

# IMPERIAL

## MEng Individual Project

Lemmagen: Extending the WebAssembly Specification Toolchain  
for Theorem Provers

Taichi Maeda  
Diego Cupello  
Rao Xiaojia  
Philippa Gardner  
Conrad Watt

June 25, 2025

# Background

# WebAssembly

WebAssembly (Wasm) is a low-level bytecode language [HRS<sup>+</sup>17].

- Runs on the Web for client and server applications.
- Compilation target for major programming languages.
- Rapidly evolving with rising industry adoption.



WEBASSEMBLY

# WebAssembly

Wasm is characterised by its strong **formalism** [Ros25].

*if blocktype instr<sub>1</sub><sup>\*</sup> else instr<sub>2</sub><sup>\*</sup> end*

- The **block type** must be **valid** as some **function type**  $[t_1^*] \rightarrow [t_2^*]$ .
- Let  $C'$  be the same **context** as  $C$ , but with the **result type**  $[t_2^*]$  prepended to the **labels** vector.
- Under context  $C'$ , the instruction sequence *instr<sub>1</sub><sup>\*</sup>* must be **valid** with type  $[t_1^*] \rightarrow [t_2^*]$ .
- Under context  $C'$ , the instruction sequence *instr<sub>2</sub><sup>\*</sup>* must be **valid** with type  $[t_1^*] \rightarrow [t_2^*]$ .
- Then the compound instruction is valid with type  $[t_1^* \text{ i32}] \rightarrow [t_2^*]$ .

$$\frac{C \vdash \text{blocktype} : [t_1^*] \rightarrow [t_2^*] \quad C, \text{labels } [t_2^*] \vdash \text{instr}_1^* : [t_1^*] \rightarrow [t_2^*] \quad C, \text{labels } [t_2^*] \vdash \text{instr}_2^* : [t_1^*] \rightarrow [t_2^*]}{C \vdash \text{if blocktype instr}_1^* \text{ else instr}_2^* \text{ end} : [t_1^* \text{ i32}] \rightarrow [t_2^*]}$$

# WebAssembly

Wasm even includes **soundness theorems** in its specification [Wor22a]:

**Theorem (Progress).** If a configuration  $S; T$  is valid (i.e.,  $\vdash S; T : [t^*]$  for some result type  $[t^*]$ ), then either it is terminal, or it can step to some configuration  $S'; T'$  (i.e.,  $S; T \hookrightarrow S'; T'$ ).

**Theorem (Preservation).** If a configuration  $S; T$  is valid with result type  $[t^*]$  (i.e.,  $\vdash S; T : [t^*]$ ), and steps to  $S'; T'$  (i.e.,  $S; T \hookrightarrow S'; T'$ ), then  $S'; T'$  is a valid configuration with the same result type (i.e.,  $\vdash S'; T' : [t^*]$ ). Furthermore,  $S'$  is an **extension** of  $S$  (i.e.,  $\vdash S \preceq S'$ ).

# WasmCert

This formalism enables a **precise mechanisation** of Wasm.

WasmCert-Rocq and WasmCert-Isabelle provide  
a complete mechanisation of Wasm in Rocq and Isabelle/HOL [WRPP<sup>+</sup>21].

```
Inductive be_typing : t_context -> seq basic_instruction -> function_type -> Prop :=  
| bet_const : forall C v, be_typing C [::BI_const v] (Tf [::] [::typeof v])  
| bet_unop : forall C t op,  
  unop_type_agree t op -> be_typing C [::BI_unop t op] (Tf [::t] [::t])  
| bet_binop : forall C t op,  
  binop_type_agree t op -> be_typing C [::BI_binop t op] (Tf [::t; t] [::t])  
...
```

# Problem

Formal rules need to be translated into multiple formats.

- Declarative representations in  $\text{\LaTeX}$  format
- Algorithmic representations in prose format
- Reference interpreter in OCaml
- Test suite in .wast format
- Mechanised definitions in Coq

Manual translation is just too tedious!

