Certified Infinite Descent Criteria in Isabelle/HOL

Jamie Wright
University Of Sheffield, UK

Liron Cohen
Ben-Gurion University of the Negev, Israel

Reuben N. S. Rowe
Royal Holloway University of London, UK

Andrei Popescu
University Of Sheffield, UK

Abstract

Infinite Descent is the global trace condition that underpins the soundness of cyclic reasoning and, in program analysis, the size change termination principle. Many (semi-)decision procedures for Infinite Descent are known, based on criteria ranging from automata-based constructions and relation-based characterizations, to effective (but incomplete) heuristics. Although these criteria are well studied on paper and implemented in tools, a unified, machine-checked account that relates them to the (abstract) Infinite Descent property has been missing. We present an Isabelle/HOL mechanization of this landscape. We develop a reusable, locale-based framework of sloped graphs that defines Infinite Descent at an abstract level, independently of any concrete graph encoding. Within this framework we formalize standard *complete* criteria and prove their equivalence to the locale-level `InfiniteDescent` predicate. We also formalize tool-facing sufficient criteria, prove their soundness, and certify incompleteness where appropriate via verified counterexamples. Along the way we contribute reusable Isabelle lemmas for ω-regular reasoning over streams and for Büchi-automata constructions needed by the inclusion proofs.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Logic and verification; Theory of computation → Automated reasoning

Keywords and phrases Cyclic Proof, Infinite Descent, Size-Change termination, Büchi automata

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.14

Supplementary Material Persistent DOI and browseable version for Isabelle formalization
Software (Zenodo artifact): `https://doi.org/10.5281/zenodo.20323853` [30]
InteractiveResource (Browseable source code): `jamierwright.github.io/InfiniteDescent` [31]

Funding Jamie Wright, Liron Cohen, and Andrei Popescu were supported by the Royal Society travel grant "Cyclic Reasoning Mechanisms for Interactive Theorem Proving". Liron Cohen was additionally supported by ISF Grant No. 2876/25. Jamie Wright and Andrei Popescu were additionally supported by: EPSRC grant EP/X015114/1 "COVERT"; and "Copilots for Isabelle" under the AI for Math Fund, an initiative by Renaissance Philanthropy with funding support from XTX Markets.

1 Introduction

Cyclic reasoning has emerged as a powerful technique for automated verification, enabling proofs of inductive properties without explicit inductive invariants [26, 24, 12, 5, 14, 11, 9]. The soundness of cyclic reasoning is underpinned by a global validity condition known as Infinite Descent. In a cyclic proof graph, Infinite Descent requires that every infinite path admits some infinitely progressing (i.e., decreasing) trace. In termination analysis this condition is widely known as the size change principle [20, 16, 21].

Because deciding Infinite Descent is PSPACE-complete, practical tools rely on a range of equivalent characterizations and fast sufficient checks that may return "don't know". A number

of different algorithms have been developed, based on these criteria, including Büchi automata-theoretic formulations [20, 4, 25] and algebraic closure-based characterizations [16, 8]. The recent Cyclone tool also incorporates a number of fast (polynomial time) semi-algorithms, based on sufficient (but not necessary) criteria for Infinite Descent [10].

To achieve end-to-end trust in verification systems that rely on these algorithms, we need machine-checked guarantees that each alternative criterion is sound and, where appropriate, complete with respect to Infinite Descent. This paper supplies that missing layer of certification, using Isabelle/HOL. We begin by developing a locale-centric framework for *sloped* graphs—the abstract setting in which the Infinite Descent property is expressed most generally. This isolates the mathematical core of Infinite Descent from any particular decision procedure or concrete encoding.

Building on this abstract definition, we then formalize the equivalent criteria that form the basis of existing algorithms for deciding Infinite Descent [8]. Specifically, we mechanize the 'vertex-language' and 'slope-language' automata-based criteria, and the algebraic relation-based criteria, connecting them to the Infinite Descent property via machine-checked proofs of equivalence. We also formalize the effective sufficient criteria used in Cyclone, as well as (a generalisation of) a sufficient criterion considered by Sprenger and Dam [27]. For each we provide machine-checked proofs of soundness and demonstrate incompleteness via verified counterexamples.

