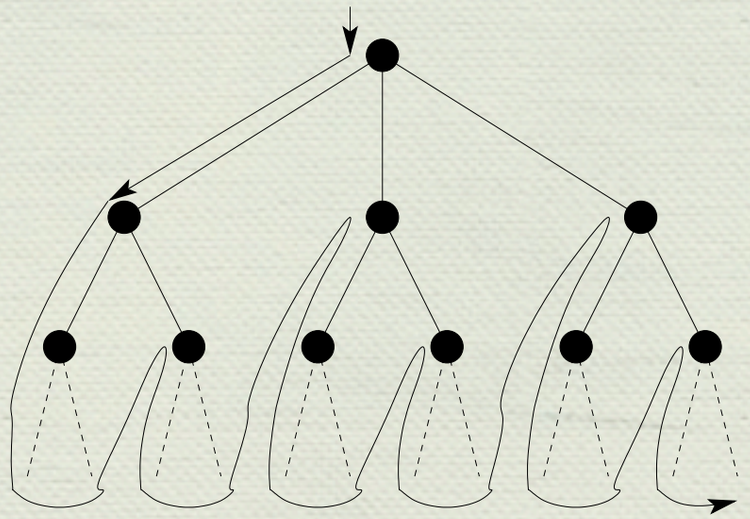
An Isabelle/HOL-based Model of Stratego-like Traversal Strategies

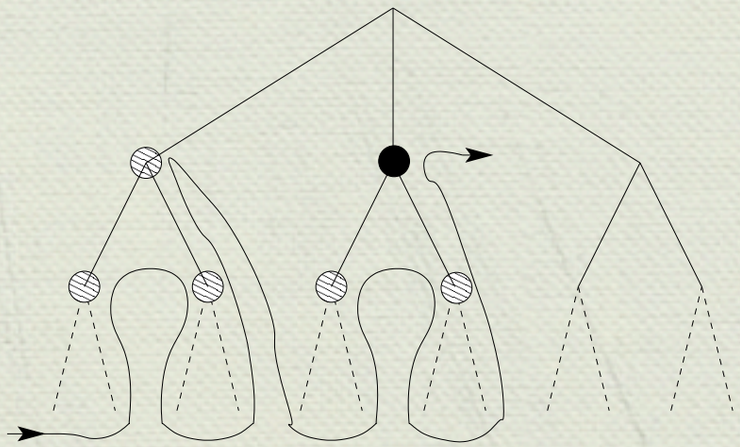
Markus Kaiser and Ralf Lämmel
Software Languages Team
Universität Koblenz-Landau

Term (tree) traversal



complete top-down traversal with transformation at all nodes

This work: model strategies formally and determine properties!



incomplete bottom-up traversal with transformation at one node

Stratego-like strategies

Strategy primitives

$$
\begin{array}{lcl}
s & ::= & t \rightarrow t \\
 & | & \epsilon \\
 & | & \delta \\
 & | & s;s \\
 & | & s \leftarrow\!\!+ s \\
 & | & \Box(s) \\
 & | & \Diamond(s)
\end{array}
$$

rewrite rules

id & fail

sequ & choice

all & one

...

+ recursion

+ type-case (possibly)

Strafunski's so-called **adhoc** combinator

Traversal schemes

$$
\begin{array}{lcl}
topdown(s) & = & s;\Box(topdown(s)) \\
bottomup(s) & = & \Box(bottomup(s));s \\
oncetd(s) & = & s \leftarrow\!\!+ \Diamond(oncetd(s)) \\
oncebu(s) & = & \Diamond(oncebu(s)) \leftarrow\!\!+ s \\
stoptd(s) & = & s \leftarrow\!\!+ \Box(stoptd(s)) \\
stopbu(s) & = & \Box(stopbu(s)) \leftarrow\!\!+ s \\
innermost(s) & = & repeat(oncebu(s))
\end{array}
$$

Some variation points

- Transformation vs. query.
- Single vs. cascaded traversal.
- Top-down vs. bottom-up traversal.
- Depth-first vs. breadth-first traversal.
- Left-to-right traversal and vice versa.
- Full vs. single-hit vs. cut-off traversal.
- Types vs. general predicates as milestones.
- Fixpoint by equality test vs. fixpoint by failure.
- Local choice vs. full backtracking vs. explicit cut.
- Traversal with effects (accumulation, cloning, etc.).

