CHUMP

A Mechanized Verification of Curricle

Justin Frank and Pierce Darragh

CMSC 838L

May 1, 2024

Curtailed Curricle Curriculum

- A type system for safe intermittent computing
- Generalization over various implementations of checkpointing
- Support for both just-in-time and atomic intermittent execution
- Inference of checkpointing from (non-)idempotence annotations
- Built on top of Rust

Our Motivation

- Curricle is cool
- Mechanization is *also* cool
- Want a mechanized semantics of intermittent systems
    - Proving correctness on intermittent semantics directly is *hard*
    - Want to write proofs about continuous execution that translate to intermittent behavior

Mechanizing Curricle

- Extended paper has...
  - ~10 pages of typing rules, inference rules, and related material
  - ~10 pages of formal semantics
  - ~30 pages of proofs
- It would be a shame if something were to... *happen* while implementing it all
- Solution: mechanization

Project Outline

- Simplify language and feature set
  - But add a little something new, as a treat
- Decide on a meaning of "correctness"
- Prove it, using adjusted versions of theorems and relations from the paper

Language and Features

Some Simplifications

(Because mechanization is hard)

- Simplified language
  - No structs
  - No functions
- Simplified semantics
  - Only undo checkpointing
  - DINO-style checkpoints
- Simplified type system
  - Separate Curricle from base types
- Simplified machine state
  - No volatile memory

Additional Features

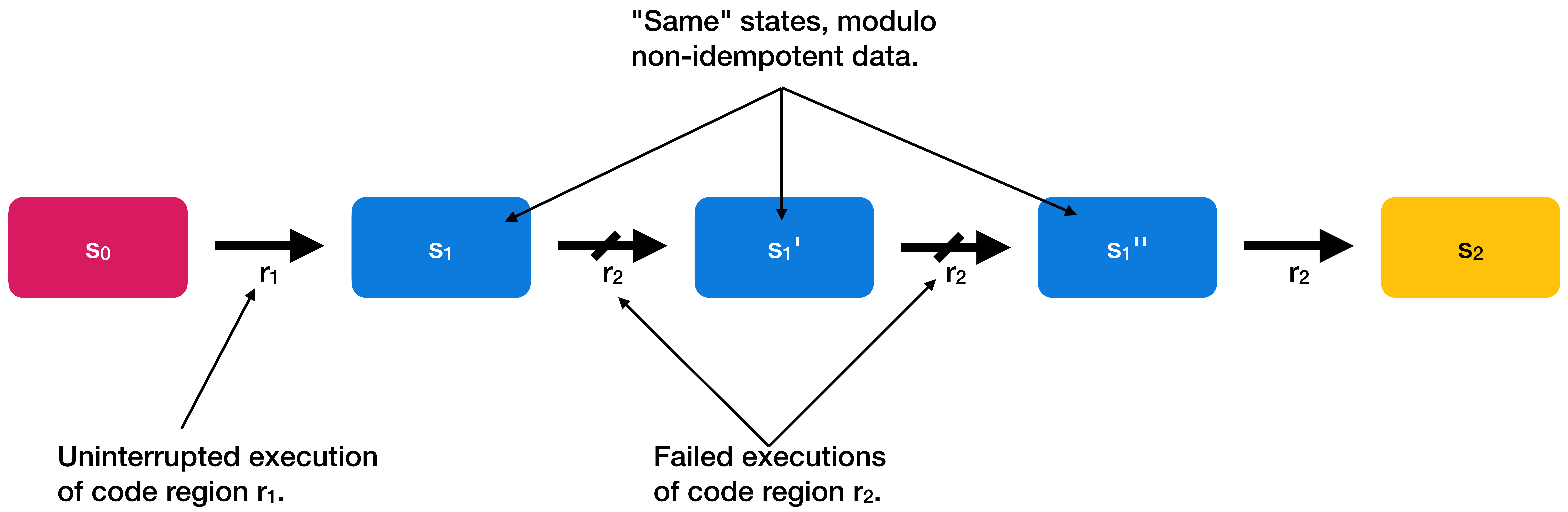
- Support for output in non-JIT intermittent execution
- Unbounded loops with arbitrarily nested breaks

