}(t) = true\}$ is empty, then we have, according to the above definition, $I_\beta(\mathbf{iterate}(x : S_1; y : T = t_0 \mid u)t) = I_\beta(t_0)$.

Example To get familiar with Definition 4 let us evaluate the meaning of the formula $\mathbf{iterate}(x : S; y : Bool = true \mid y \wedge u)(true)$ in structure $\mathcal{M} = (M, I)$. In addition to the restrictions on the variables imposed by the syntax definition of **interate**-terms, we require that y does not occur in u. The range formula being $true$ we obtain as the basis of iteration the set of all elements of sort S in M, say $a_1 < \ldots < a_n$. The initial value m_0 of the accumulator is the Boolean value $true$. For the first step in the iteration we compute $m_1 = I_{\beta^1}(y \wedge u)$, where $\beta^1(x) = a_1$ and $\beta^1(y) = true$. In other words $m_1 = true$ if $\mathcal{M} \models u[a_1]$ and $m_1 = false$ otherwise. We also notice that once the accumulator is assigned the value $false$ it will be stuck with this value for the rest of the iteration. Thus the final value m_n of the accumulator is $true$ if for all a_i $\mathcal{M} \models u[a_i]$ holds true and $false$ otherwise. Summing up, we may use $\mathbf{iterate}(x : S; y : Bool = true \mid y \wedge u)(true)$ as a definition of the universal quantification $\forall x_S u$.

Likewise $\mathbf{iterate}(x : S; y : Bool = false \mid y \vee u)(true)$ behaves as $\exists x_S u$.

Definition 5

Let t, s be formulas in $\mathcal{L}_{it}$, $\mathcal{M} = (M, I)$ a finite structure.

1. *t is called* valid *in $\mathcal{M}$, in symbols $\mathcal{M} \models t$, if for all β $I_\beta(t) = true$.*
2. *t is called* universally valid*, in symbols $\models t$, if it is valid for all finite structures $\mathcal{M}$.*
3. *t is* a logical consequence *of s, in symbols $s \models t$, if for any finite structure $\mathcal{M}$ such that s is valid in $\mathcal{M}$ also t is valid in $\mathcal{M}$.*

For formulas, i.e. terms t of sort Boolean we will also use notations like $(\mathcal{M}, \beta) \models t$ instead of $I_\beta(t) = true$, or even $\mathcal{M} \models t[a_1, \ldots, a_k]$, where $x_1, \ldots, x_k$ are all free variables in t (listed in some fixed order) and $\beta(x_i) = a_i$.

Definition 6

We obtain (a syntactic variant of) first-order logic $\mathcal{L}_{FO}$ with a linear order if we replace item 4 in Definition 2 by

4. *If t is a formula, x a variable of sort S then also $\forall xt$ and $\exists xt$ are formulas.*

and leave all other clauses unchanged.

3 Expressive Power of L_{it}

Having formulated Iterate Logic in the previous section we will now start collecting facts on its expressive power. We will use the following concept for comparing different logics with respect to their expressive power. We will only consider logics that use as interpretations structures of the kind defined in Definition 3. This is no severe restriction since it is met by most logics we are interested in, e.g. first-order logic, (monadic) second-order logic, fixed-point logics, transitive closure logics.

Definition 7
We say, logic $\mathcal{L}_2$ is as a expressive as $\mathcal{L}_1$, in symbols

$$\mathcal{L}_1 \leq \mathcal{L}_2$$

if for every formula t_1 in $\mathcal{L}_1$ there is a formula t_2 in $\mathcal{L}_2$ such that for all finite structures $\mathcal{M}$ and variable assignments β

$$(\mathcal{M}, \beta) \models_1 t_1 \text{ iff } (\mathcal{M}, \beta) \models_2 t_1$$

$\mathcal{L}_2$ is strictly more expressive than $\mathcal{L}_1$, in symbols $\mathcal{L}_1 < \mathcal{L}_2$ if $\mathcal{L}_1 \leq \mathcal{L}_2$ and $\mathcal{L}_2 \not\leq \mathcal{L}_1$.

The example following Definition 4 establishes $\mathcal{L}_{FO} \leq L_{it}$. We will eventually show that L_{it} is strictly more expressive than $\mathcal{L}_{FO}$. To prepare the ground we prove two simple definability lemmas for L_{it}.