Taken together, our Isabelle development can serve as a certification layer for both existing and future tool pipelines. Once a concrete problem instance is encoded as a sloped graph, Isabelle can validate an Infinite Descent judgement by checking any one of the certified criteria inside our framework, which are underpinned by formalized soundness theorems showing that they entail Infinite Descent. In particular, an external tool may act as a fast front end that searches for evidence that a chosen criterion holds (for example, a closure summary for the algebraic criterion, or a finite witness shape suitable for the automata-based criteria), and Isabelle can then replay that evidence against our definitions. At present we do not provide a general proof reconstruction pipeline from arbitrary tool output into Isabelle proofs; rather, we formalize the semantic core and the correctness theorems that such integrations would build on. This supports a workflow in which tools such as Cyclone remain the fast front end check, while our Isabelle development plays the role of trusted verifier. More broadly, we view this development as a candidate backend for future Isabelle integrations, for example to discharge termination obligations in a more Sledgehammer-style workflow.

Certified size change termination has been mechanized in Isabelle in several settings, including rewriting via IsaFoR/CeTA [28, 19] and earlier work on SCT certificates [18]. Beyond Isabelle, size-change ideas also surface in other ecosystems. For example, systems such as Idris [15] and Agda [13] implement SCT-based termination checking as part of typechecking and PVS mechanizes and compares several termination criteria (including size-change variants) and connects them to PVS termination obligations [22]. Our contribution is orthogonal: we mechanize *Infinite Descent itself* as an abstract semantic condition, with size-change termination and cyclic-proof soundness appearing as instances, so that multiple criteria can be compared and reused in one framework.

Beyond the core meta-theory, our mechanization also required non-trivial engineering around existing Isabelle infrastructure. As a by-product, we contribute reusable lemmas for ω-regular reasoning over streams and strengthen existing libraries for Büchi automata constructions used in the inclusion proofs. Our Isabelle mechanization is available, as both executable source code [30] and browseable HTML [31]. Throughout the paper, we hyperlink

to relevant parts of the latter, indicated by  icons.

Contributions.

The main contributions of our work are:

- **Isabelle framework for Infinite Descent.** A locale-based framework for sloped graphs, traces, and the abstract Infinite Descent property.
- **Certified decision criteria.** Machine-checked soundness and completeness for (i) the various Büchi criteria and (ii) the complete criterion based on relational closure.
- **Certified tool-facing heuristics.** Machine-checked soundness, plus verified incompleteness where appropriate, for several sufficient criteria, including checks used in Cyclone. We also present and formalize a novel sufficient criterion (XSD).
- **Library and engineering contributions.** Reusable extensions to Isabelle libraries for Büchi automata and infinite streams supporting the proof architecture used by the above.

Outline.

For readability, and to keep the paper reasonably self contained, each technical section first recalls the relevant mathematical theory and then presents the corresponding Isabelle/HOL development, discussing its key engineering choices. The remainder of the paper is organised as follows. Section 2 introduces sloped graphs and our abstract formulation of Infinite Descent; presents the locale based Isabelle framework; and illustrates it via a concrete instantiation. Section 3 focuses on complete criteria for Infinite Descent: section 3.1 covers the Büchi automata criteria, and section 3.2 covers the relation-based closure criterion. Section 4 examines sound but incomplete criteria: section 4.1 revisits Sprenger and Dam's criterion and presents our generalized variant, XSD; and section 4.2 deals with the cycle-based heuristics used in Cyclone. Finally, Section 5 concludes.

2 Infinite Descent in Sloped Graphs

We begin by describing our mechanization of the abstract formalization of the Infinite Descent property. We follow the formulation, given in [8], in terms of *sloped graphs*.

2.1 Underlying Mathematical Theory

Sloped graphs are standard directed graphs decorated with just the additional information necessary to describe Infinite Descent. Each nodes is associated with a (sub)sets of *positions* (taken from a set Pos), and edges are decorated with relations that describe the 'slopes' connecting positions of the source node to those of the target node. More specifically, we consider a set $\mathsf{Slope} = \{\rightsquigarrow, \searrow\}$ of slopes comprising a non-strict and strict decrease, respectively. As such, sloped graphs provide a general setting into which concrete applications of the Infinite Descent principle (e.g. program termination, correctness of cyclic proofs, etc.) can be mapped.