Correctness

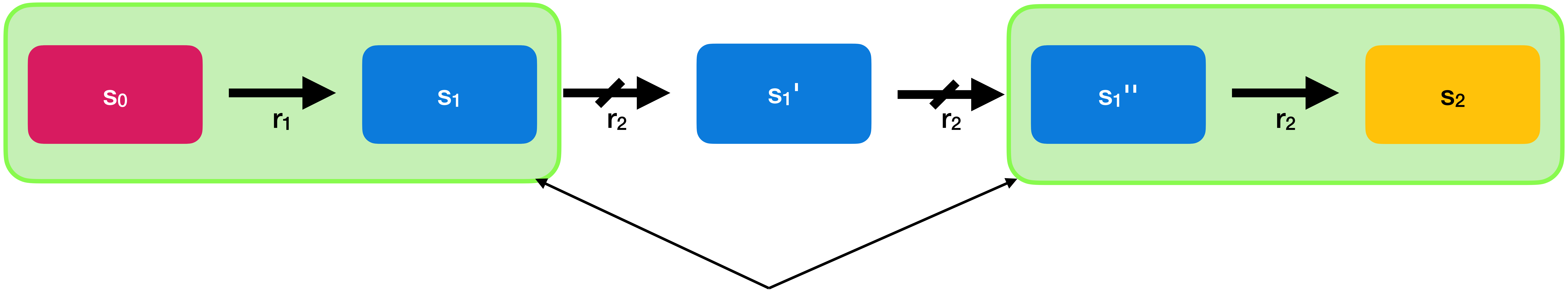
Correctness

$$\begin{tikzcd}
s_0 \arrow[r, "r_1"'] & s_1 \arrow[r, "/" marking, "r_2"'] & s_1' \arrow[r, "/" marking, "r_2"'] & s_1'' \arrow[r, "r_2"'] & s_2
\end{tikzcd}$$