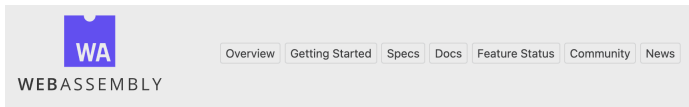
- Prone to human error.
- Not scalable as the Wasm specification continues to grow.

# SpecTec

SpecTec DSL serves as a **“single source of truth”** from which key artefacts can be auto-generated [YSL<sup>+</sup>24].

Not limited to Wasm (but some backends are specialised).

Wasm 3.0 specification will be produced with SpecTec [Ros25]!



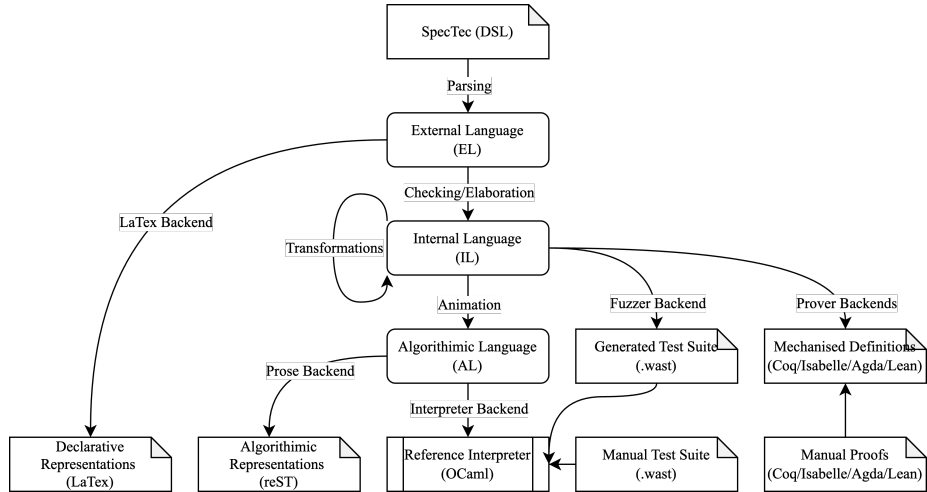
## SpecTec has been adopted

*Published on March 27, 2025 by [Andreas Rossberg](#) .*

Two weeks ago, the Wasm Community Group voted to adopt [SpecTec](#) for authoring future editions of the Wasm spec. In this post, I'll shed some light on what SpecTec is, what it helps with, and why it takes Wasm to a new level of rigor and assurance that is unprecedented when it comes to language standards.



# SpecTec



# IL2Rocq

IL2Rocq is the work of Diego, a master's student from last year [Cup24].

Major achievements by Diego:

- Implemented a **SpecTec backend** for Rocq mechanised definitions
- Completed the **preservation proof** using auto-translated Rocq definitions!

# Timeline

1. **2017~**: Wasm 1.0 draft is released [HRS<sup>+</sup>17]

# Timeline

1. **2017~**: Wasm 1.0 draft is released [HRS<sup>+</sup>17]
2. **2018~**: WasmCert is developed [Wat18, WRPP<sup>+</sup>21]
3. **2019~**: Wasm 1.0 is officially released [Wor19]

# Timeline

1. **2017~**: Wasm 1.0 draft is released [HRS<sup>+</sup>17]
2. **2018~**: WasmCert is developed [Wat18, WRPP<sup>+</sup>21]
3. **2019~**: Wasm 1.0 is officially released [Wor19]

# Timeline

1. **2017~**: Wasm 1.0 draft is released [HRS<sup>+</sup>17]
2. **2018~**: WasmCert is developed [Wat18, WRPP<sup>+</sup>21]
3. **2019~**: Wasm 1.0 is officially released [Wor19]
4. **2024~**: IL2Rocq is developed [Cup24]
5. **2024**: Preservation proof is completed [Cup24]

# Timeline

1. **2017~**: Wasm 1.0 draft is released [HRS<sup>+</sup>17]
2. **2018~**: WasmCert is developed [Wat18, WRPP<sup>+</sup>21]
3. **2019~**: Wasm 1.0 is officially released [Wor19]
4. **2024~**: IL2Rocq is developed [Cup24]
5. **2024**: Preservation proof is completed [Cup24]
6. **2025**: Progress proof is completed (**NEW**)
7. **2025**: Lemmagen is developed (**NEW**)

# Project



# Contributions

- **Progress Proof**
- **Decidable Equality Proofs**
- **First-Order Logic Extension in DSL**
- **Template Mechanism in DSL**

# Project

## Progress Proof

# Contributions

- **Progress Proof**
- Decidable Equality Proofs
- First-Order Logic Extension in DSL
- Template Mechanism in DSL

# Progress Proof

Continuation of Diego's work on IL2Coq and the preservation proof [Cup24].