Lemma 1. *For any formula t and terms t_1, t_2 of sort $S \in \mathcal{S}$ there is an L_{it}-term s, such that*

$$s = (\mathit{if}\ t\ \mathit{then}\ t_1\ \mathit{else}\ t_2)$$

Proof: Set $s = \mathbf{iterate}(x : S; y : S = t_2 \mid t_1)t$ where the variables x, y are different and do both not occur in t, t_1 and t_2. In any structure $\mathcal{M} = (M, I)$ we have for all β

$$\{a \in I(S) \mid I_{\beta^a_x}(t) = true\} = \begin{cases} M \text{ if } I_\beta(t) = true \\ \emptyset \ \text{ if } I_\beta(t) = false \end{cases}$$

From this we immediately get

$$I_\beta(s) = \begin{cases} I_\beta(t_1) \text{ if } I_\beta(t) = true \\ I_\beta(t_2) \text{ if } I_\beta(t) = false \end{cases}$$

∎

Lemma 2. *For any function symbol $f \in \Sigma$ there is a formula $s(x,y)$ in L_{it}, such that for any structure $\mathcal{M} = (M, I)$, any β, and any elements $a, b \in M$*

$$I_{\beta^{a\,b}_{x\,y}}(s) = true$$
$$\Leftrightarrow$$
$$\textit{there is } n \in \mathbb{N} \textit{ with } n \leq \textit{ card}(M) \textit{ such that } b = I_\beta(f)^n(a)$$

Proof: Set $s = \exists w((\mathbf{iterate}(z; v = x \mid u)true) \doteq y)$
where $\quad u = \text{if } (z < w) \text{ then } f(v) \text{ else } v$

■

Lemma 2 already suggests that $\mathcal{L}_{it}$ is strictly stronger than first-order logic even on finite structures. But we can do better.

For the remainder of this argument, which will come to a conclusion with the proof of Lemma 3, we fix a particular family of signatures Σ_A. The parameter A is being looked at as an alphabet, as it is used in formal language theory. For the alphabet A we denote, as usual, the set of all words over A by A^*. The signature Σ_A contains the binary relation symbol $<$, a constant symbol first, a unary, postfix function $+1$ and for every $a \in A$ a unary relation symbol $Q_a()$. It does not hurt to assume in addition that Σ_A also contains a constant last and a function symbol -1.

Furthermore let Φ_A be the formula stating that

1. $<$ is a strict linear ordering with first and last element,
2. $+1$ is the successor function on non-maximal elements m and $m+1 = m$ for the greatest element
3. -1 is the predecessor function on non-minimal elements and first $-1 =$ last,
4. for every position p there is exactly one $a \in A$ such that $Q_a(p)$ is true.

Note, the asymmetric behaviour of $+1$ and -1 at the endpoints. Every finite model of Φ_A has an isomorphic copy whose elements are the natural numbers from 0 to some n. We will only consider models of Φ_A in this restricted sense. Then, the set of finite non-empty words $A^* \setminus \{\Lambda\}$ is in one-to-one correspondence with the finite models of Φ_A. For $w \in A^* \setminus \{\Lambda\}$ we denote the associated Σ_A structure by $\mathcal{M}_w$. We quickly recall the relationship between w and $\mathcal{M}_w$. Full descriptions are contained in the literature quoted in the next paragraph. Let $0, \ldots, n-1$ be all the elements of $\mathcal{M}_w$. The cardinality n of $\mathcal{M}_w$ equals the length of the word w. The elements i are thought of as the positions in the word w. $\mathcal{M}_w \models Q_a(0)$ holds true if and only if the first letter in w equals a, $\mathcal{M}_w \models Q_b(1)$ iff the second letter of w equals b and so on. We have to exclude the empty word, Λ, since in our setting the empty model $\mathcal{M}_\Lambda$ is not possible.