Correctness

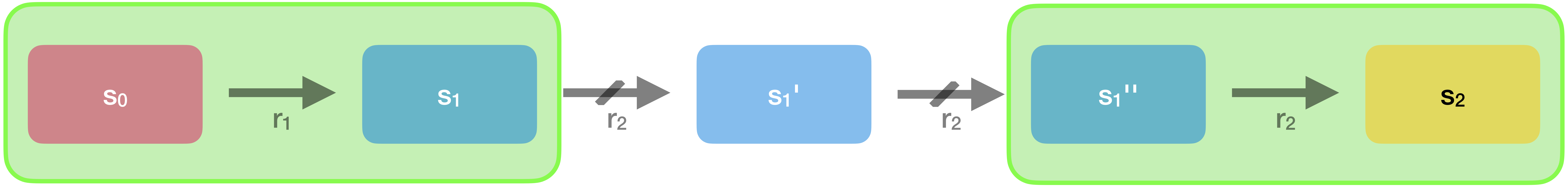


Correctness

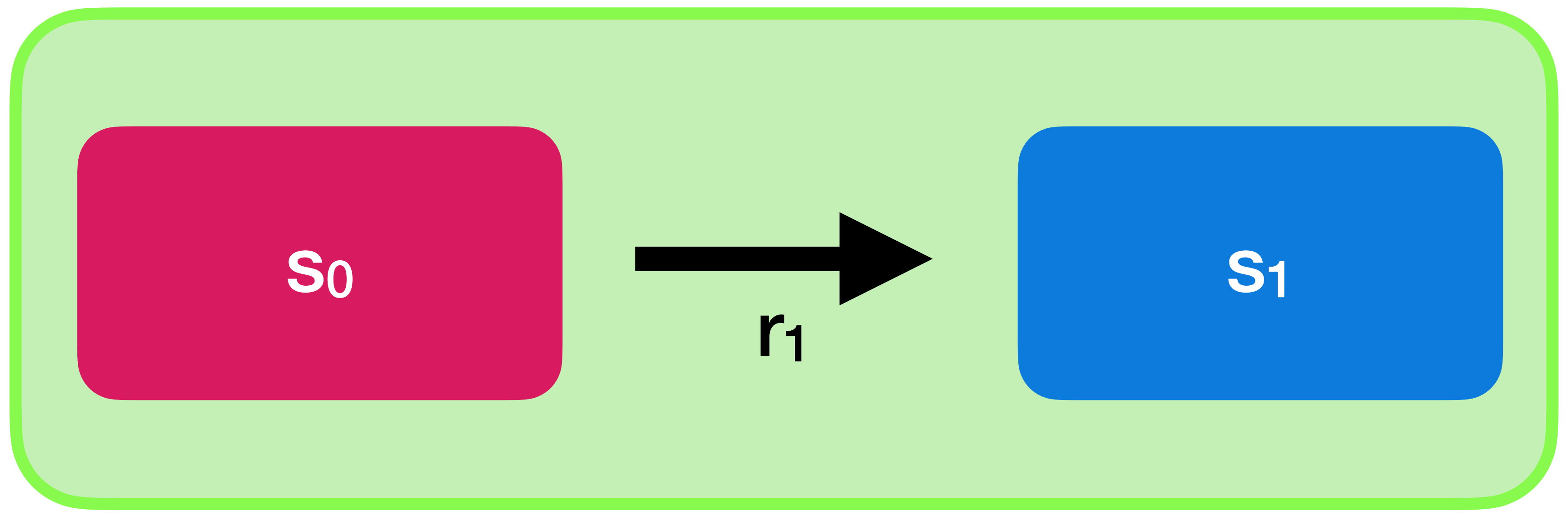


Locally continuous executions.

Correctness

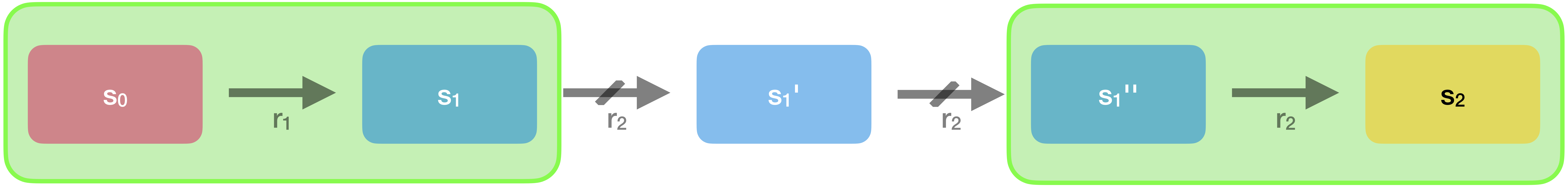


Intermittent execution.