- Proof by **mutual induction** on the structures of 2-3 typing relations.
- Accommodates key differences between the SpecTec DSL and WasmCert-Rocq.

```
Theorem t_progress :  
  forall s f es ts,  
    Config_ok (config__ (state__ s f) es) ts ->  
    terminal_form es /\  
    exists s' f' es',  
    Step (config__ (state__ s f) es) (config__ (state__ s' f') es').
```

# Progress Proof

Continuation of Diego's work on IL2Coq and the preservation proof [Cup24].

- Proof by **mutual induction** on the structures of 2-3 typing relations.
- Accommodates key differences between the SpecTec DSL and WasmCert-Rocq.

```
Lemma t_progress_e: forall s C C' f vcs es tf ts1 ts2 lab ret,
  Admin_instrs_ok s C es tf ->
  tf = functype__ ts1 ts2 ->
  C = (upd_local_label_return C' (map typeof f.(frame__LOCALS)) lab ret) ->
  Module_instance_ok s f.(frame__MODULE) C' ->
  map typeof vcs = ts1 ->
  Store_ok s ->
  not_lf_br es ->
  not_lf_return es ->
  terminal_form (list__val__admininstr vcs ++ es) \ /
  exists s' f' es', Step (config__ (state__ s f) (list__val__admininstr vcs ++ es))
    (config__ (state__ s' f') es').
```

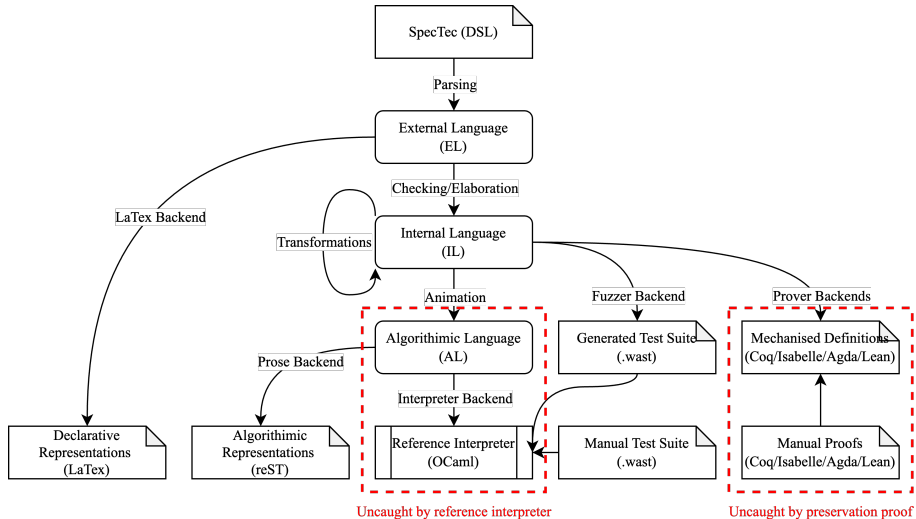
# Modifications to DSL

**Key errors in the DSL** found during the proof development  
(mostly acknowledged by Andreas and Conrad):

```
+rule Step/ctxt-seq:
+  z; val* admininstr* admininstr''* ~> z'; val* admininstr'* admininstr''*
+  -- Step: z; admininstr* ~> z'; admininstr'*

rule Step/ctxt-frame:
-  s; f; (FRAME_ n `{f'} admininstr*) ~> s'; f; (FRAME_ n `{f'} admininstr'*)
-  -- Step: s; f'; admininstr* ~> s'; f'; admininstr'*
+  s; f; (FRAME_ n `{f'} admininstr*) ~> s'; f; (FRAME_ n `{f''} admininstr'*)
+  -- Step: s; f'; admininstr* ~> s'; f''; admininstr'*
```

# Modifications to DSL



# Project

## Decidable Equality Proofs



# Contributions

- Progress Proof
- **Decidable Equality Proofs**
- First-Order Logic Extension in DSL
- Template Mechanism in DSL

# Motivation

Decidable equality is essential for case analysis in Rocq proofs.