The following formula t_{anbn} in signature Σ_A for $A = \{a, b\}$.

$$
\begin{aligned}
&\exists x(\ \text{first} < x \wedge x < \text{last} \wedge \\
&\qquad (\mathbf{iterate}(z; v = (if\ Q_a(\text{first})\ \mathbf{then}\ \text{first}\ \mathbf{else}\ \text{last}) \mid u)true) \doteq \text{first}) \\
&\text{where} \\
&u = if(z \leq x \wedge Q_a(z))\,\text{then}\,z\,\text{else} \\
&\qquad (if(z \leq x \wedge \neg Q_a(z))\,\text{then}\,\text{last}\,\text{else} \\
&\qquad (if(x < z \wedge \neg Q_a(z))\,\text{then}\,\text{last}\,\text{else} \\
&\qquad (if(x < z \wedge Q_a(z))\,\text{then}\,v - 1)))
\end{aligned}
$$

will play the key role in the proof to follow.

Lemma 3.
For every $w \in \{a, b\}^*$, $w \neq \Lambda$ and its corresponding model $\mathcal{M}_w$ of Φ_A:

$$\mathcal{M}_w \models t_{anbn} \quad \Leftrightarrow w \in aa^n bb^n$$

Proof of the Lemma 3
We first assume $\mathcal{M}_w \models t_{anbn}$. There is thus a natural number n, first $= 0 < n <$ last $= k$ such that

$$m = I_\beta(\mathbf{iterate}(z; v = \text{first} \mid u)true) = \text{first}$$

with $\beta(x) = n$. Unravelling this according to Definition 4 we get for all $0 \leq i \leq k$:

$$m_i = \begin{cases} i & \text{if } i \leq n \wedge Q_a(i) \\ k & \text{if } i \leq n \wedge Q_b(i) \\ i-1 & \text{if } i > n \wedge Q_b(i) \\ k & \text{if } i > n \wedge Q_a(i) \end{cases}$$

We first observe that if $m_i = k$ for some i then $m = m_k$ can never be 0, as claimed. Thus we have vor all $i \leq n$ $Q_a(i)$ and for all $i > n$ $Q_b(i)$. Given that, we see that $m = n - (k - n) = 0$, i.e. $k = 2n$. Note that the stipulation first $-1 =$ last is important here (first $-1 =$ first would not work). So, we have seen that w is of the form $a^n b^n$ with $n > 0$.

For the reverse implication of the claim, we start with a word w of the form $a^n b^n$ with $n > 0$ and want to show $\mathcal{M}_w \models t_{anbn}$. As a witnessing element for the leading existential quantifier we use n. The rest is straight forward.

∎

The question which classes of words can be defined in first-order logic and in monadic second-order logic has been intensively investigated, surveys may be found e.g. in [16,17,13]. By a well-known result of Büchi, the sets of finite words definable by monadic second-order logic are precisely the regular sets of words. Now, $aa^n bb^n$ denotes a context-free but not regular language, thus Lemma 3 tells us that $\mathcal{L}_{it} \not\leq \mathcal{L}_{MonSO}$. We have already observed that $\mathcal{L}_{FO} \leq \mathcal{L}_{it}$ is true. Thus we also get $\mathcal{L}_{FO} < \mathcal{L}_{it}$.

We compare $\mathcal{L}_{it}$ with the transitive closure logic $\mathcal{L}_{tc}$, that has been introduced by Neil Immerman in [7, Section 3].

Definition 8
The terms of $\mathcal{L}_{tc}$ are defined as in Definition 2 exept that clause 4 is replaced by

4. *If t is a term of sort S and s is a term of sort Bool, $x_1, \ldots, x_k, y_1, \ldots, y_k$ a sequence of distinct variables, and R a relational variable not free in s, then*

$$[R(x_1, \ldots, x_k, y_1, \ldots, y_k) \equiv TC(s)]t$$

is also a term in $\mathcal{L}_{tc}$ and of sort S.

Definition 9
Let $\mathcal{M} = (M, I_\mathcal{M})$ be a finite structure, β a variable assignment.

In order to define the interpretation of $\mathcal{L}_{tc}$-terms in $\mathcal{M}$ we only need to deal with the TC-construct $[R(\boldsymbol{x}, \boldsymbol{y}) \equiv TC(s)]t$. Here $\boldsymbol{x}$ is shorthand for the tupel $x_1, \ldots, x_k$ of variables, likewise $\boldsymbol{y}$ abbreviates the tupel $y_1, \ldots, y_k$ of the same length. $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ are $2k$ different variables.