▶ **Definition 1.** *A* sloped graph $\mathsf{SG} = (\mathsf{Node}, \mathsf{Edge}, \mathsf{PosOf}, \mathsf{R})$ *is a finite directed graph* $(\mathsf{Node}, \mathsf{Edge})$ *augmented with:*

- *a function* $\mathsf{PosOf} : \mathsf{Node} \to \mathscr{P}(\mathsf{Pos})$ *assigning a finite set of* positions *to each node;*
- *a mapping* R *assigning to each* $(nd, nd') \in \mathsf{Edge}$ *a* sloped relation $\mathsf{R}_{nd,nd'} \subseteq \mathsf{PosOf}(nd) \times \mathsf{PosOf}(nd') \times \mathsf{Slope}$ *such that if* $\mathsf{R}_{nd,nd'}(p, p', s)$ *and* $\mathsf{R}_{nd,nd'}(p, p', s')$ *then* $s = s'$.

Thus, sloped relations are *single-valued* in the slope component. This information could also be modelled as a (partial) function from pairs of nodes to slopes. However, later, a notion of *composition* for this data will be important, and so it seems preferable to work with (this restriction of) the more general concept of relations.

To express Infinite Descent, we must talk about infinite paths (or *ipaths*) within (sloped) graphs. These are infinite sequences $(nd_i)_{i\in\mathbb{N}}$ of nodes that are pairwise connected via edges in the graph, i.e. such that $\forall i \in \mathbb{N}.\ (nd_i, nd_{i+1}) \in \mathsf{Edge}$. The notion of descent then derives from being able to construct a trace consisting of positions, taken from each successive node of an ipath, that are connected via some slope in the relation associated with the corresponding edge.

▶ **Definition 2.** *Given a sloped graph* $\mathsf{SG} = (\mathsf{Node}, \mathsf{Edge}, \mathsf{PosOf}, \mathsf{R})$*, a* trace *for an ipath* $(nd_i)_{i\in\mathbb{N}}$ *in* SG *is an infinite sequence* $(p_i)_{i\in\mathbb{N}}$ *of positions such that* $\forall i \in \mathbb{N}$*,* $p_i \in \mathsf{PosOf}(nd_i)$ *and* $\exists s \in \mathsf{Slope}.\ \mathsf{R}_{nd_i,nd_{i+1}}(p_i, p_{i+1}, s)$*. The trace is said to be* decreasing *if for all* $i \in \mathbb{N}$*:* $\exists j \geq i.\ \mathsf{R}_{nd_j,nd_{j+1}}(p_j, p_{j+1}, \searrow)$*. An ipath* $(nd_i)_{i\in\mathbb{N}}$ *is said to be* descending *if there exists* $j \in \mathbb{N}$ *and a decreasing trace for the tail* $(nd_i)_{i>j}$*.*

Infinite Descent, a property quantifying over infinite paths, is then simply stated.

▶ **Definition 3.** *A sloped graph* SG *is said to satisfy* Infinite Descent *when all its ipaths are descending.*

2.2 Isabelle Mechanization

Our mechanization utilizes Isabelle's module system, known as locales [17, 3]. Within a locale, definitions and theorems are formulated relative to fixed entities (types, constants and assumptions). However, from outside, these are polymorphic in the locale's fixed types, universally quantified over the locale's constants, and conditional upon the locale's assumptions. Locales can be extended, permitting the use of inheritance for these fixed types, constants and assumptions.

In our Isabelle mechanization, we first define a locale for modelling directed graphs, fixing a set of nodes from an abstract data type, `'node`, and representing the edge relation as a characteristic function over this abstract data type.

```
locale Graph =
fixes Node :: "'node set"
and edge :: "'node ⇒ 'node ⇒ bool"
```

Using Isabelle's codatatypes, we model ipaths in this locale as a predicate on streams (i.e., infinite sequences) over the `'node` abstract data type.

```
definition ipath :: "'node stream ⇒ bool" where
"ipath ≡ alw (holdsS Node) aand alw (holds2 edge)"
```

The definition of this predicate utilizes a shallow embedding of LTL from the Isabelle library [1] (providing temporal operators `alw` for always, `ev` for eventually, and `infs` for infinitely often) and our own constants `holdsS` and `holds2` that "lift" point-wise and pair-wise predicates, respectively, to the level of streams. Thus, an ipath is a `'node` stream, each of whose elements belong to the fixed `Node` set and whose pair-wise consecutive elements are related by the fixed `edge` predicate.