- In **Rocq proofs**, we must show  $x = y \vee x \neq y$  explicitly (constructive logic).
- In **hand-written proofs**, showing  $x = y \vee x \neq y$  is not necessary (classical logic).

Manually proven for each data type in the progress proof.

Automated by extending IL2Rocq.

# Decidable Equality Proofs

Based on WasmCert-Rocq's approach [BGP<sup>+</sup>25].

- Fully automated proofs for all data types.
- Proofs are **significantly simplified** compared to WasmCert-Rocq (about 100 to 10 lines).

```
Ltac rect'_build_projection proj rect :=  
  let t :=  
    lazymatch type of rect with  
    | forall P : ?t -> Type, _ => t  
    end in  
  let g := rect'_type_projection proj rect in  
  refine (_ : g);  
  ...
```

```
Definition administrative_instruction_eq_dec : forall e1 e2 :  
  ⇨ administrative_instruction,  
    {e1 = e2} + {e1 <> e2}.  
Proof. decidable_equality_using  
  ⇨ administrative_instruction_rect'. Defined.
```

# Decidable Equality Proofs

Based on WasmCert-Rocq's approach [BGP<sup>+</sup>25].

- Fully automated proofs for all data types.
- Proofs are **significantly simplified** compared to WasmCert-Rocq (about 100 to 10 lines).

```
Create HintDb eq_dec_db.
```

```
Ltac decidable_equality_step :=  
  do [ by eauto with eq_dec_db | decide equality ].
```

```
Fixpoint admininstr_eq_dec (v1 v2 : admininstr ) {struct v1} :  
  {v1 = v2} + {v1 <> v2}.
```

```
Proof. decide equality; do ? decidable_equality_step. Defined.
```

# Project

## First-Order Logic Extension in DSL

# Contributions

- Progress Proof
- Decidable Equality Proofs
- **First-Order Logic Extension in DSL**
- Template Mechanism in DSL

# Motivation

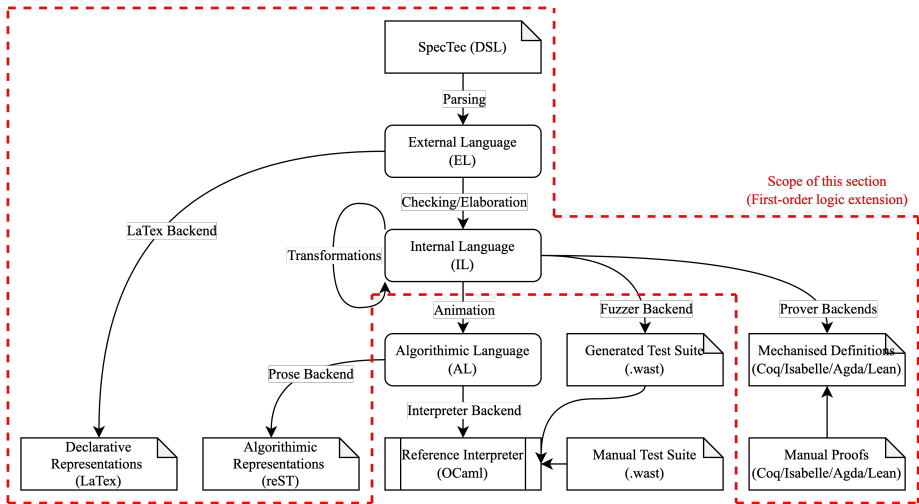
More theorems are expected in the Wasm 3.0 draft [Wor22b]:

- Principal types
- Type lattice
- Properties on compositionality of instruction sequences

Makes sense to specify theorems in the DSL as a **“single source of truth”** [YSL<sup>+</sup>24]:

- Enforces uniform style of theorem statements across theorem provers.
- Allows translation of theorems into  $\text{\LaTeX}$  and prose formats.
- Guarantees that theorems in the specification precisely match the mechanised definitions.