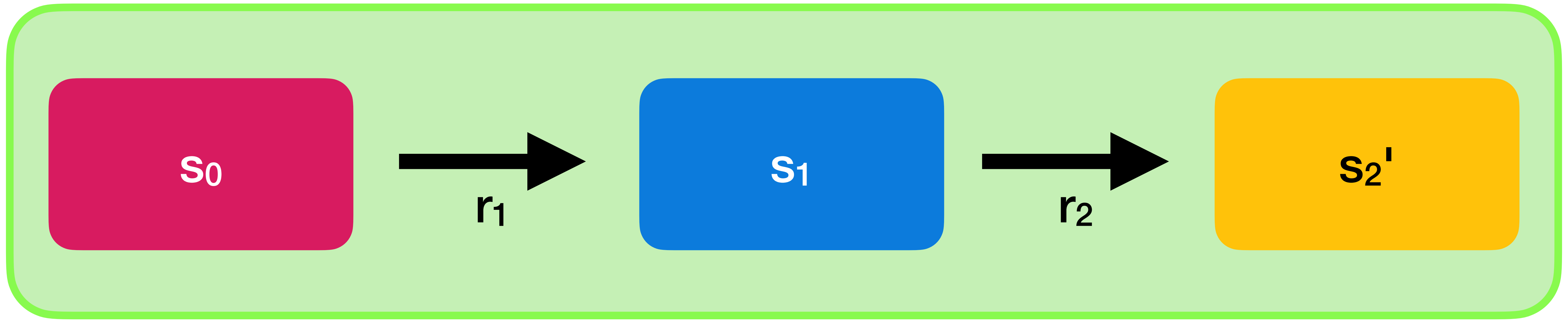


Simulation of continuous execution.

Correctness

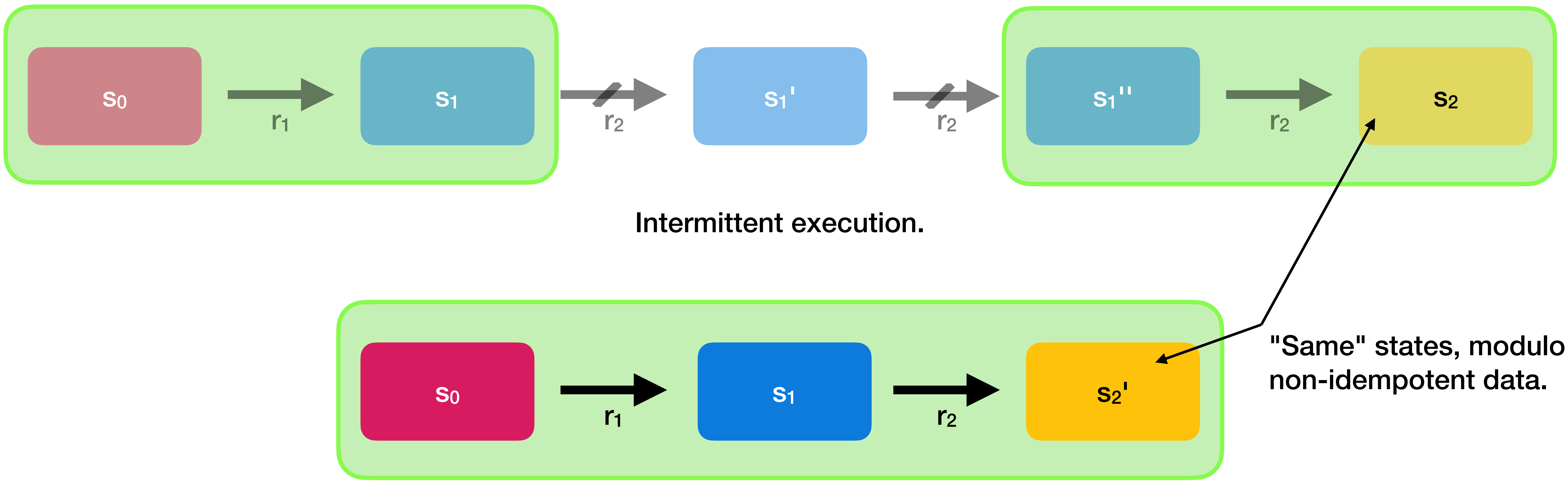


Intermittent execution.



Simulation of continuous execution.

Correctness



Intermittent execution.

"Same" states, modulo non-idempotent data.

Simulation of continuous execution.