We first observe that s introduces a binary relation R_s^β on the set M^k via the definition

$$R_s^\beta(\boldsymbol{a}, \boldsymbol{b}) \Leftrightarrow (\mathcal{M}, \beta) \models s[\boldsymbol{a}, \boldsymbol{b}]$$

Again $\boldsymbol{a}$ abbreviates the tupel $a_1, \ldots, a_k$, and $\boldsymbol{b}$ the tupel $b_1, \ldots, b_k$. Furthermore we made use of $(\mathcal{M}, \beta) \models s[\boldsymbol{a}, \boldsymbol{b}]$ as a suggestive shorthand for $(\mathcal{M}, \beta') \models s$ with

$$\beta'(u) = \begin{cases} a_i & \textit{if } u = x_i \\ b_i & \textit{if } u = y_i \\ \beta(u) & \textit{otherwise} \end{cases}$$

(Compare the remark following Definition 5.)
Then

$$I_{\mathcal{M},\beta}([R(x_1, \ldots, x_k, y_1, \ldots, y_k) \equiv TC(s)]t) = I_{\mathcal{M}^*,\beta}(t)$$

where $\mathcal{M}^*$ coincides with $\mathcal{M}$, but in addition $I_{\mathcal{M}^*}(R)$ is the transitive closure of the relation R_s^β. Notice, that the relation symbol R may occur free in t.

Remark: The superscript β in R_s^β is neccessary, since $\boldsymbol{x}, \boldsymbol{y}$ may not be all the free variables of s.

Lemma 4.

$$\mathcal{L}_{it} \leq \mathcal{L}_{tc}$$

Proof: We have to produce for every $\mathcal{L}_{it}$-formula r an $\mathcal{L}_{tc}$-formula r^{tc} such that for all structures $\mathcal{M}$ and variable assignments β

$$(\mathcal{M}, \beta) \models_{it} r \Leftrightarrow (\mathcal{M}, \beta) \models_{tc} r^{tc}$$

We will in fact construct for every $\mathcal{L}_{it}$-term r a $\mathcal{L}_{tc}$-formula $\bar{r}$, containing a new additional free variable z such that for all structures $\mathcal{M} = (M, I)$ and variable assignments β

$$I_\beta(r) = a \Leftrightarrow (\mathcal{M}, \beta_z^a) \models_{tc} \bar{r} = \mathit{true}$$

If r is a formula, i.e. a term of sort Boolean, then $r^{tc} = \bar{r}(\mathit{true}/z)$ yields the desired translation.

Naturally $\bar{r}$ will be constructed by structural induction. To reduce technical complications, we use always the same reserved variable z as the new variable in $\bar{r}$ for any r and assume that z does not occur among the variables in $\mathcal{L}_{it}$.

If r is simply a functional term $r = f(x_1, \ldots, x_n)$ then $\bar{r} = (z \doteq f(x_1, \ldots, x_n))$ The only challenging induction step is the case

$$r = \textbf{iterate}(x : S; y : T = t_0 \mid u)t.$$

By induction hypothesis there are $\mathcal{L}_{tc}$-formulas $\bar{t_0}$ and $\bar{u}$ such that for all $\mathcal{M} = (M, I)$ and β

$$I_\beta(t_0) = a \Leftrightarrow (\mathcal{M}, \beta_z^a) \models_{tc} \bar{t_0} = \mathit{true}$$
$$I_\beta(u) = a \Leftrightarrow (\mathcal{M}, \beta_z^a) \models_{tc} \bar{u} = \mathit{true}$$

Since t has to be a term of sort Boolean the induction hypothesis provides us with a $\mathcal{L}_{tc}$-formula t^{tc} such that for all $\mathcal{M}$ and β

$$(\mathcal{M}, \beta) \models_{it} t \Leftrightarrow (\mathcal{M}, \beta) \models_{tc} t^{tc}$$