# Scope





# Design

DSL syntax extensions and examples:

```
def ::=
```

```
...
```

```
"theorem" thmid ":" exp hint*
```

```
"lemma" thmid ":" exp hint*
```

```
"theorem" thmid hint* "=" exp
```

```
"lemma" thmid hint* "=" exp
```

```
exp ::=
```

```
...
```

```
"@" "(" relid ":" exp ")"
```

```
"@" "(" exp ")" iter*
```

```
forall args exp
```

```
exists args exp
```

```
theorem t_progress =
```

```
  forall (s, f, admininstr*, t?)
```

```
    @(Config_ok: |- s; f; admininstr* : t?) =>
```

```
    $terminal__form(admininstr*) \/
```

```
    exists (s', f', admininstr'*)
```

```
    @(Step: s; f; admininstr* ~> s'; f'; admininstr'*)
```

# Results

Produced by Coq,  $\text{\LaTeX}$  and prose backends:

```
Theorem t_progress : forall (v_s : store) (v_f : frame) (v_admininstr :  
  ↪ (list admininstr)) (v_t : (option valtype)), ((Config_ok (config__  
  ↪ (state__ v_s v_f) v_admininstr) v_t) -> ((fun terminal_form  
  ↪ v_admininstr) \/ exists (v_s' : store) (v_f' : frame) (v_admininstr'  
  ↪ : (list admininstr)), (Step (config__ (state__ v_s v_f) v_admininstr)  
  ↪ (config__ (state__ v_s' v_f') v_admininstr')))).
```

**Proof.** Admitted.

$$(\text{Progress}) \forall s, f, instr^*, t^?. \vdash s; f; instr^* : t^? \Rightarrow$$
$$\text{terminal\_form}(instr^*) \vee \exists s', f', instr'^*. s; f; instr^* \hookrightarrow s'; f'; instr'^*$$

## Theorem (Progress)

For all  $s, f, instr^*, t^?$ , if  $instr^*$  is valid with type  $t^?$  under the context  $f$  and the store  $s$ , then  $instr^*$  is in terminal form or there exists  $s', f', instr'^*$  such that  $((s, f), instr^*)$  steps to  $((s', f'), instr'^*)$

# Project

## Template Mechanism in DSL

# Contributions

- Progress Proof
- Decidable Equality Proofs
- First-Order Logic Extension in DSL
- **Template Mechanism in DSL**

# Motivation

Proofs tend to be lengthy (about 2000-3000 lines).

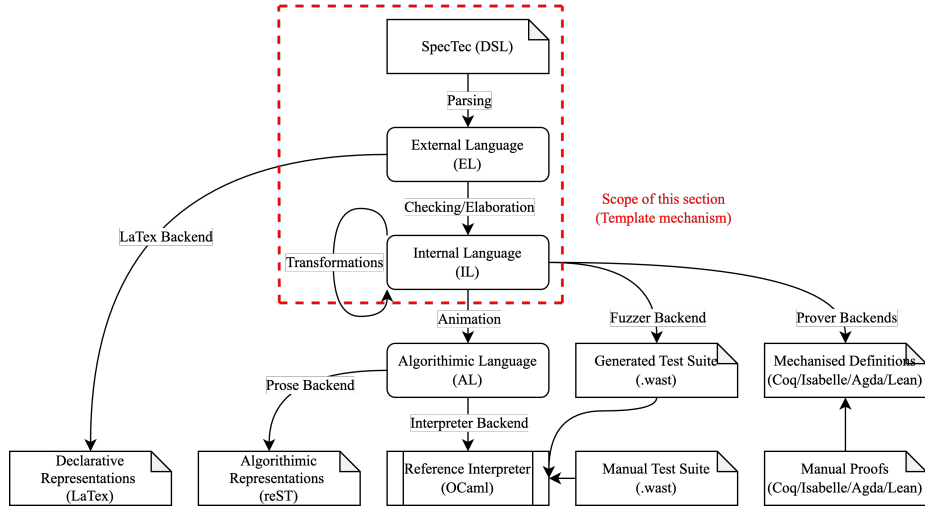
Long proofs should be extracted into **auxiliary lemmas**.

e.g. Individual cases, induction steps and inversion lemmas.

