The new architecture (right) of the blame-subtyping theorem CastSafeFor ℓ/m in the ABT library, which
x2p/ ({{4
Representing/1
Philip Wadler and Robert Bruce Findler. Well-typed programs can’t be blamed. In
EDI (Jeremy G. Siek and Tianyu Chen. Parameterized cast calculi and reusable meta-
{⟨ tt , tt ⟩ ℓ/m ⟦x ⟧/ λ p x M SafeFor ℓ
Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. Re-
xp EDC ABTPredicate
stands for the
m x = (m {/ m m
Jeremy G. Siek, Peter Thiemann, and Philip Wadler. Blame and coercion: Together
/m (m m m m m Term x Email:
The Gradual Typing in Agda project develops a
generic cast calculus called CC⇒) that is parameter-
etized by casts. The meta-theory of the Grad-
ually Typed Lambda Calculus (GTLC) developed with CC⇒) is reusable across multiple variants of
cast representations. Siek and Chen [1] prove
1) type safety 2) blame-subtyping, and 3) the dy-
namic gradual guarantee about CC⇒). To re-
duce repetitive work, it is necessary to abstract
away the similarities between different cast rep-
resentations and the proofs of various lemmas and
theorems. The former is done by leveraging the
module system of Agda, which makes code and proof sharing between cast representations pos-
sible (Table 1). This poster targets the latter half of
the problem: generalizing over the preservation
proofs of various predicates. Many proofs in-
volve lemmas about “substitution preserves”, such as “substitution preserves type” in type preserva-
tion, “substitution preserves term precision” in the
dynamic gradual guarantee, and “substitution pre-
serves ‘safe for’” in the blame-subtyping theorem
[2]. Specifically in this project, we focus on the last
one: proving the blame theorem [3, 4, 2] by using
the ABT library to represent CC⇒ terms and get
the substitution lemma for free (Figure 1).

![Figure: The new architecture (right) of the blame-subtyping theorem proof compared with the old one (left). The proofs of the highlighted lemmas are replaced by the instantiation of the ABT library. Edges indicate dependency.]

### Background

#### [Gradual Typing]
Gradual typing is a paradigm that combines static and dynamic typing by inserting checks on the boundaries. Consider the following partially typed program, where * stands for the statically unknown type:

\[
\begin{align*}
\text{(define (abs (n : Int)) ⇒ Int} \\
\text{(if (n < 0) (- n) n))} \\
\text{(define (dist (n : Int) (m : *)) ⇒ Int} \\
\text{(abs (n - m))}
\end{align*}
\]

Function call \text{dist(0, ‘yes’)} will result in a failed cast, which is caught at runtime. Intermedi-
ate languages where all casts are explicit are usu-
ally called cast calculi.

#### [Abstract Binding Tree]
We use the ABT library, which is an Agda implementation of Chapter 1 of Harper [5]. For example, Figure 2 represents the term \((\lambda x : *. x) \ (\text{unit} \langle \text{Unit} ⇒ * \rangle)\) in CC⇒ as an ABT:

![Figure: Representing (λx:*:x) (unitCont ⇒ *) as ABT]

#### Approach and Implementation

Instead of the usual approach of defining the “safe for” predicate as a datatype, we instantiate module ABTPredicate in the ABT library, which contains a notion of “generic predicate on ABT”. For example, consider the rule for cast:

\[
\text{safefor-cast : V (S T M) \ {c : \text{Cast} (S \Rightarrow T)} \ {t} \ ⇒ \ \text{CastBlameSafe} c \ t} \ ⇒ \ \text{M SafeFor} t
\]

It becomes a clause of \(P_s\), which is for the predicate on operator nodes:

\[
P_s \ (\text{op-cast } c) \ (t_a : []) \ (t_t , tt) \ t = \text{CastBlameSafe} c \ t \times t \times t_a
\]

Essentially, we need one such clause for each op-
erator, the right hand side of which stands for the
constraints. For cast, they say \(c = 0\) cast is safe 2) the cast term and its sub-term are safe for the same
label. The counterpart of \(P_v\) on variable nodes is \(V_s\).

The “safe for” predicate is then created as an instan-
tiation of the generic predicate, by passing the de-
definitions of both \(V_s\) and \(P_s\). Similarly, we instantiate module SubsPreserve and get the lemma “sub-
istitution preserves ‘safe for’” for free.

We state the blame-subtyping theorem as:

\[
\text{soundness-cl : V A \ \{M : \text{Term}\} \ {t} \ ⇒ \ \text{M SafeFor} t} \ ⇒ \ \tau (M \ \text{⇒ blame A \ t})
\]

The proof is by induction on the reduction sequence
\(M \ ⇒ \ \text{blame A \ t}\), which depends on the substitution lemma and “reduction preserves ‘safe for’”. The sub-
proof of the latter follows the overall structure of type
preservation.

### Theorectical Results of the Gradual Typing in Agda Project (Table 1)

We compare the proofs of various properties (rows) across cast representations (columns) before (to the left of /) and after (to the right of /) Gradual Typing in Agda. The status of a proof can be 1) : has proof assistant mechanism 2) p: has pen-and-paper proof in prior literature or 3) x: has no proof.

<table>
<thead>
<tr>
<th>Cast Representation</th>
<th>EDA</th>
<th>EDI</th>
<th>AB</th>
<th>EDC</th>
<th>LDC</th>
<th>LC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type Safety</td>
<td>n / n</td>
<td>x / x</td>
<td>n / n</td>
<td>x / x</td>
<td>p / p</td>
<td>n / n</td>
</tr>
<tr>
<td>Blame-subtyping Theorem</td>
<td>p / p</td>
<td>x / x</td>
<td>n / n</td>
<td>p / p</td>
<td>n / n</td>
<td></td>
</tr>
<tr>
<td>Dynamic Gradual Guarantee</td>
<td>x / x</td>
<td>x / x</td>
<td>n / x</td>
<td>x / x</td>
<td>x / x</td>
<td>x / x</td>
</tr>
</tbody>
</table>

### Summary of Contributions

We obtain a shorter Agda proof of the blame-
subtyping theorem of the parameterized cast cal-
culus CC⇒ by switching to the abstract
binding tree (ABT) library as the representation for terms. We define the “safe for” predicate in the style of the library and acquire the substitu-
tion lemma for free. We argue that this proof technique is suitable for proving the preser-
vation of arbitrary predicates on terms.

### Future Work

There are two possible future research directions:

- [The ABT Library] The ABT library does not pro-
vide generic predicates about reduction. A possi-
ble extension to the library may handle the reduc-
tion of ABT generally. In addition to “substitution preserves predicate”, we can then obtain generic proofs of properties about rewriting like conflu-
ence.

- [Gradual Typing in Agda] We plan to make the transition to using ABTs across the entire project. We are also investigating ways to decouple blame-
strategy, UD versus D, from our current proof of
the dynamic gradual guarantee. Additionally, we
would like to incorporate mutable references and
different heap models into the project.

### References

[1] Jeremy G. Siek and Tianyu Chen. Parameterized cast calculi and reusable meta-
theory for gradually typed lambda calculi. Journal of Functional Programming, 34:

[2] Philip Wadler and Robert Bruce Findler. Well-typed programs can’t be blamed. In
European Symposium on Programming; ESOP 2021, pages 1–16, March 2021.


again for the first time. In Conference on Programming Language Design and Imple-
mentation, PLDI, June 2015.

9781107029574.

### Contact Information

- Website: https://homes.luddy.indiana.edu/chen512
- Email: chen512@indiana.edu