To mechanize the concept of sloped graphs, we first define a simple data type `slope` with constructors `Main`(tain) and `Decr`(ease) (representing $\rightsquigarrow$ and $\searrow$, respectively) and then extend the `Graph` locale with further parameters and assumptions.

```
locale Sloped_Graph = Graph Node edge
for Node :: "'node set" and edge :: "'node ⇒ 'node ⇒ bool" +
fixes PosOf :: "'node ⇒ 'pos set"
and RR :: "('node × 'pos) ⇒ ('node × 'pos) ⇒ slope ⇒ bool"
assumes Node_finite: "finite Node"
and PosOf_finite: "⋀nd. nd ∈ Node ⟹ finite (PosOf nd)"
and RR_PosOf: "⋀nd P nd' P' sl. RR (nd,P) (nd',P') sl ⟹
                    {nd,nd'} ⊆ Node ∧ P ∈ PosOf nd ∧ P' ∈ PosOf nd'"
and RR_SlopeRels:
  "⋀nd nd'. {nd,nd'} ⊆ Node ⟹ (λP P'. RR (nd,P) (nd',P')) ∈ SlopedRels"
```

In particular, the `Sloped_Graph` locale is (additionally) parameterized by the position assignment function `PosOf` and the predicate `RR` specifying the collection of sloped relations. We also impose some well-formedness assumptions: the sets of nodes and positions must be finite, and `RR` must be consistent with the graph structure. We also require `RR` to characterise relations that are single-valued in the `slope` component (cf. theorem 1), expressed externally to the locale via the definition of a set `SlopedRels`.

We point out a deliberate divergence from theorem 1 in the type we choose for the predicate `RR`: rather than `('node × 'node) ⇒ ('pos × 'pos × slope) ⇒ bool`, a perhaps more obvious choice for an edge-indexed family, we instead use a "flattened" representation that considers slopes between node-position pairs: `('node × 'pos) ⇒ ('node × 'pos) ⇒ slope ⇒ bool`. This is more convenient for our preferred representation of traces as streams of such node-posistion pairs, allowing for a 'point-free' combinator-based style of reasoning using LTL operators. For example, we give the following definition of descending ipaths in our formalisation.

```
definition descentIpath :: "'node stream ⇒ 'pos stream ⇒ bool" where
"descentIpath nds Ps ≡
 ev (alw (holds2
            (λ(nd,P)(nd',P'). RR (nd,P) (nd',P') Main ∨ RR (nd,P) (nd',P') Decr))
       aand alw (ev (holds2 (λ(nd,P)(nd',P'). RR (nd,P) (nd',P') Decr))))
    (szip nds Ps)"
```

Here, we lift the action of `RR` on node-position pairs, via LTL operators, to a predicate on streams of such pairs (i.e. a trace). The `szip` operator, from Isabelle's stream library, is then used to construct the trace to which the predicate is applied. We prove this is equivalent to the index-based formulation of theorem 2.

Finally, we mechanize the Infinite Descent condition via, essentially, a direct translation of theorem 3.

```
definition InfiniteDescent :: bool where
"InfiniteDescent ≡ ∀nds. ipath nds ⟶ (∃Ps. descentIpath nds Ps)"
```

2.3 Instantiating the Abstract Framework

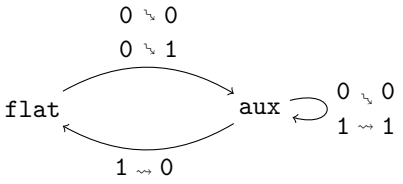
We can already demonstrate the utility of our framework, instantiating the `Sloped_Graph` locale with a concrete model suitable for reasoning about size-change termination, e.g. as used in Agda [13].

Given a collection of (mutually recursive) function definitions, and a (well-founded) notion of size on the data values over which they operate, a recipe for reducing the termination problem to Infinite Descent could proceed by generating a sloped graph as follows (cf. [20]).