- Enhances readability and maintainability.
- Enforces finer details of proof structure across theorem provers.

But the statements of these lemmas can be quite repetitive!

# Scope



# Design

Auxiliary lemmas **without** template mechanism:

```
lemma Step_pure__preserves_nop = forall (s, C, ft)
  @(Admin_instrs_ok: s; C |- NOP : ft) =>
  @(Step-pure: NOP ~> eps) =>
  @(Admin_instrs_ok: s; C |- eps : ft)

lemma Step_pure__preserves_local_tee = forall (s, C, ft, val, x)
  @(Admin_instrs_ok: s; C |- val (LOCAL.TEE x) : ft) =>
  @(Step-pure: val (LOCAL.TEE x) ~> val val (LOCAL.SET x)) =>
  @(Admin_instrs_ok: s; C |- val val (LOCAL.SET x) : ft)
```

# Design

Auxiliary lemmas **with** template mechanism:

```
template
lemma Step_pure__preserves =
  forall (s, C, ft, {{ ...relations.Step_pure.rules.*.freevars }})
    @(Admin_instrs_ok: s; C |- {{ relations.Step_pure.rules.*.before }} : ft) =>
    @(Step_pure: {{ relations.Step_pure.rules.*.before }} ~>
      {{ relations.Step_pure.rules.*.after }}) =>
    {{ relations.Step_pure.rules.*.premises }} =>
    @(Admin_instrs_ok: s; C |- {{ relations.Step_pure.rules.*.after }} : ft)
```



# Results

Produced by Coq backend:

```
Lemma Step_pure__preserves__nop : forall (v_ft : functype) (v_C : context) (v_s :  
  ⇨ store), ((Admin_instrs_ok v_s v_C [(admininstr__NOP )] v_ft) -> ((Step_pure  
  ⇨ [(admininstr__NOP )] []) -> (Admin_instrs_ok v_s v_C [] v_ft))).
```

Proof. Admitted.

```
Lemma Step_pure__preserves__local_tee : forall (v_x : idx) (v_val : val) (v_ft :  
  ⇨ functype) (v_C : context) (v_s : store), ((Admin_instrs_ok v_s v_C [(v_val :  
  ⇨ admininstr);(admininstr__LOCAL_TEE v_x)] v_ft) -> ((Step_pure [(v_val :  
  ⇨ admininstr);(admininstr__LOCAL_TEE v_x)] [(v_val : admininstr);(v_val :  
  ⇨ admininstr);(admininstr__LOCAL_SET v_x)]) -> (Admin_instrs_ok v_s v_C [(v_val  
  ⇨ : admininstr);(v_val : admininstr);(admininstr__LOCAL_SET v_x)] v_ft))).
```

Proof. Admitted.

# Evaluation

# Progress Proof

## Key strengths:

Identified **key errors in the DSL** [YSL<sup>+</sup>25].

- Not detected by the reference interpreter nor the preservation proof in SpecTec!
- Highlights the importance of developing both proofs in this project.

Identified **key issues in IL2Rocq** [Cup24].

- Handling of conjunction of premises
- Handling of iteration dimensions
- Translation of list slicing notations

Some lemmas were **reformulated from scratch** (not just a direct porting process).

# Progress Proof

Consistent use of **SSReflect** syntaxes.

