Verification of the Deutsch-Schorr-Waite marking algorithm on Agda1

Yoshinori Tanabe

University of Tokyo, CVS AIST

joint work with

Yoshifumi Yuasa (Tokyo Institute of Technology, CVS AIST)
Toshifusa Sekizawa (CVS AIST)
Koichi Takahashi (CVS AIST)
Background

- Interested in: application of modal fixedpoint logics to analyze graph rewriting systems.
- A target: programs that manipulate pointers.
- This talk: a case study – correctness proof of Deutsch-Schorr-Waite marking algorithm (DSW)
Deutsch-Schorr-Waite marking algorithm (DSW)

What is DSW?

- Marks all nodes that are reachable from the root in the manner of depth first search.

```
root
  l r
  m l r
  m r
  m
```

- Does not use a stack to hold the nodes for backtracking. Instead, rewrites the pointers to remember the parent node.

```
root
  l r
  m l r
  m r
  m
```
Why do we want to verify DSW?

- Because it is regarded as a benchmark of program analysis methods.

_The Schorr-Waite algorithm is the first mountain that any formalism for pointer aliasing should climb._

— Richard Bornat
Related work

Yang (2000)  Formal proof on Bunched Implications (~ Separation Logic )
Mehta, Nipkow (2003)  Formal proof, with Isabelle/HOL.
Hubert and Marche (2005)  Formal proof, with coq, generating proof obligations from a C source code, using CADUCEUS.

Ours:

- Constructs proofs. System built on top of Agda1.
- An automatic validity checker, used to reason about atomic statements, enables concise proofs. DSW partial correctness proof is about 320 lines.
Related work

Yang (2000)  Formal proof on Bunched Implications (∼ Separation Logic )
Mehta, Nipkow (2003)  Formal proof, with Isabelle/HOL.
Hubert and Marche (2005)  Formal proof, with coq, generating proof obligations from a C source code , using CADUCEUS.

Ours:

- Constructs proofs. System built on top of Agda1.
- An automatic validity checker, used to reason about atomic statements, enables concise proofs. DSW partial correctness proof is about 320 lines.
1. Introduction

2. Logic that describes Heaps

3. Hoare logic on Agda

4. Partial correctness of DSW

5. Termination of DSW

6. Conclusion
Introduction

2. Logic that describes Heaps

3. Hoare logic on Agda

4. Partial correctness of DSW

5. Termination of DSW

6. Conclusion
Programming Language PML

- All variables are of pointer type.
- A pointer points to an object. An object has boolean fields and pointer fields.

**Statements**
- Atomic statements
  - \( x := \text{NULL} \)
  - \( x := y \) \((x, y: \text{variables})\)
  - \( x := y.f \) \((f: \text{pointer-type field})\)
  - \( x.b := \text{True} \), \( x.b := \text{False} \) \((b: \text{boolean-type field})\)
  - \( x.f := y \)
- if-then-else statements
- while statements

**Conditions**
- \( x == y \)
- \( x.b == \text{True}, \ x.b == \text{False} \)
t := root; p := NULL;
while (!(p == NULL) || !(t == NULL || t.m == True)) do {
  if (!(t == NULL || t.m)) then {
    x := p; p := t; t := t.l;
    p.l := x; p.m := True; p.s := False;
  } else if (p.s == False) then {
    x := t; t := p.r; y := p.l; p.r := y;
    p.l := x; p.s := True;
  } else {
    x := t; t := p; p := p.r; t.r := x;
  }
}
Alternation Free modal $\mu$-calculus with Nominals

- $PS \ni p$ (Propositional Symbol)
- $Nom \ni n$ (Nominal)
- $MS \ni m$ (Modality Symbol)
- $FORM \ni \varphi$ (Formula)

\[
\ ::=\ p\ |\ n\ |\ X\ |\ \neg\varphi\ |\ \varphi\ \lor\ \varphi\ |\ \langle m \rangle \varphi\ |\ \langle m^{-1} \rangle \varphi\ |\ \mu X.\ \varphi\ |\ @$n.\ \varphi
\]

A nominal is an atomic formula. It is satisfied only one state of a Kripke structure.