- Each function f induces one node $\hat{f}$ of the sloped graph, and the positions of $\hat{f}$ correspond to (the indices of) the formal parameters of f.

```
fun flat and aux where
   "flat [] = []"
 | "flat (l#ls) = aux l ls"
 | "aux [] ls = flat ls"
 | "aux (x#xs) ls = x # aux xs ls"
```

(a) Isabelle definitions of functions `flat` and `aux`.



(b) The induced sloped graph.

Figure 1 The functions `flat` and `aux`, and the corresponding induced sloped graph.

- An edge between nodes $\hat{f}$ and $\hat{g}$ is induced by a call to g within the definition of f,[1] and is assigned a sloped relation that relates position i to position j as follows: via slope $\searrow$ when the j^{th} argument of the call is (syntactically) known to be (strictly) smaller than the i^{th} parameter of f; via slope $\rightsquigarrow$ if the j^{th} argument of the call is (syntactically) equal to the the i^{th} parameter of f; and does not relate them at all if neither is the case.

The functions are terminating if this induced sloped graph satisfies Infinite Descent (the converse does not necessarily hold, of course).

For example, consider the function `flat` for flattening a list of lists and its mutually recursively defined helper function `aux`, shown in fig. 1a (taken from [2, §1.4]). The (infix) '`#`' operator is Isabelle's notation for the list constructor. Thus, notice that the arguments of the call to `aux` within the definition of `flat` (i.e. `l` and `ls`) are obtained by destructing the parameter (`l#ls`), so we have `l` $\leq$ `l#ls` and `ls` $\leq$ `l#ls`. Similarly, in the second clause of the definition for `aux`, we have `xs` $\leq$ `x#xs`. The corresponding sloped graph, induced by the construction described above, is shown in fig. 1b. We annotate the edges with the individual elements belonging to the associated sloped relation, e.g. writing `0` $\searrow$ `0` instead of $(0, 0, \searrow)$ as a notational short-hand.

To represent the sloped graph, we define a concrete `node` data type with constructors for each function, and then define a set `Node` containing all the values of this data type. For concrete positions we take natural numbers, defining a type synonym `pos`. We then define a mapping `PosOf`, with `PosOf(flat) = {0}` and `PosOf(aux) = {0,1}`, since `flat` and `aux` have 1 and 2 parameters, respectively. The concrete predicate specifying the `edge`s of the sloped graph is straightforwardly defined. The data for the sloped relations is defined as a concrete set `RR_Set` of tuples, containing e.g. tuples `((Aux, 0), (Aux, 0), Decr)` and `((Aux, 1), (Aux, 1), Main)` representing how the parameters of `aux` relate to the arguments passed to the recursive call. This is used to define a concrete relation `RR` of the type required by the `Sloped_Graph` locale. Finally, we use these concrete definitions to instantiate the locale. Because they are based on finite sets and discrete data types, the proof obligations required by the locale can be discharged automatically by the `auto` tactic.

```
interpretation Sloped_Graph where
  Node = Node and edge = edge and PosOf = PosOf and RR = RR
 apply unfold_locales
  by(auto simp: RR_defs SlopedRels_def Node_def elim: PosOf.cases)
```

With this instantiation, we can then utilize the framework to provide proofs of Infinite Descent for this graph using the various criteria we discuss below.

[1] Without loss of generality, we can assume that a function f contains only a single call to any given function g: otherwise, introduce 'auxiliary' functions for each call and replace the calls to g within f by calls to the new auxiliary functions; each auxiliary function then simply calls g.

3 Equivalent Criteria for Infinite Descent

This section concerns two families of alternative criteria that are logically *equivalent* to Infinite Descent, and are used as the basis for decision procedures. While these criteria are already well-established, our contribution is their Isabelle/HOL mechanization within the sloped graph locale, and formal proofs of their soundness and completeness relative to the locale-level `InfiniteDescent` predicate.