As a first step towards the construction of the $\mathcal{L}_{tc}$-formula $\bar{r}$ consider the formula

$$
\begin{array}{ll}
e(x_1,x_2,y_1,y_2) = \exists x t^{tc} \wedge [& \\
& (first_t(x_1) \wedge next_t(x_1,y_1) \wedge \bar{t_0}(x_2/z) \wedge \bar{u}(y_1/x,x_2/y,y_2/z)) \\
& \vee \\
& (\neg first_t(x_1) \wedge next_t(x_1,y_1) \wedge \bar{u}(y_1/x,x_2/y,y_2/z)) \\
&]
\end{array}
$$

with

$$
\begin{array}{ll}
first_t(x_1) & = t^{tc}(x_1/x) \wedge \neg\exists x(t^{tc} \wedge x < x_1) \\
last_t(x_1) & = t^{tc}(x_1/x) \wedge \neg\exists x(t^{tc} \wedge x_1 < x) \\
next_t(x_1,y_1) & = t^{tc}(x_1/x) \wedge t^{tc}(y_1/x) \wedge x_1 < x_2 \wedge \neg\exists x(t^{tc} \wedge x_1 < x < x_2)
\end{array}
$$

For any terms w, s_1, s_2 the term $w(s_1/x,s_2/y)$ denotes the term obtained from w by replacing simultaneously x by s_1 and y by s_2.

The looked for translation of r can now we define as

$$
\begin{array}{l}
\bar{r} = (\neg\exists x t^{tc} \wedge \bar{t_0}) \vee \\
\quad (\neg\exists z_1,z_2(t^{tc}(z_1/x) \wedge t^{tc}(z_2/x) \wedge z_1 \not\equiv z_2) \wedge \exists x \exists w(t^{tc} \wedge \bar{t_0}(w/z) \wedge \bar{u}(w/y))) \vee \\
\quad \exists z_1,z_2(t^{tc}(z_1/x) \wedge t^{tc}(z_2/x) \wedge z_1 \not\equiv z_2) \wedge \\
\quad [R(\boldsymbol{x},\boldsymbol{y}) \equiv TC(e)](\exists z_1,z_2,z_3(first_t(z_1) \wedge last_t(z_2) \wedge \bar{t_0}(z_3/z) \\
\qquad\qquad\qquad \wedge R(z_1,z_3,z_2,z)))
\end{array}
$$

Let $\mathcal{M}$ be a finite structure and β a variable assignment. The three disjunctive components of $\bar{r}$ correspond to the cases that there

1. is no element satisfying the translated range formula t^{tc},
2. is exactly one element satisfying the translated range formula t^{tc},
3. are at least two elements satisfying the translated range formula t^{tc}.

Obviously the first two cases, have been properly dealt with. We assume from now on $(\mathcal{M},\beta) \models \exists z_1,z_2(t^{tc}(z_1/x) \wedge t^{tc}(z_2/x) \wedge z_1 \not\equiv z_2)$. We will use the notation from Definition 4, clause 3 for the evaluation of $\mathbf{iterate}(x:S;y:T=t_0 \mid u)t$ in $(\mathcal{M},\beta)$. Let further r_e denote the transitive closure of the relation induced by e on M^2. By the definition of e we have for any i, $1 \le i < n$ and any pair (b_1,b_2) in M^2:

$$(\mathcal{M},\beta) \models e[a_i,m_i,b_1,b_2] \text{ iff } b_1 = a_{i+1}, b_2 = m_{i+1}$$

This implies for all $(b_1,b_2) \in M^2$:

$$r_e(a_1,m_0,b_1,b_2) \text{ iff } b_1 = a_i \text{ and } b_2 = m_i \text{ for some } i, 1 \le i \le n$$

From this it is easy to see that the formula $[R(\boldsymbol{x},\boldsymbol{y}) \equiv TC(e)](\exists z_1,z_2,z_3(first_t(z_1) \wedge last_t(z_2) \wedge \bar{t_0}(z_3/z) \wedge R(z_1,z_3,z_2,z)))$ is true in $(\mathcal{M},\beta)$ if and only if $\beta(z) = m_n$.

∎

Open Questions

We conclude with a set of open questions, whose answers will shed more light on the status of the new logic $\mathcal{L}_{it}$. All questions concern finite structures only unless explicitely mentioned otherwise.

1. Can transitive closure be definied in $\mathcal{L}_{it}$?
2. Compare $\mathcal{L}_{it}$ to the Dynamic Logic whose programs are for-programs. This necessitates to adjust Definition 7, since the Kripke structures needed for the interpretation of formulas in Dynamic logic are different from the structures introduced in Definition 3.
3. Compare $\mathcal{L}_{it}$ to logics with a least-fix-point operator.