Traversal gone bad

- What could go wrong?
  - A traversal ***diverges***.
  - A traversal ***fails (nearly) always*** (say, too often).
  - A traversal ***succeeds w/o transformation*** ((nearly) always).
  - A traversal ***does not traverse deeply***.

Let's traverse trees over naturals.

Switching to Haskell!

- **Types**
  - **data** Nat = Zero | Succ Nat
  - **data** Tree a = Node {rootLabel :: a, subForest :: [Tree a]}
- **Sample trees**
  - tree1 = Node { rootLabel = Zero, subForest = [] }
  - tree2 = Node { rootLabel = True, subForest = [] }
  - tree3 = Node { rootLabel = Succ Zero, subForest = [tree1,tree1] }

Traversal gone bad

- Increment naturals in the tree
  - Rewrite rule
    - `increment n = Just (Succ n)`
  - Strategy
    - `topdown (adhoc id increment) tree1`

expands before descent and hence diverges

Apply increment on naturals and behave like the identity function for all other types.

Traversal gone bad

- Increment naturals in the tree
  - Rewrite rule
    - `increment n = Just (Succ n)`
  - Strategy
    - `bottomup (adhoc id increment) tree3`

increments bottom-up and hence doubles

Traversal gone bad

- Increment naturals in the tree
  - Rewrite rule
    - `increment n = Just (Succ n)`
  - Strategy
    - `stoptd (adhoc id increment) tree1`

increment vacously succeeds for tree nodes (too early)

Use fail hence!

Termination behavior

- **`topdown s`** may diverge even for terminating **`s`**.
  - It's terminating if **`s`** does not increase term size.
- **`bottomup s`** is terminating as long as **`s`** is terminating.
- **`stoptd s`** is terminating as long as **`s`** is terminating.
- **`innermost s`** may diverge even for terminating **`s`**.
  - It's terminating if **`oncebu s`** "decreases" some measure.

Success / failure behavior

"make sense" properties still to be formalized!

- **`topdown s`** may fail if **`s`** may fail.
  - **`s`** should fail only exceptionally "to make sense".
- **`stoptd s`** cannot possibly fail (no matter what **`s`**).
  - **`s`** should succeed rarely "to make sense".
- **`oncebu s t`** succeeds for **`t`** for if **`s`** succeeds for a subterm of **`t`**.
  - **`s`** should succeed rarely "to make sense".

An Isabelle/HOL-based Model of Stratego-like Traversal Strategies

See the paper for details.

- Input (inspiration)
  - Function combinators for strategic programming
    - Paper and pencil SOS of strategic programming
- Show correspondence of both definitions
- Formalize and prove laws & properties on top
- **Use Isabelle/HOL for mechanized model**

Functional model in Isabelle/HOL

- Term constructors
  - `types con = nat;`
- Terms
  - `datatype cterm = C con "cterm list";`
- Strategies (functions on terms)
  - `types strategy = "cterm => result";`
  - `types result = "cterm option";`

Strikingly similar to "Strafunski"; types could be added as well.