3.1 Automata-based criteria

The first family of criteria reduces Infinite Descent to a language inclusion problem by interpreting (descending) ipaths as words in an ω-regular language [20, 4, 25, 8]. We describe two approaches for constructing this interpretation, which (following the terminology of [8]) we refer to as the 'vertex-language' and 'slope-language' criteria, respectively. Note that whilst we use the terminology "node" rather than "vertex" in the current work, we continue to use the name 'vertex-language' when referring to the existing automata-based construction in order to maintain consistency with the previous literature.

3.1.1 Underlying Mathematical Theory

We first recall the standard notion of *Büchi automaton*, which is a tuple $\mathcal{B} = (\Sigma, Q, q_0, \Delta, F)$ with finite alphabet Σ, states Q, initial state q_0, transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and accepting set F. An ω-word is a stream $w = (a_i)_{i\in\mathbb{N}}$ over Σ, and a run over w is a stream $(s_i)_{i\in\mathbb{N}}$ with $s_0 = q_0$ and $(s_i, a_i, s_{i+1}) \in \Delta$ for all i. The run is accepting if it visits F infinitely often, and $\mathsf{Lang}(\mathcal{B})$ is the set of ω-words admitting an accepting run. For the following, we fix a sloped graph $\mathsf{SG} = (\mathsf{Node}, \mathsf{Edge}, \mathsf{PosOf}, \mathsf{R})$.

Vertex-language Criterion

The Büchi automata of the vertex-language construction recognize words over the alphabet $\Sigma = \mathsf{Node}$. The *path* automaton $\mathsf{PAut}_{\mathsf{Node}}(\mathsf{SG})$ recognizes exactly the ipaths of SG, and the *trace* automaton $\mathsf{TAut}_{\mathsf{Node}}(\mathsf{SG})$ recognizes exactly the descending ipaths of SG.

▶ **Definition 4** (Path automaton). *The automaton* $\mathsf{PAut}_{\mathsf{Node}}(\mathsf{SG}) = (\Sigma, Q, q_0, \Delta, F)$ *consists of* $Q = \mathsf{Node} \cup \{q_0\}$ *where* $q_0 \notin \mathsf{Node}$, $F = \mathsf{Node}$, *and* $\Delta = \{(q_0, nd, nd) \mid nd \in \mathsf{Node}\} \cup \{(nd, nd', nd') \mid (nd, nd') \in \mathsf{Edge}\}$.

A run of $\mathsf{PAut}_{\mathsf{Node}}(\mathsf{SG})$ first chooses an initial node (via q_0) and then follows edges while consuming the visited vertices. Thus, $\mathsf{Lang}(\mathsf{PAut}_{\mathsf{Node}}(\mathsf{SG})) = \mathsf{IPath}(\mathsf{SG})$, since all runs are accepting.

▶ **Definition 5** (Trace automaton). *The automaton* $\mathsf{TAut}_{\mathsf{Node}}(\mathsf{SG}) = (\Sigma, Q', q_0', \Delta', F')$ *consists of* $Q' = \{q_0'\} \cup \{(nd, p, s) \mid nd \in \mathsf{Node} \wedge p \in \mathsf{PosOf}(nd) \wedge s \in \mathsf{Slope}\}$ *with* q_0' *fresh,* $F' = \{(nd, p, \searrow) \mid nd \in \mathsf{Node} \wedge p \in \mathsf{PosOf}(nd)\}$, *and* $\Delta' = \Delta_1 \cup \Delta_2 \cup \Delta_3$ *where*

$\Delta_1 = \{(q_0', nd, q_0') \mid nd \in \mathsf{Node}\}$

$\Delta_2 = \{(q_0', nd, (nd, p, s)) \mid nd \in \mathsf{Node} \wedge p \in \mathsf{PosOf}(nd) \wedge s \in \mathsf{Slope}\}$

$\Delta_3 = \{((nd, p, s), nd', (nd', p', s')) \mid (nd, nd') \in \mathsf{Edge} \wedge \mathsf{R}_{nd,nd'}(p, p', s')\}$

An accepting run of $\mathsf{TAut}_{\mathsf{Node}}(\mathsf{SG})$, via transitions in Δ_1, first consumes some arbitrary finite sequence of nodes. Then, via Δ_2, it starts tracing by choosing a position of the next node being consumed. Thereafter, via Δ_3, it traces positions along edges according to R, recording

the witnessed slope. Acceptance requires witnessing $\searrow$ infinitely often along the traced suffix. Hence $\mathsf{Lang}(\mathsf{TAut}_{\mathsf{Node}}(\mathsf{SG})) = \{w \cdot \pi \mid w \in \mathsf{Node}^*, \pi \in \mathsf{IPath}(\mathsf{SG}) \text{ and } \pi \text{ is descending}\}$.