We restrict it to the alternation free fragment
Heaps as Kripke structures

Kripke str / AF$\mu$N

State
Nominal
Propositional Symbol
Modality Symbol

Heap / PML

Object
Variable
Boolean Field
Pointer Field

Nom = \{root, t, p\}
PS = \{m, s\}
Mod = \{l, r\}
AF$\mu$N formulas describe heap properties

- $@\text{root} (p \land m)$
  - $p$ and root point to the same object, and the value of field $m$ is true at the object. The value of boolean fields $p$ and $m$ of root is true.

- $@t \langle l \rangle s$
  - The value of field $s$ of $l$-successor of $t$ is true.

- $@p \mu X (t \lor (m \land \langle r \rangle X))$
  - From object $p$, object $t$ is reachable by following $r$-pointers. And all $m$ fields between $p$ and $t$ is true.
Two key facts on AFμN

- An effective decision procedure for satisfiability exists. We have built a validity checker using the procedure.
- The strongest postconditions $sp(\sigma, \varphi)$ of formula $\varphi$ with respect to PML atomic statements $\sigma$ can effectively been calculated.

E.g. Let $\sigma = t.m := \text{false}$, $\varphi = @p \nu X (m \land [r]X)$. Then,

$$sp(\sigma, \varphi) = @p \nu X((m \lor t)) \land [r]X$$
Hoare triple

For PML program $\sigma$ and AF$\mu$N formulas $\varphi$ and $\psi$:

$$\{\varphi\} \sigma \{\psi\} \overset{\text{def}}{\iff} K \models \varphi \implies \sigma(K) \models \psi$$

If $\sigma$ is an atomic statement, $\{\varphi\} \sigma \{\psi\}$ can be judged using the following fact:

If

$$\text{sp}(\sigma, \varphi) \rightarrow \psi$$

is valid, then

$$\{\varphi\} \sigma \{\psi\}$$

holds.
Hoare triple

For PML program $\sigma$ and AF$\mu$N formulas $\varphi$ and $\psi$:

$$\{ \varphi \} \sigma \{ \psi \} \overset{\text{def}}{\iff} K \models \varphi \implies \sigma(K) \models \psi$$

If $\sigma$ is an atomic statement, $\{ \varphi \} \sigma \{ \psi \}$ can be judged using the following fact:

- If

  $$\text{sp}(\sigma, \varphi) \rightarrow \psi$$

  is valid, then

  $$\{ \varphi \} \sigma \{ \psi \}$$

  holds.
Statements in PML

- **Atomic statements**
  - \( x := \text{NULL} \)
  - \( x := y \) (\( x, y \): variables)
  - \( x := y.f \) (\( f \): pointer-type field)
  - \( x.b := \text{true}, \ x.b := \text{false} \) (\( b \): boolean-type field)
  - \( x.f := y \)

- **if-then-else statements**

- **while statements**
Hoare logic inference rules

\[
\begin{align*}
\frac{\text{spec}(A, c ; \cdot) \vdash B}{A \{c\} B} \quad \text{atmHc} \\
\frac{A \land \bar{s} \{cs_0\} B \quad A \land \neg \bar{s} \{cs_1\} B}{A \{\text{if } s \text{ then } \{cs_0\} \text{ else } \{cs_1\}\} B} \quad \text{iteHc} \\
\frac{A \vdash C \quad C \land \bar{s} \{cs\} C \quad C \land \neg \bar{s} \vdash B}{A \{\text{while } s \text{ do } \{cs\}\} B} \quad \text{whlHc} \\
\frac{A \{c\} C \quad C \{cs\} B}{A \{c; cs\} B} \quad \text{cnsHc}
\end{align*}
\]
1. Introduction
2. Logic that describes Heaps
3. Hoare logic on Agda
4. Partial correctness of DSW
5. Termination of DSW
6. Conclusion
DSW correctness

- **Partial correctness**
  - All nodes reachable from the root will be marked.
  - All nodes not reachable from the root will not be marked.
  - All pointers will be restored.

- **Termination**
  - The program terminates.
DSW correctness

- **Partial correctness**
  - All nodes reachable from the root will be marked.
  - All nodes not reachable from the root will not be marked.
  - All pointers will be restored.

- **Termination**
  - The program terminates.
DSW correctness

- **Partial correctness**
  - All nodes reachable from the root will be marked.
  - All nodes not reachable from the root will not be marked.
  - All pointers will be restored.

- **Termination**
  - The program terminates.
Hoare triple to be proved

\{\text{pre}\} \text{dsw} \{\text{post}\}

where

- \text{pre} = \@a (\neg m \land \langle l \rangle b \land \langle r \rangle c) \land \@\text{root}\ \mu X (\neg m \land (a \lor \langle l \rangle X \lor \langle r \rangle X)),
- \text{post} = \@a (m \land \langle l \rangle b \land \langle r \rangle c).

