An instantiation algorithm for TLA+ expressions

March 2, 2017

1 Overview

TLA+ has two kinds of substitution: instantiation of modules, which preserves validity, and beta-reduction of lambda-expressions, which does not necessarily preserve validity. Following the notions of SANY, we distinguish three kinds of variables: temporal constants and temporal variables are bound by instantiation whereas formal parameters are bound by lambda abstraction. Furthermore, ENABLED binds all temporal variables within its argument which are in the context of the next state operator '.

In the presence of unresolved instantiations or substitutions, it is often unclear, which operator binds the occurrence of a particular temporal variable: when we consider the expression $(\lambda\ fp : \mathit{ENABLED}\ (fp \# fp'))(x)$, it seems that the variable x is bound in the module. But in the beta-reduced form $\mathit{ENABLED}\ (x \# x')$ of this expression, only the first occurrence of x is bound by the module but its second (primed) occurrence is bound by ENABLED[1].

Furthermore, the two substitutions do not commute. For example, let us consider modules Foo and Bar:

```
---- MODULE Foo ----

VARIABLE x

E(u) == x' # u'
D(u) == ENABLED ( E(u) )

THEOREM T1 == D(x)

====
```

```
---- MODULE Bar ----

VARIABLE y

I == INSTANCE Foo with x <- y

THEOREM T2 == I!D(y)

====
```

It looks like I!T1 and T2 talk about the same formula D(y), but this is not the case: the first can be read as I!(D(y))[2] and the second as (I!D)(y). In other words, it makes a difference if we beta-reduce first or if we instantiate first. Reducing D(x) first leads to ENABLED (x' # x'), which – following the renaming instuctions in "Specifying Systems" – becomes ENABLED ($x' # $x') by the instantiation of x with y, because primed occurrences of $x are bound by their enclosing ENABLED, not by the instantiation.

Instantiating first keeps the occurrence of the formal parameter u intact, leading to I!D(u) == ENABLED (u # $x') but reducing the application I!D(y) leads to ENABLED (y' # $x'). Now it is clear that I!(D(y)) is unsatisfiable while (I!D)(y) is valid.[3].

In the following, we will develop algorithms for both kinds of substitutions. Since the occurrence of a lambda reducible expression (redex) as a subterm of an instantiation may block the evaluation of the latter, inner substitutions (i.e. those closer to the leaves of the term tree) must be evaluated before outer ones. Definitions will also need special consideration because we allow some of them to stay folded. But since a definition can contain substitutions, they can not be treated like temporal constants.[4]

[1]To prevent complications from renaming, we will not assume alpha equivalence but will handle the renaming of bound variables explicitly.

[2]This is not valid TLA+ syntax.

[3]Since one of the axioms of TLA+ is that (TRUE # FALSE), we can always find a domain element different from x. In fact, the axioms of set-theory even enforce denumerable models.

[4]Actually, already a definition D(x) == e contains a substitution because it is equivalent to D == LAMBDA x : e.

2 New Attempt: Explicit substitutions

The original idea here is to represent both beta-reduction and instantiation explicitly in the term graph. Then the two formulas in the introduction could be written as D(u){u ↦ x}[x ↦ y] and D(u)[x ↦ y]{u ↦ y}, where reduction is denoted by curly braces and instantiation is denoted by square braces. Actually, the SANY data-structures allow to write reduction as application to an abstraction: D(u){u ↦ x} is then just (LAMBDA u: D(u))(x). Again, this is not legal TLA+ but allowed by SANY. Nevertheless, having explicit substitution nodes drastically simplifies the formal treatment of the behavior[5].