Function combinator *all*

```
all_def: all s t  =
    case (postMapAll (map s (children t))) of
          None  → None
        | Some l  → Some (C (con_of t) l)
```

```
postMapAll [] = Some []
postMapAll (r#rs) =
     case r of
           None  → None
         | Some x  → (case (postMapAll rs) of
                               None  → None
                             | Some xs  → Some (x#xs))
```

postMapAll :: ('a option) list → ('a list) option

Paper & pencil SOS

$$\frac{\forall i \in \{1, \ldots, n\}.\ s \mathbin{@} t_i \rightsquigarrow t'_i}{\Box(s) \mathbin{@} c(t_1, \ldots, t_n) \rightsquigarrow c(t'_1, \ldots, t'_n)} \qquad [\mathsf{all}^+]$$

$$\frac{\exists i \in \{1, \ldots, n\}.\ s \mathbin{@} t_i \rightsquigarrow \uparrow}{\Box(s) \mathbin{@} c(t_1, \ldots, t_n) \rightsquigarrow \uparrow} \qquad [\mathsf{all}^-]$$

Recover SOS as lemmas

$$\frac{\forall i \in \{1,\ldots,n\}.\ s \mathbin{@} t_i \rightsquigarrow t'_i}{\Box(s) \mathbin{@} c(t_1,\ldots,t_n) \rightsquigarrow c(t'_1,\ldots,t'_n)} \quad [\mathsf{all}^+]$$

lemma *all_pos_sos:*

(∀ (i::nat). 1 ≤ i ∧ i ≤ n ⟹ s (ts i) = Some (ts' i))

⟹ (all s (C c (vector n ts)) = Some (C c (vector n ts')))

Laws and properties

1. Laws

2. Termination behavior

3. Success/failure behavior

aka "modeling recursive (partial) strategies"

aka "in-/fallibility"

Laws and properties

1. **Laws**

2. Termination behavior

3. Success/failure behavior

Some laws of strategy primitives

lemma *sequ_assoc_law:*
$$sequ\ s\ (sequ\ s'\ s'') = sequ\ (sequ\ s\ s')\ s''$$
lemma *choice_assoc_law:*
$$choice\ s\ (choice\ s'\ s'') = choice\ (choice\ s\ s')\ s''$$

---

lemma *distr_left_law:*
$$sequ\ s\ (choice\ s'\ s'') = choice\ (sequ\ s\ s')\ (sequ\ s\ s'')$$
NOT A ***lemma*** *distr_right_law:*
$$sequ\ (choice\ s\ s')\ s'' = choice\ (sequ\ s\ s'')\ (sequ\ s'\ s'')$$

---

lemma *all_id_law :* $all\ id = id$
lemma *one_fail_law :* $one\ fail = fail$

---

lemma *all_constant_law :* $constant\ t \Longrightarrow all\ s\ t = id\ t$
lemma *all_not_constant_law :* $\neg(constant\ s) \Longrightarrow all\ fail\ t = fail\ t$
lemma *one_constant_law:* $constant\ t \Longrightarrow one\ s\ t = fail\ t$
lemma *one_not_constant_law :* $\neg(constant\ t) \Longrightarrow one\ id\ t = id\ t$

Fusion law of all

lemma *all_fusion_law: sequ (all s) (all s') = all (sequ s s')*

---

axioms
map'_rule: map' (sequ s s') xs = bind (map' s xs) (map' s');
consts
bind :: 'a option → ('a → 'z option) → 'z option
map' :: ('a → 'a option) → 'a list → 'a list option
defs
map'_def: map' s xs = postMapAll (map s xs)
primrec
bind None s = None
bind (Some x) s = s x

Follows from fusion law for monadic list map say for the Maybe monad.

A. Pardo. Fusion of recursive programs with computational effects. *Theoretical Computer Science*, 260(1–2):165–207, 2001.

Laws and properties

1. Laws
2. **Termination behavior**
3. Success/failure behavior

Non-models of recursive strategies (Example: bottom-up traversal scheme)

- Operational intuition
  - **`bottomup s == sequ (all (bottomup s)) s`**
- *Recursive definition* not admitted.
- *Axiom* may lead to inconsistent logic.

This is the definition used in Haskell (Strafunski) and Stratego!

Modeling recursive strategies I / III

- Start from recursive "definition"
  - **`bottomup s == sequ (all (bottomup s)) s`**
- Untie recursive knot
  - **`bottomup_step s c rs =`**

```
case (postMapAll rs) of
    None => None
    Some ts' => s (C c ts')
```

Obviously,
this is a proper definition.
It was obtained by
unfolding definitions and
parametrization et al.