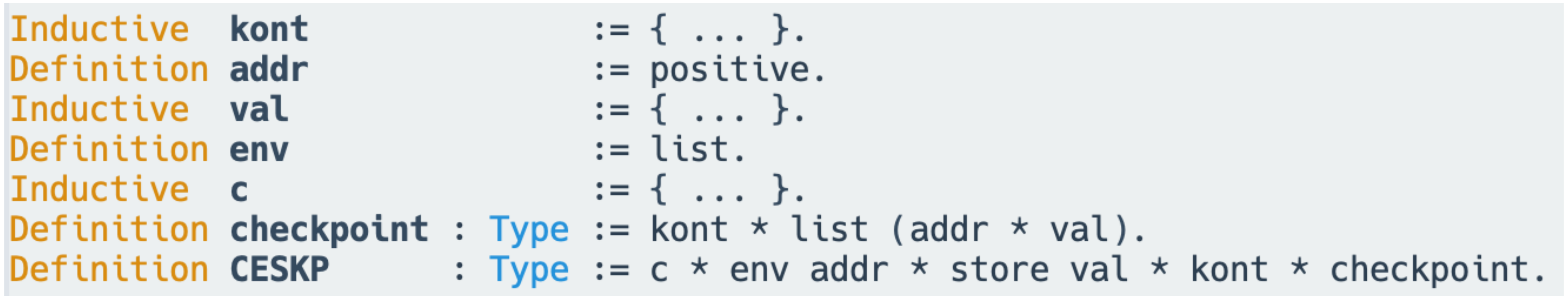
Correctness

- Curricle has a two-part notion of correctness:
    1. Non-interference: power failures do not adversely affect end result
    2. Intermittent exec. w/ no power failures is equivalent to continuous exec.
- We changed it a little bit:
    - $\forall$ exec., $\exists$ reset-free exec. equivalent to the original exec.

Proof Approach

Machine Model

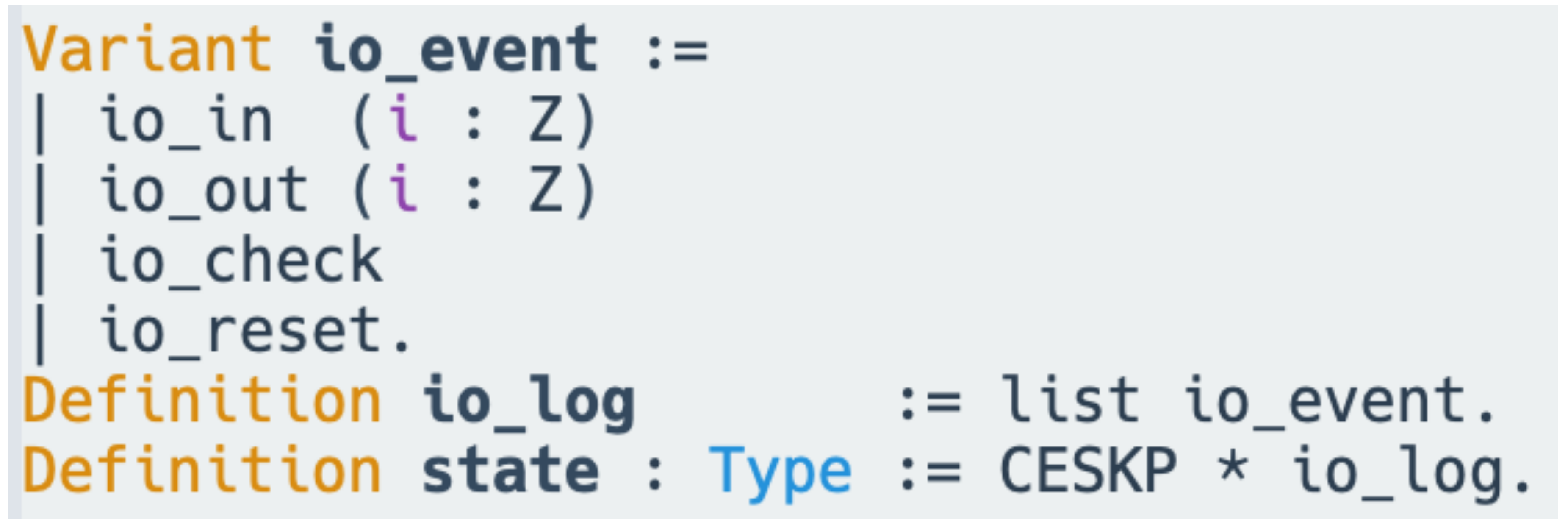
- Extended CESK with checkpoints, making CESKP
    - Control, Environment, Store, Kontinuation, checkPoint
- CESK is a semantic model both team members are comfortable with
- Regularly used in mechanized formalizations (so should be "easy" to adapt)



```
Inductive  kont                  := { ... }.
Definition addr                  := positive.
Inductive  val                   := { ... }.
Definition env                   := list.
Inductive  c                     := { ... }.
Definition checkpoint : Type := kont * list (addr * val).
Definition CESKP      : Type := c * env addr * store val * kont * checkpoint.
```