- \text{a}, \text{b}, \text{c}: fresh nominals
- \text{a}: arbitrary object, reachable from the root at the beginning
- \text{b}: left successor of \text{a}
- \text{c}: right successor of \text{a}
The dsw program

t := root; p := NULL;
while (!(p == NULL) || !(t == NULL || t.m == True)) do {
  if (!(t == NULL || t.m)) then { /* push */
    x := p; p := t; t := t.l;
    p.l := x; p.m := True; p.s := False;
  } else if (p.s == False) then { /* swing */
    x := t; t := p.r; y := p.l; p.r := y;
    p.l := x; p.s := True;
  } else { /* pop */
    x := t; t := p; p := p.r; t.r := x;
  }
}

The dsw program

initialization;
while ( cond_push | cond_swing | cond_pop ) {
  if ( cond_push ) { push }
  else if ( cond_swing ) { swing }
  else /* cond_pop */ { pop }
}

Main issue: to find an invariant of the while loop.
The dsw program

```
initialization;
while ( cond_push | cond_swing | cond_pop ) {
    if      ( cond_push  ) { push  }
    else if ( cond_swing  ) { swing }
    else /* cond_pop */ { pop   }
}
```

Main issue: to find an invariant of the while loop.
Abstract transition system (ATS)

- $A = \neg a \land \neg \text{NULL} \land \neg p \land \beta(\text{NULL})$
- $S11 = a \land (m \land \neg s \land \langle r \rangle c) \land p \land a \land t \land b$
- $\ldots \ldots$
The ATS is correct if the following are valid.

- \( \text{sp}(\text{push}, A \land S01) \rightarrow (A \land S01) \lor (A \land S11) \)
- \( \text{sp}(\text{swing}, A \land S01) \rightarrow (A \land S01) \)
- \( \ldots \ldots \)
- \( \text{sp}(\text{pop}, A \land S22) \rightarrow (A \land S21) \lor (A \land S22) \)

They actually are valid, and can be confirmed with the validity checker. Formula “\((A \land S01) \lor \cdots \lor (A \lor S22)\)” is an invariant of the while loop.
The ATS is correct if the following are valid.

- $\text{sp}(\text{push}, A \land S01) \rightarrow (A \land S01) \lor (A \land S11)$
- $\text{sp}(\text{swing}, A \land S01) \rightarrow (A \land S01)$
- $\ldots$
- $\text{sp}(\text{pop}, A \land S22) \rightarrow (A \land S21) \lor (A \land S22)$

They actually are valid, and can be confirmed with the validity checker. Formula "$(A \land S01) \lor \cdots \lor (A \lor S22)$" is an invariant of the while loop.
The ATS is correct if the following are valid.

- $sp(push, A \land S01) \rightarrow (A \land S01) \lor (A \land S11)$
- $sp(swing, A \land S01) \rightarrow (A \land S01)$
- $\ldots$ $\ldots$
- $sp(pop, A \land S22) \rightarrow (A \land S21) \lor (A \land S22)$

They actually are valid, and can be confirmed with the validity checker.

Formula “$(A \land S01) \lor \cdots \lor (A \lor S22)$” is an invariant of the while loop.
The ATS is correct if the following are valid.

- \( \text{sp}(\text{push}, A \land S01) \rightarrow (A \land S01) \lor (A \land S11) \)
- \( \text{sp}(\text{swing}, A \land S01) \rightarrow (A \land S01) \)
- \( \ldots \)
- \( \text{sp}(\text{pop}, A \land S22) \rightarrow (A \land S21) \lor (A \land S22) \)

They actually are valid, and can be confirmed with the validity checker. Formula “(A \land S01) \lor \cdots \lor (A \lor S22)” is an invariant of the while loop.
Constructing a proof on Agda1

Construct a term for each validity checking by calling the automatic validity checker, via Agda1’s plug-in mechanism.