Modeling recursive strategies II/III

Derive recursive definition as ***inductive set***.

```
consts
  bottomup_step :: strategy  → con  → result list  → result
  bottomup_set :: strategy  → (cterm  × result) set

defs
  bottomup_step s c rs  =
    case (postMapAll rs) of
        None  → None
      | Some ts'  → s (C c ts')

inductive bottomup_set s
  intros
    rule[intro!]:
      (∀ (t::cterm). in_list t ts  ⟹
        (∃ (r::result).
              in_list (t,r) (zip ts rs)
            ∧ length ts = length rs
            ∧ (t,r):(bottomup_set s)))
          ⟹ (C c ts,bottomup_step s c rs):(bottomup_set s)
```

As shown before

- form term/result pairs for all kids
- retrieve recursive results from set
- add another step of traversal to set

Modeling recursive strategies III/III

1. Untie recursive knot (*done*)
2. Derive inductive set (*done*)
3. Convert set to function (*omitted*)
4. Prove correctness of function

Slogan:
"So what didn't work as an axiom or as a definition does still hold as a lemma."

lemma *bottomup_rec:*
bottomup_fun s t = sequ (all (bottomup_fun s)) s t;

Laws and properties

1. Laws

2. Termination behavior

3. **Success/failure behavior**

In-/fallibility of the strategy primitives

lemma *id_not_fail:*
infallible id
lemma *sequ_not_fail:*
infallible s $\wedge$ *infallible s'* $\Longrightarrow$ *infallible (sequ s s')*
lemma *choice_not_fail:*
infallible s $\vee$ *infallible s'* $\Longrightarrow$ *infallible (choice s s')*
lemma *all_not_fail:*
infallible s $\Longrightarrow$ *infallible (all s)*

---

lemma *fail_fail :* *fallible fail*
lemma *sequ_fail :* *fallible s* $\Longrightarrow$ *fallible (sequ s s')*
lemma *all_fail :* *fallible s* $\Longrightarrow$ *fallible (all s)*
lemma *one_fail :* *fallible (one s)*

NOT A ***lemma*** *choice_fail:*
fallible s $\wedge$ *fallible s'* $\Longrightarrow$ *fallible (choice s s')*

Infallibility of the bottom-up scheme

lemma *bottomup_not_fail:*
infallible s $\Longrightarrow$ *infallible (bottomup_fun s)*

Prove by induction on size of input term

Lemma for induction step

lemma *bottomup_not_fail_step:*
infallible s
$\wedge$ *(*$\forall$ *(t'::cterm). size t' <* $<$ *size t* $\Longrightarrow$ *bottomup_fun s t'* $\neq$ *None)*
$\Longrightarrow$ *bottomup_fun s t* $\neq$ *None*

A note on complexity

Theory	**LOC**	**KB**	**All**	**Main**	**Other**
Terms (§3)	82	3	16	0	16
Primitives (§3)	146	5	25	3	22
SOS (§4)	56	2	12	12	0
Laws (§5)	895	33	161	29	132
Model of (§6, §7)					
• *repeat*	576	25	105	2	103
• *bottomup*	163	10	23	2	21
• *topdown*	247	15	34	2	32
• *oncebu*	148	8	21	2	19
• *innermost*	23	1	3	2	1
(In)fallbility of (§8)					
• *bottomup*	36	2	5	1	4
• *topdown*	126	6	19	4	15
• *stoptd*	46	2	7	1	6
• *innermost*	7	1	1	1	1

Last slide

- **Future work**
  1. Formalization of "make sense" properties
  2. More general treatment of recursion
  3. More ***auto***mated proofs
  4. Twelf? Coq?
  5. Improve "usability" of traversal strategies
  6. Develop correct optimizations for schemes
  7. Incorporation of term-rewriting theory
  8. Model of typed strategies

Acknowledgment: this work has also benefited from collaboration with Simon Thompson (see our LDTA 2008 paper in particular).