As previously noted, Infinite Descent is equivalent to the language inclusion problem.

▶ **Theorem 6** (Vertex-language Criterion). SG *satisfies Infinite Descent if and only if* $\mathsf{Lang}(\mathsf{PAut}_{\mathsf{Node}}(\mathsf{SG})) \subseteq \mathsf{Lang}(\mathsf{TAut}_{\mathsf{Node}}(\mathsf{SG}))$.

Slope-language Criterion

Deciding the vertex-language criterion is expensive since the state space of the trace automaton, whose complement must be computed, scales with both nodes and positions. An alternative interpretation of ipaths shifts the alphabet from the graph's nodes to the sloped relations associated with the edges. That is, the Büchi automata of the slope-language construction recognise words over the alphabet $\Sigma = \{\, \mathsf{R}_{nd,nd'} \mid (nd, nd') \in \mathsf{Edge} \,\}$. This means we can define the trace automaton solely in terms of positions.

▶ **Definition 7** (Path automaton). *The automaton* $\mathsf{PAut}_{\mathsf{R}}(\mathsf{SG}) = (\Sigma, Q, q_0, \Delta, F)$ *consists of* $Q = \mathsf{Node} \cup \{q_0\}$ *where* $q_0 \notin \mathsf{Node}$, $\Delta = \{(q_0, \mathsf{R}_{nd,nd'}, nd') \mid (nd, nd') \in \mathsf{Edge}\} \cup \{(nd, \mathsf{R}_{nd,nd'}, nd') \mid (nd, nd') \in \mathsf{Edge}\}$, *and* $F = \mathsf{Node}$.

Thus $\mathsf{PAut}_{\mathsf{R}}(\mathsf{SG})$ recognizes exactly the sequences of sloped relations that arise along some ipath.

▶ **Definition 8** (Trace automaton). *The automaton* $\mathsf{TAut}_{\mathsf{R}}(\mathsf{SG}) = (\Sigma, Q', q_0', \Delta', F')$ *consists of* $Q' = \{q_0'\} \cup \{(p, s) \mid p \in P \wedge s \in \mathsf{Slope}\}$ *where* q_0' *is fresh and* $P = \bigcup_{nd \in \mathsf{Node}} \mathsf{PosOf}(nd)$, $F' = \{(p, \searrow) \mid p \in P\}$, *and* $\Delta' = \Delta_1 \cup \Delta_2 \cup \Delta_3$ *where*

$$\Delta_1 = \{(q_0', R, q_0') \mid R \in \Sigma\}$$

$$\Delta_2 = \{(q_0', R, (p', s')) \mid R \in \Sigma \wedge p' \in P \wedge s' \in \mathsf{Slope}\}$$

$$\Delta_3 = \{((p, s), R, (p', s')) \mid p \in P \wedge s \in \mathsf{Slope} \wedge R \in \Sigma \wedge R(p, p', s')\}$$

As before, Δ_1 allows the automaton to consume some arbitrary prefix, and Δ_2 starts tracing by guessing a position. Thereafter, Δ_3 updates the recorded position according to the next relation being consumed, whilst remembering which slope was traversed. Again, acceptance requires witnessing $\searrow$ infinitely often, and inclusion between the languages of the automata is equivalent to Infinite Descent.

▶ **Theorem 9** (Slope-language Criterion). SG *satisfies Infinite Descent if and only if* $\mathsf{Lang}(\mathsf{PAut}_{\mathsf{R}}(\mathsf{SG})) \subseteq \mathsf{Lang}(\mathsf{TAut}_{\mathsf{R}}(\mathsf{SG}))$.

3.1.2 Isabelle Mechanization

We mechanize the vertex-language and slope-language criteria inside the `Sloped_Graph` locale. We utilize Isabelle libraries for automata [7, 6] that provide a data type for representing Büchi automata, `('label, 'state) nba`, parameterized over the types used to represent the alphabet and the state space, respectively. Infinite words and runs are represented as streams, and acceptance is defined using LTL operators over these streams.