- Enhances readability and maintainability.
- WasmCert-Rocq contains a mixture of both Rocq and SSReflect syntaxes.

```
Goal 1
l, c : memaddr
ts1, ts2 : seq valtype
C : context
s : store
Hstep : Step_pure
      [:: admininstr__CONST (valtype__INN inn_I32) (c : val_);
        admininstr__BR_IF l] [:: admininstr__BR l]
Hval : (c : val_) ≠ 0
ts1_comp, ts2_comp, ts3_comp, ts4_comp : seq valtype
H1_comp : ts1 = ts1_comp ++ ts2_comp
H2_comp : ts2 = ts1_comp ++ ts3_comp
ts1_comp0, ts2_comp0, ts3_comp0, ts4_comp0 : seq valtype
H1_comp0 : ts2_comp = (ts1_comp0 ++ ts2_comp0)%list
H2_comp0 : ts4_comp = (ts1_comp0 ++ ts3_comp0)%list
H3_comp0 : ts2_comp0 = ts4_comp0
H4_comp0 : Admin_instr_ok s C
      (admininstr__CONST (valtype__INN inn_I32) c)
      (functype__ ts4_comp0 ts3_comp0)
ts1_comp1, ts2_comp1, ts3_comp1, ts4_comp1 : seq valtype
H1_comp1 : ts4_comp = (ts1_comp1 ++ ts2_comp1)%list
H2_comp1 : ts3_comp = (ts1_comp1 ++ ts3_comp1)%list
H3_comp1 : ts2_comp1 = ts4_comp1
ts : seq valtype
ts' : resulttype
H1 : ts3_comp1 = ts ++ ts'
H2 : ts4_comp1 = ts3_comp1 ++ [:: valtype__INN inn_I32]
H3 : (l < Datatypes.length (context__LABELS C))%coq_nat
H4 : lookup_total (context__LABELS C) l = ts'

{! /!}
Admin_instrs_ok s C [:: admininstr__BR l] (functype__ ts1 ts2)
```

```
Goal 1
l, c : memaddr
C : context
s : store
Hstep : Step_pure
      [:: admininstr__CONST (valtype__INN inn_I32) (c : val_);
        admininstr__BR_IF l] [:: admininstr__BR l]
Hval : (c : val_) ≠ 0
ts1, ts2, ts, ts1', ts2', ts3, ts', ts1'', ts3' : seq valtype
Hconst : Admin_instr_ok s C (admininstr__CONST (valtype__INN inn_I32) c)
      (functype__ ts1'' ts3')
ts'', ts3'', ts2'' : seq valtype
Hts' : ts' ++ ts3' = ts'' ++ ts3''
ts'' : seq valtype
tslab : resulttype
Hts2'' : ts2'' = ts'' ++ tslab
Hts3'' : ts3'' = ts2'' ++ [:: valtype__INN inn_I32]
Hlookup : lookup_total (context__LABELS C) l = tslab
Hlen : l < Datatypes.length (context__LABELS C)

{! /!}
Admin_instrs_ok s C [:: admininstr__BR l]
(functype__ (ts ++ ts' ++ ts1'') (ts ++ ts'' ++ ts2''))
```

# Progress Proof

## Future work:

Pending tasks for Wasm mechanisation:

- Update the preservation proof to accommodate the changes in the DSL/IL2Coq.
- Upgrade the proofs to Wasm 2.0 [Wor22a].
- Mechanisation of numerics and module instantiation [Wor22a].

# First-Order Logic Extension in DSL

## Key strengths:

- Backwards compatible/minimally intrusive design.
- Correctness confirmed through **visual comparison** and **migration of proofs**.

## Room for improvements:

- Support quantification of variables of family types.
- Support updating theorem statements iteratively.
- The **@ symbol** is currently required for disambiguation in the LR(1) grammar.

# Template Mechanism in DSL

## Key strengths:

- Backwards compatible/minimally intrusive design.
- Correctness confirmed through **visual comparison** and **migration of proofs**.
- Designed as a **generic mechanism**.
- Allows quantification of free variables in the substituted expressions.

## Room for improvements:

- Handle name conflicts with free variables in the substituted expressions.
- Formalise the process of template expansion.

# Conclusion



# Contributions

- **Progress Proof**
- **Decidable Equality Proofs**
- **First-Order Logic Extension in DSL**
- **Template Mechanism in DSL**

# Contributions

These are all **interconnected** (not random)!