The Outside World

- Input and output are modeled as external constructs
    - i.e., separate from the CESKP machine state
- Resets are considered IO events



```
Variant io_event :=
| io_in  (i : Z)
| io_out (i : Z)
| io_check
| io_reset.
Definition io_log        := list io_event.
Definition state : Type := CESKP * io_log.
```

Separating the Type Systems

- Curricle's type system is all-in-one
- We separate base types from Curricle's type qualifiers

$$\dfrac{\begin{array}{c} \Pi(p) = l_p \qquad \Gamma, \Pi, \mathsf{Wts} \vdash^{atom}_{e} e : t@q_e \Rightarrow \Gamma' \qquad \Gamma', \Pi, \mathsf{Wts} \vdash^{atom}_{elft} p : @q_l \Rightarrow \Gamma'_l \\ \langle qId_e, qIO_e \rangle = q_e \sqcup \langle qId_{pc}, qIO_{pc} \rangle \sqcup q_l \qquad \Gamma'_l(l_p) = t@\langle qId_p, qIO_p \rangle \\ qId_e \sqsubseteq qId_p \downarrow \qquad qId_p \uparrow^{qAcc} \leq_a \mathsf{Wt} \qquad \mathsf{UpdatePoint}(\Pi, p, e : t) = \Pi' \end{array}}{\Gamma, \Pi, \mathsf{Wts}, \langle qId_{pc}, qId_{pc} \rangle \vdash^{atom}_{c} p := e \Rightarrow \Gamma'_l[\Pi'(p) : \langle qId_p, qIO_e \rangle], \Pi', (\mathsf{Wts}, \Pi'(p))} \ \textsc{AssgnSingle}$$

Energy Model

- Curricle uses logical energy
  - Energy is encoded into the proofs as a decreasing natural number
  - Energy = 0 $\implies$ power failure
- We use a nondeterministic semantic model
  - Any state can transition to a reset instead of its usual next step