One wonders if in Definition 2 one could not have introduced the iterate operator with more than one accumulator, i.e. an operator of the form $\mathbf{iterate}(x; y^1 = t^1, \ldots, y^k = t^k \mid u^1, \ldots, u^k)t$. The result of this operator will be read off from the first accumulator. Formally we modify clause 3 of Definition 4 as follows:

The interpretation $m = I_\beta(\mathbf{iterate}(x; y^1 = t^1, \ldots, y^k = t^k \mid u^1, \ldots, u^k)t)$ is computed as follows.

Let $\{a_1, \ldots, a_n\} = \{a \in I(S_1) \mid I_{\beta^a_x}(t) = true\}$ with $a_1 < \ldots < a_n$.

We define m^j_i for $1 \leq j \leq k$, $0 \leq i < n$ by:

1. $m^j_0 = I_\beta(t^j)$
2. $m^j_{i+1} = I_{\beta^{i+1}}(u^j)$
   where
   $$\beta^{i+1}(z) = \begin{cases} \beta(z) & \text{if } z \neq x, y \\ a_{i+1} & \text{if } z = x \\ m^j_i & \text{if } z = y^j \end{cases}$$

Now we set $m = m^1_n$.

Let us denote by L_{it-n} the logic which allows iterators with k accumulators for all $k \leq n$. Thus $L_{it-1} = L_{it}$

5. Is L_{it-n+1} more xpressive than L_{it-n}?

6. This is more a task than a question. The addition of a linear order to all structures strongly influences the expressive power of $\mathcal{L}_{it}$. Are there ways to avoid this?
7. We have restricted the semantics of $\mathcal{L}_{it}$ to finite structures, mainly for the reason that otherwise iterate-terms may not terminate. Extend the definition of $\mathcal{L}_{it}$ by allowing terms with undefined values and waive the finiteness restriction.

References

1. ACM, editor. *Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA '99*, volume 34 (10) of *ACM SIGPLAN notices*. ACM, ACM Press, 1999.

2. R. France. A problem-oriented analysis of basic uml static modeling concepts. In ACM [1].
3. M. Gogolla and M. Richters. On constraints and queries in UML. In Schader and Korthaus [15], pages 109–121.
4. A. Hamie. A formal semantics for checking and analysing UML models. In L. Andrade, A. Moreira, A. Deshpande, and S. Kent, editors, *Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How?*, 1998.
5. A. Hamie, J. Howse, and S. Kent. Navigation expressions in object-oriented modelling. In *FASE'98*.
6. A. Hamie, J. Howse, and S. Kent. Interpreting the object constraint language. In *Proceedings of Asia Pacific Conference in Software Engineering*. IEEE Press, July 1998.
7. N. Immerman. Languages that capture complexity classes. *SIAM J. of Computing*, 16(4):760–778, 1987.
8. S.-K. Kim and D. Carrington. Formalizing the UML class diagram using Object-Z. volume 1723 of *LNCS*, pages 83–98. Springer, 1999.
9. P. Krishnan. Consistency Checks for UML. In *Proceedings of the Asia Pacific Software Engineering Conference (APSEC 2000)*, pages 162–169, December 2000.
10. L. Mandel and M. V. Cengarle. On the expressive power of OCL. In *FM'99 - Formal Methods. World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 1999. Proceedings, Volume I*, volume 1708 of *LNCS*, pages 854–874. Springer, 1999.
11. OMG. Object constraint language specification, version 1.3. chapter 7 in [12]. OMG Document, March 2000.
12. OMG. OMG unified modeling language spezification, version 1.3. OMG Document, March 2000.
13. J.-E. Pin. Logic on words. *Bulletin of the EATCS*, 54:145–165, 1994.
14. M. Richters and M. Gogolla. On formalizing the UML object constraint language OCL. In T. W. Ling, S. Ram, and M. L. Lee, editors, *Proc. 17th Int. Conf. Conceptual Modeling (ER'98)*, pages 449–464. Springer, Berlin, LNCS 1507, 1998.
15. M. Schader and A. Korthaus, editors. *The Unified Modeling Language: technical aspects and applications*. Physica-Verlag, 1998.
16. H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Progress in Theoretical Computer Science. Birkhäuser, 1994.
17. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Vol. B : Formal Models and Semantics*, pages 135–192. Elsevier, Amsterdam, 1990.
18. J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modelling with UML*. Object Technology Series. Addison-Wesley, Reading/MA, 1999.