- **Progress Proof**
  - ↓ Identified areas of improvement in SpecTec/IL2Rocq
- **Decidable Equality Proofs**
  - ↓ Required as part of Coq translation
- **First-Order Logic Extension in DSL**
  - ↓ Required to specify template definitions
- **Template Mechanism in DSL**

# Conclusion

Challenges overcome in this project:

- Lots of background material (e.g. Rocq, SSReflect).
- Lots of **code reading** (e.g. WasmCert-Rocq, SpecTec, IL2Coq).
- Lots of **proofs and coding** (e.g. frontend, middleends, backends in SpecTec).
- Exploring various **design choices** for Lemmagen.

Lemmagen only serves as a proof of concept at the moment.

But the reactions have been positive so far!

# IMPERIAL

# Thank You

MEng Individual Project

June 25, 2025

# SpecTec Highlight

For those having a hard time reading the DSL

```
;; Memory instructions
```

```
rule Step_read/load-num-trap:
```

```
  z; (CONST (INN I32) i) (LOAD t mo) ~> TRAP
```

```
  -- if $(i + mo.OFFSET + $size(t)/8 > |$mem(z, 0).BYTES|)
```

```
rule Step_read/load-num-val:
```

```
  z; (CONST (INN I32) i) (LOAD t mo) ~> (CONST t c)
```


```
  -- if $bytes(t, c) = $mem(z, 0).BYTES[i + mo.OFFSET : $size(t)/8]
```


# Errata I


Minor corrections made to the report since its submission:

Location	Error	Correction
p.32-33	Inappropriate use of <code>repeat</code> tactical in Figures 3.26 and 3.27	Replaced with <code>do ?</code> tactical to match SSReflect-native style
p.30, par 5, line 2	Inaccurate description of the proof in Figure 3.24	Clarified that the proof proceeds by establishing the <code>Config_sound</code> co-recursively
p.65, par 3, line 2	Unclear discussion of the limitation imposed on iterative proof development	Clarified that this limitation is particularly relevant due to frequent updates
p.21, par.3	Missing description of the modifications to the <code>Step/ctxt-frame</code> rule	Clarified that this rule has been modified to allow changes in the frame state <code>f</code>

# References I

 M. Bodin, P. Gardner, J. Pichon, C. Watt, and X. Rao.  
Wasmcert-coq.  
GitHub repository, 2025.  
Last accessed: 20 January 2025.

 Diego Cupello.  
Il2coq: Automatic translation of inductive logic programming concepts into coq.  
2024.  
Accessed: 2025-01-13.

 Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien.  
Bringing the web up to speed with webassembly.  
SIGPLAN Not., 52(6):185–200, June 2017.

# References II



Andreas Rossberg.

Spectec has been adopted, March 2025.

Accessed: 2025-06-07.



Conrad Watt.

Mechanising and verifying the webassembly specification.

In Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, page 53–65, New York, NY, USA, 2018. Association for Computing Machinery.



World Wide Web Consortium (W3C).

Webassembly specification release 1.0, 2019.

Last accessed: 20 January 2025.



World Wide Web Consortium (W3C).

Webassembly specification release 2.0, 2022.

Last accessed: 20 January 2025.



# References III



World Wide Web Consortium (W3C).

Webassembly specification release 3.0 (draft 2025-05-15), 2022.

Last accessed: 20 January 2025.



Conrad Watt, Xiaojia Rao, Jean Pichon-Pharabod, Martin Bodin, and Philippa Gardner.

Two mechanisations of webassembly 1.0.

2021.



Dongjun Youn, Wonho Shin, Jaehyun Lee, Sukyoung Ryu, Joachim Breitner, Philippa Gardner, Sam Lindley, Matija Pretnar, Xiaojia Rao, Conrad Watt, and Andreas Rossberg.

Bringing the webassembly standard up to speed with spectec.

Proc. ACM Program. Lang., 8(PLDI), June 2024.

# References IV



Dongjun Youn, Wonho Shin, Jaehyun Lee, Sukyoung Ryu, Joachim Breitner, Philippa Gardner, Sam Lindley, Matija Pretnar, Xiaojia Rao, Conrad Watt, and Andreas Rossberg.

Spectec.

GitHub repository, 2025.

Last accessed: 20 January 2025.