A proof term for a while-statement can be generated by giving these terms to a service routine.
Size of the partial correctness proof

- Agda source: 320 lines
- Running time in seconds:

<table>
<thead>
<tr>
<th></th>
<th>Properties 1,2</th>
<th>Property 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
<td>10.55</td>
<td>0.50</td>
</tr>
<tr>
<td><strong>invPush</strong></td>
<td>28.19</td>
<td>6.97</td>
</tr>
<tr>
<td></td>
<td>0.88</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>invSwing</strong></td>
<td>25.42</td>
<td>11.84</td>
</tr>
<tr>
<td></td>
<td>1.17</td>
<td>0.55</td>
</tr>
<tr>
<td><strong>invPop</strong></td>
<td>12.67</td>
<td>8.63</td>
</tr>
<tr>
<td></td>
<td>0.69</td>
<td>0.95</td>
</tr>
<tr>
<td><strong>exits loop</strong></td>
<td>2.98</td>
<td>0.63</td>
</tr>
</tbody>
</table>

(Windows XP/1.33 GHz Core 2 Duo)
1. Introduction
2. Logic that describes Heaps
3. Hoare logic on Agda
4. Partial correctness of DSW
5. Termination of DSW
6. Conclusion
Observation

Three “ranking functions”:

- $S_1(\mathcal{K}) = \{ s \in K \mid \mathcal{K}, s \models \neg m \}$
- $S_2(\mathcal{K}) = \{ s \in K \mid \mathcal{K}, s \models \neg s \}$
- $S_3(\mathcal{K}) = \{ s \in K \mid \mathcal{K}, s \models \mu X (p \lor \langle r^{-1} \rangle(s \land X) \lor \langle l^{-1} \rangle(\neg s \land X)) \}$
First, “push” cannot happen infinitely often.

Then, neither “swing” or “pop” can happen infinitely often. Therefore, the loop terminates.

Two issues:

- How can we judge “dec” or “non-inc”?
- Generally, how can we conclude that the loop terminates?
First, “push” cannot happen infinitely often.

Then, neither “swing” or “pop” can happen infinitely often. Therefore, the loop terminates.

Two issues:

- How can we judge “dec” or “non-inc”?
- Generally, how can we conclude that the loop terminates?
First, “push” cannot happen infinitely often.

Then, neither “swing” or “pop” can happen infinitely often.

Therefore, the loop terminates.

Two issues:

How can we judge “dec” or “non-inc”?

Generally, how can we conclude that the loop terminates?
First, “push” cannot happen infinitely often.

Then, neither “swing” or “pop” can happen infinitely often. Therefore, the loop terminates.

Two issues:

- How can we judge “dec” or “non-inc”?
- Generally, how can we conclude that the loop terminates?
First, “push” cannot happen infinitely often.

Then, neither “swing” or “pop” can happen infinitely often. Therefore, the loop terminates.

Two issues:

- How can we judge “dec” or “non-inc”?
- Generally, how can we conclude that the loop terminates?
Judging decrease

Ranking function $S_\varphi(K) = \{ s \in K \mid K, s \models \varphi \}$ w.r.t. $\sigma$ is

- **non-increasing** if

  $$[o](\varphi \rightarrow \text{sp}(\sigma, \varphi))$$

  is valid.

- **decreasing** if both of

  $$[o](\varphi \rightarrow \text{sp}(\sigma, \varphi))$$

  and

  $$\langle o\rangle(\neg \varphi \land \text{sp}(\sigma, \varphi))$$

  are valid.

where $o$ is the global modality, i.e. its interpretation is always $K \times K$. 
We associate “dec” and “non-inc” to each nodes of the transition system, depending on the judgement.

- “non-inc” is put to nodes not related to execution statement
- We model check the following LTL specification:

\[
\left( \bigwedge_i \neg((\lozenge \Box \text{NI}_i) \land (\Box \lozenge \text{D}_i)) \right) \rightarrow \lozenge \text{end}
\]

If this succeeds, the loop terminates.
1 Introduction
2 Logic that describes Heaps
3 Hoare logic on Agda
4 Partial correctness of DSW
5 Termination of DSW
6 Conclusion
Conclusion

Summary
- Proving properties of pointer manipulating programs using logic $AF_\mu N$.
- A system for partial correctness proofs is implemented.
  - Hoare logic on top of Agda1
  - An automatic validity checker for $AF_\mu N$ is integrated via Agda1 plug-in mechanism.
  - DSW partial correctness has successfully been proved.
- Termination is proved based on $AF_\mu N$ with LTL model checking.

Future Work
- Build a system for termination proofs on top of Agda.
- Implementation of the full decision procedure for $AF_\mu N$. 
Thank you for your attention.