2.1 Datastructures

node	content	comment
Module	list of constants, list of variables, list of instances, list of definitions	
(Temporal) Constant	name, arity	Set of Constants CS
(Temporal) Variable	name	arity == 0, Set of Variables VS
(Formal)Parameter	name, arity	Set of Paramters FP
Definition	name, arity, expression body	Set of Definitions DS, all FPs in body are bound
Expression	constant **or** variable **or** parameter **or** definition **or** abstraction **or** application **or** substin **or** fpsubstin	
Application	head expression, argument expression list	head.arity == list length
Abstraction	parameter, expression body	
FPSubstIn	list of pairs formal parameter, expression	explicit substitution node
SubstIn	module, instantiation, expression body	the explicit instantiation node
Instantiation	list of assigments of variables/constants to expressions	

The definition and instantiation elements do not contain arguments since they can always be rewritten in terms of abstractions: D(x) == F is equivalent to D == LAMBDA x : F and I(x)!D is equivalent to LAMBDA x : I!D. We write abstraction as $\lambda x : F$, explicit substitutions as $\sigma = \{x \leftarrow t\}$ and explicit instantiations as $[M : x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$ where the variables / constants x_i are exactly those declared in the module M.

To allow for more flexibility, we state the algorithm as a set of rewrite rules. The goal is to permute explicit substitutions further inside the term and resolve them at the leaves of the term tree (rules 1, 2, 9 and 10) or, if the leaf is a folded definition, let the substitutions accumulate at the leaf (rules 3 and 11). Explicit substitutions readily distributes over application (rule 4) and abstraction (rule 7). Multiple substitutions can also be accumulated into one (rule 8). Applications of abstractions directly convert into explicit substitutions, provided that there no name clashes (rule 5). Otherwise, we have to rename the bound variable accordingly (rule 6).

insert instantiation rules 12, 13. Problematic as long as "inside enabled" is not well defined.

We will neeed the notion of free variables which needs a little elaboration in the presence of explicit substitutions and instantiations.

Definition 1 (Free Variables)**.** *We define the free variables $FV(t)$ inductively:*

- $FV(x) = \{x\}$ *if x is a temporal variable, temporal constant or formal parameter*
- $FV(s(t)) = FV(s) \cup FV(t)$
- $FV(\lambda x : s) = FV(s) \backslash \{x\}$

[5]For instance, explicit substitution nodes are vital to the termination proof of the permutation algorithm.

- $FV(s\sigma) = \{FV(x\sigma) | x \in FV(s)\}$
- $FV(s[M : \rho]) = \{FV(\rho_M(x)) | x \in FV(s)\}$

A substitution σ is a function which maps formal parameters to expression and which differs from the identity function id only at a finite number of points. We write $\sigma = \{x \leftarrow e\}$ for the function

$$\sigma(y) = \begin{cases} e & \text{if } y = x \\ y & otherwise \end{cases}$$

. The composition $f \circ g$ of two substitutions is simply function composition $f(g(x))$. The removal operator $\sigma \backslash S$ is defined as:

$$\sigma(y) \backslash S = \begin{cases} y & \text{if } y \in S \\ \sigma(y) & otherwise \end{cases}$$

For a substitution σ, we define the domain $dom(\sigma)$ as the (finite) set of formal parameters which have non-trivial assignments and the range $rg(\sigma)$ as the (finite) set of formal parameters occurring free in the image of $dom(\sigma)$.

Describe instantiations.

We assume we have a set $Unfolded$ of definitions which are unfolded and a context tracing if we are inside ENABLED. When we denote meta-variables standing for any term in boldface, arbitrary FP substins as σ, and an assignment of $CS(M) \cup VS(M)$ to arbirary terms as ρ_M, then the rewrite rules are:

	rewrite rule			side conditions	
1	$c\sigma$	$\rightarrow$	c	$c \in CS \cup VS$	
2	$x\sigma$	$\rightarrow$	$\sigma(x)$	$x \in FP$	
3	D	$\rightarrow$	b	$D \in DS, b = D.body, D \in Unfolded$	
4	$(\mathbf{f}(\mathbf{g})))\sigma$	$\rightarrow$	$(\mathbf{f}\sigma)(\mathbf{g}\sigma)$		
5	$(\lambda x : \mathbf{s})(\mathbf{t})$	$\rightarrow$	$s(\{x \leftarrow t\})$	$x \notin FV(\mathbf{t})$	
6	$(\lambda x : \mathbf{s})(\mathbf{t})$	$\rightarrow$	$(\lambda z : \mathbf{s}(\{x \leftarrow z\}))(\mathbf{t})$	$x \in FV(\mathbf{t}), z \notin FV(\mathbf{s}) \cup FV(\mathbf{t})$	
7	$(\lambda x : \mathbf{s})\sigma$	$\rightarrow$	$\lambda x : \mathbf{s}(\sigma \backslash \{x\})$		
8	$\mathbf{t}\sigma_1\sigma_2$	$\rightarrow$	$\mathbf{t}(\sigma_1 \circ \sigma_2)$		
9	$[M : \rho_M]!c$	$\rightarrow$	$\rho_M(c)$	$c \in CS \cup VS$	
10	$[M : \rho_M]!x$	$\rightarrow$	x	$x \in FP$	
11	$[M : \rho_M]!D$	$\rightarrow$	$[M : \rho_M]!(b)$	$D \in DS, D \in Unfolded, b = D.body$	
12	$[M : \rho_M]!(\mathbf{f}(\mathbf{g}))$	$\rightarrow$	$[M : \rho_M]!(\mathbf{f})($ $[M : \rho_M]!(\mathbf{g}))$	$\mathbf{f} \neq'$ or $\mathbf{f}$ outside of EN	
13	$[M : \rho_M]!(\mathbf{g}')$	$\rightarrow$	$([M : c \leftarrow \$c]!(\mathbf{g}))'$	$'$ inside of $EN, \$c \notin CS \cup VS$	*

Remarks:

* this is unclean since it doesn't capture the difference between $EN(x \neq x' \wedge x = x')$ and $EN(x \neq x') \wedge EN(x = x')$ well

FP-substitution stops at folded definitions and CS/VS substitutions
CS/VS-substitution stops at folded definitions and FP-substitutions

2.2 Confluence

We consider all critical pairs:

- Rule 4 and rule 5 at position (1):
  For this to happen we can assume that $x \notin FV(t)$. Still we need to make a case distinction between $x \in t\sigma$ and $x \notin t\sigma$

- $x \notin t\sigma$: Starting rewriting at ϵ, we derive:

$$\begin{aligned}((\lambda x : s)t)\sigma &\to ((\lambda x : s)\sigma)(t\sigma)\\ &\to (\lambda x : s(\sigma\backslash\{x\}))(t\sigma)\\ &\to (s(\sigma\backslash\{x\}))\{x \leftarrow t\sigma\} \quad \text{if } x \notin FV(t\sigma)\\ &\to s(\sigma\backslash\{x\} \circ \{x \leftarrow t\sigma\})\end{aligned}$$

Starting rewriting at (1), we derive:

$$\begin{aligned}((\lambda x : s)t)\sigma &\to (s\{x \leftarrow t\})\sigma)\\ &\to s(\{x \leftarrow t\} \circ \sigma)\end{aligned}$$

But $(\sigma\backslash\{x\} \circ \{x \leftarrow t\sigma\}) = (\{x \leftarrow t\} \circ \sigma)$:

$$(\sigma\backslash\{x\} \circ \{x \leftarrow t\sigma\})(y) = \begin{cases} t\sigma & \text{if } y = x\\ y\sigma & otherwise\end{cases}$$

$$(\{x \leftarrow t\} \circ \sigma)(y) = \begin{cases} t\sigma & \text{if } y = x\\ y\sigma & otherwise\end{cases}$$

- $x \in FV(t\sigma)$: Starting rewriting at ϵ, we derive:

$$\begin{aligned}((\lambda x : s)t)\sigma &\to ((\lambda x : s)\sigma)(t\sigma)\\ &\to (\lambda x : s(\sigma\backslash\{x\}))(t\sigma)\\ &\to (\lambda z : (s(\sigma\backslash\{x\})\{x \leftarrow z\})(t\sigma) \quad \text{if } x \in FV(t\sigma), z \notin FV(s(\sigma\backslash\{x\})) \cup FV(t\sigma)\\ &\to (s(\sigma\backslash\{x\})\{x \leftarrow z\})\{z \leftarrow t\sigma\}\\ &\to (s(\sigma\backslash\{x\} \circ \{x \leftarrow z\}))\{z \leftarrow t\sigma\}\\ &\to s((\sigma\backslash\{x\} \circ \{x \leftarrow z\}) \circ \{z \leftarrow t\sigma\})\end{aligned}$$

handle $x \in t\sigma$ case

- Rule 4 and rule 6 at position (1): Starting rewriting at ϵ we derive:

$$\begin{aligned}((\lambda x : s)t)\sigma &\to ((\lambda x : s)\sigma)(t\sigma)\\ &\to (\lambda x : s(\sigma\backslash\{x\}))(t\sigma)\end{aligned}$$