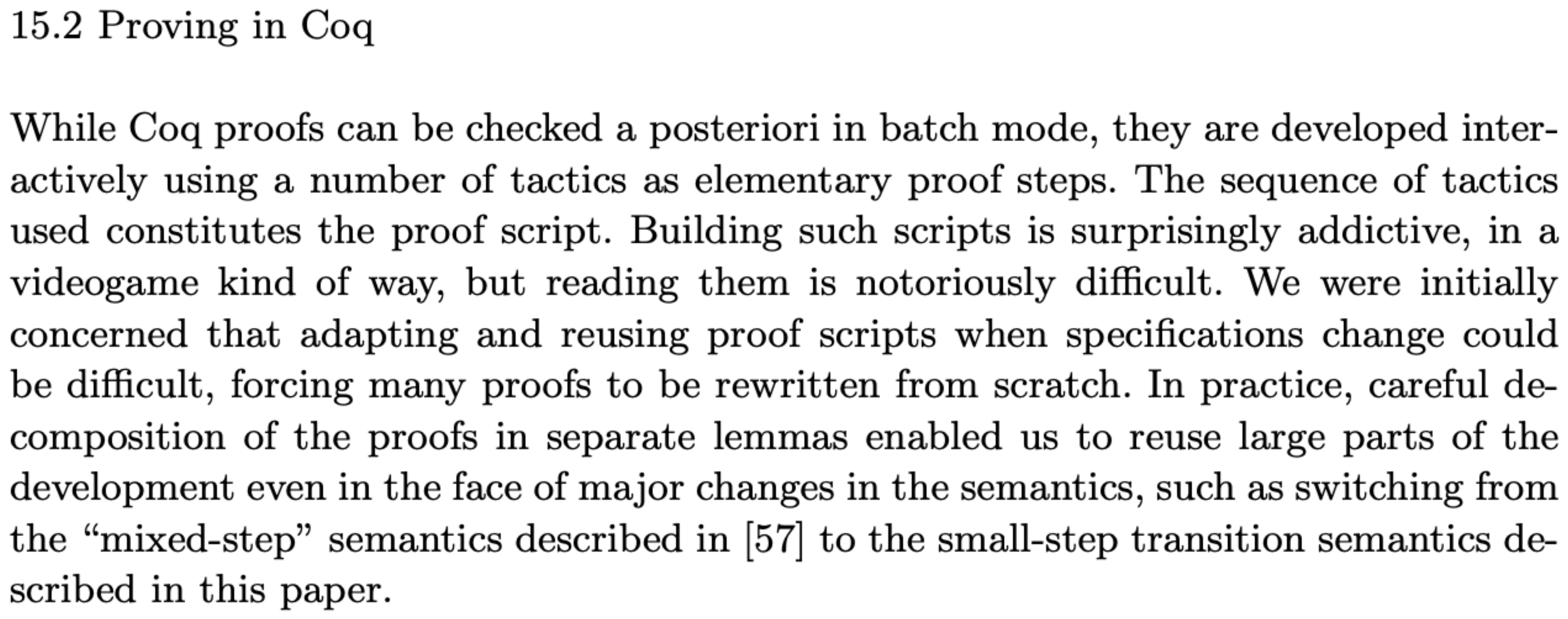
Side Quests

- Proofs of *progress* and *preservation* in the base type system
  - ~800 lines of Rocq
- (Most of) a formalization of lattices for the Curricle types
  - ~100 lines of Rocq

Side Quests

- Proofs of *progress* and *preservation* in the base type system
  - ~500 lines of Rocq
  - Not necessary
- (Most of) a formalization of lattices for the Curricle types
  - ~100 lines of Rocq
  - Not necessary

Side Quests



15.2 Proving in Coq

While Coq proofs can be checked a posteriori in batch mode, they are developed interactively using a number of tactics as elementary proof steps. The sequence of tactics used constitutes the proof script. Building such scripts is surprisingly addictive, in a videogame kind of way, but reading them is notoriously difficult. We were initially concerned that adapting and reusing proof scripts when specifications change could be difficult, forcing many proofs to be rewritten from scratch. In practice, careful decomposition of the proofs in separate lemmas enabled us to reuse large parts of the development even in the face of major changes in the semantics, such as switching from the "mixed-step" semantics described in [57] to the small-step transition semantics described in this paper.

— "A formally verified compiler back-end", Xavier Leroy, 2009.

What Remains

Current Status

- Still implementing Curricle-based type system
- Getting to this point has taken *way* more work than we'd thought
- Plan to finish basic proofs before the end of the semester

Future Work

- Add missing language features (structs, functions, ...?)
- Implement a DSL (maybe in Racket, just for fun)

End.

Why "CHUMP"?

- IMP is a basic imperative language used for modeling semantics
- CHeckpoints + Undo + iMP = CHUMP