Starting rewriting at ϵ we derive:

$$((\lambda x : s)t)\sigma \to ((\lambda z : s\{x \leftarrow z\})t)\sigma$$

finish

2.3 Termination

We define a lexicographic order on the following measures:

1. Variable name clashes:
   - $clash(c) = clash(v) = clash(fp) = 0$
   - $clash(s(t)) = clash(s) + clash(t)$ where s is not abstraction
   - $clash(\lambda x.s)t = clash(s)$ if $x \notin FV(t)$
   - $clash(\lambda x.s)t = clash(s) + 1$ if $x \in FV(t)$
   - $clash(\lambda x.s) = clash(s)$ for the cases not covered above
2. Deepest term under an abstraction which can be applied:
   - $d(v) = d(c) = d(fp) = 0$
   - $d(\lambda x.s) = 1 + d(s)$
   - $d(s(t)) = 1 + max(d(s), d(t))$

   - $da(v) = da(c) = da(fp) = 0$
   - $da(\lambda x.s) = 1 + da(s)$
   - $da(s(t)) = \begin{cases} 1 + d(s) + max(da(s), da(t)) & \text{if } s = \lambda x.r\\ max(da(s), da(t)) & \text{otherwise}\end{cases}$

3 Open Problems

- Soundness of the Rules wrt to Denotational Semantics

4 Excursion: parametrized instantiations in SANY

In SANY, parametrized instantuitions have a representation where the parameter is shifted over the instantiation. Let us consider the modules Foo and Bar:

```
---- MODULE Foo ----

VARIABLE a

D(u) == u' # a'
E(u) == D(u) \/ ENABLED D(u)

====
```

```
---- MODULE Bar ----
EXTENDS Naturals

VARIABLE x

I(v) == INSTANCE Foo WITH a <- x+v

====
```

The term I(x)!E(x) is represented as I!E(x,x) but they are not equivalent. In the meta-notation, they would be E{u $\mapsto$ x}[a $\mapsto$ x+v]{v $\mapsto$ x} vs. E{u $\mapsto$ x, v $\mapsto$ x}[a $\mapsto$ x+v] with the same consequences as in the introduction.

compute the set of unfolded defs

$(ENABLED\ A) \Leftrightarrow C$ used instantiated

A Dropped Approach: Idempotence of Explicit Substitutions

An important property of the rewriting system is that only introduces explicit idempotent substitutions. Intuitively speaking, the duplication of a substitution σ that is distributed over an application may lead to multiple successive applications of σ. In this case, the idempotence of σ ensures that the number of applications has no influence on the result.

Lemma 1 (Idempotence of substitutions is invariant under rewriting)**.** *Let* $\mathbf{s}$ *be an expression where all explicit substitutions are idempotent and let* $\mathbf{t}$ *an expression* $\mathbf{s}$ *such that* $\mathbf{s}$ *rewrites to* $\mathbf{t}$*. Then all explicit substitutions in* $\mathbf{t}$ *are also idempotent.*

Proof. We proceed by induction on the structure of $\mathbf{s}$. If it is a leaf node (a temporal constant, temporal variable or a formal parameter), there is no explicit substitution present and the property holds trivially. Now assume as IH that the property holds for the terms $\mathbf{f}, \mathbf{g}$. Furthermore let us assume that $\mathbf{f}, \mathbf{g}$ only contain idempotent substitutions. Then by IH all substitutions in any reducts $\mathbf{rf}$ and $\mathbf{rg}$ of $\mathbf{f}$ and $\mathbf{g}$ are also idempotent. If a new node containing $\mathbf{f}$ and $\mathbf{g}$ does only have redexes inside of $\mathbf{f}$ and $\mathbf{g}$, the property trivially carries on to all redexes. Therefore we still need to check possible new redexes, where each corresponds to one of the patterns in our rewrite system which changes or introduces new substitutions:

- Rule 5: $\{x \leftarrow t\} \circ \{x \leftarrow t\} = \{x \leftarrow t\}$ because of the rule's condition that $x \notin t$.
- Rule 6: $\{x \leftarrow z\} \circ \{x \leftarrow z\} = \{x \leftarrow z\}$ because of the rule's condition that $x \in FV(t)$ but $z \notin FV(t)$.
- Rule 7: by assumption we know that σ is idempotent. But since $x(\sigma \backslash \{x\}) = x$ and $y(\sigma \backslash \{x\}) = y\sigma$ for $y \neq x$, the substitution $\sigma \backslash \{x\}$ is also idempotent.
- Rule 8: the property holds because the concatenation of idempotent functions is again idempotent. This is wrong!

□

Idempotence also ensures that domain and range of substitutions differ:

Lemma 2. *Let* σ *be an idempotent substitution. Then* $dom(\sigma) \cap rg(\sigma) = \emptyset$*.*

Proof. For the sake of contradiction suppose that there exists $x \in dom(\sigma) \cap rg(\sigma)$. Unfolding the definitions of domain and range, we know that $x\sigma = s$ form some term $s \neq x$ and that there exists a $z \in dom(\sigma)$ and a term $t[x]$ such that $\mathbf{z}\sigma = t[x]$[6].

We distinguish the following shapes of $z\sigma$:

- Constant, Variable: This case is not applicable because $t[x]$ must contain the formal parameter x.
- Formal Parameter: Assume $\mathbf{z}\sigma = x$. Because $x\sigma \neq x$, $\mathbf{z}$ must be a different parameter than x. But then $\mathbf{z}\sigma = t[x]$ is different from $\mathbf{z}(\sigma \circ \sigma) = t[s]$ which contradicts the idempotence of σ.
- Abstraction : Assume $\mathbf{z}\sigma = \lambda u.r[x]$ (with $u \neq x$). But $\lambda u.r[x]$ is different from $\lambda u \lambda u.t[x]$ which contradicts the idempotence of σ.
- Application: Assume $\mathbf{z}\sigma = f(r[x])$ for some f. Again, $f(t[x])$ is different from $f(t[f(t[x])])$ which contradicts the idempotence of σ.
- FP Substitution: Assume $\mathbf{z}\sigma = r\tau$ for some term r. By the definition of the set of free variables, $FV(r\tau) = \{FV(u\tau) | u \in FV(r)\}$, so either $r\tau$

  But $\tau = \tau \circ \tau$ so this one is idempotent.

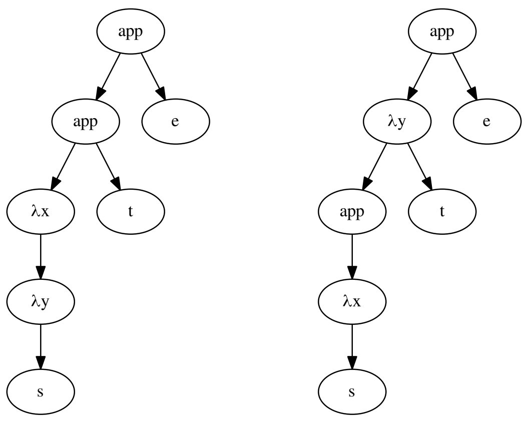
- Instantiation:

  Finish

[6] Although $t[x]$ may be x itself.

B Dropped approach: Problems with current Orderings

The ordering *da* does not decrease in some cases (e.g.: wrap the redex of the *** rule into an application redex(e)). The reason is that outer pattern matches weigh heavier than inner ones:



The weight of the abstraction on x decreases as expected, but at the same time the weight of the abstraction on y increases. Since y is higher, its weight contributes more.

$\square$