The vertex-language path and trace automata, as well as the slope-language path and trace automata, are all straightforwardly defined as values of the `nba` type, using data corresponding to the definitions above. We then prove direct translations of theorem 6 and theorem 9. In all four automata, the fresh initial state is represented uniformly via the `option` type: `None` serves as the fresh state and `Some` wraps the ordinary states.

A small but important technical point is that the vertex-language trace characterization is expressed in terms of streams whose letters lie in `Node`: tracing constraints apply only after the automaton leaves its initial "skip" mode, so without an explicit restriction the finite prefix of an input stream (of type `'node`) need not lie in the designated alphabet set. For the slope-language criterion, the alphabet is the finite set of edge-labelled sloped relations induced by R, and we connect relation streams to ipaths via the induced edge-relation stream mapping.

Our mechanization also resulted in small but useful extensions to Isabelle's existing infrastructure, which can be found in our `Buchi_Preliminaries` theory file. To keep the proofs for the automata-based criteria lightweight and reusable, we extended the LTL and Büchi complementation libraries by adding introduction and elimination rules tailored to runs, paths, and node-wise reasoning. In particular, we include a streamlined ω-lasso witness argument (i.e. language non-emptiness implies acceptance of some ω-lasso word) and a practical introduction rule for language inclusion that reduces the full language inclusion check $\mathsf{Lang}(A_1) \subseteq \mathsf{Lang}(A_2)$ to one that considers only ω-lasso words.

Lastly, we instantiated both the vertex-language and slope-language criteria to prove that Infinite Descent holds of the concrete example from section 2.3.

3.2 Relation-based Criterion

Infinite Descent also admits an equivalent *relation-based* characterization. This was first described in the context of size-change termination [20] and later generalized and refined [8]. The key idea is to summarize each finite path using a sloped relation, resulting in a finite abstraction that can be computed via a fixed-point computation. Infinite Descent can then be decided by checking a simple condition on the sloped relations that summarize loops. Sec. 3.2.1 recalls the construction and Sec. 3.2.2 describes its realization in Isabelle/HOL.

3.2.1 Underlying Mathematical Theory

For the relation-based view it is convenient to work with finite paths and cycles. Let $\mathsf{SG} = (\mathsf{Node}, \mathsf{Edge}, \mathsf{PosOf}, \mathsf{R})$ be a sloped graph. For a path $\pi = (nd_i)_{i<n}$ in SG, we define its *summary* to be the sloped relation $\mathsf{R}_\pi = \mathsf{R}_{nd_0,nd_1} \odot \mathsf{R}_{nd_1,nd_2} \odot \cdots \odot \mathsf{R}_{nd_{n-2},nd_{n-1}}$, where $\odot$ is composition of sloped-relations (defined below). Intuitively, $\mathsf{R}_\pi(p, p', s)$ holds when p can be propagated along π to p' and s records whether a strict decrease occurs anywhere on the chosen route. Given sloped relations P and Q, we define their *sloped composition* $P \odot Q$ by $(P \odot Q)(p, p'', s) \iff \exists p', s_1, s_2.\, P(p, p', s_1) \wedge Q(p', p'', s_2) \wedge s = \mathsf{max}(s_1, s_2)$. We define a (total) order $\rightsquigarrow < \searrow$ on slopes, and lift this to (a partial order on) relations by $P \leq Q$ iff $P(p, p', s) \Rightarrow \exists s'.\, (s \leq s') \wedge Q(p, p', s')$, and write $P < Q$ for the strict part. We write $P^{\oplus}$ for the least R such that $P \leq R$ and $P \odot R \leq R$ (the transitive closure of P).

To obtain the abstraction mentioned above, we maintain for each pair of nodes (nd, nd') a *thin* set of relations, meaning that it contains no two relations such that $P < Q$ (i.e., it is an anti-chain). Given a set L of sloped relations, its *downward-thinning*, defined as $\mathsf{Dt}(L) = \{P \in L \mid \neg\exists Q \in L.\, Q < P\}$, retains only the $\leq$-minimal sloped relations in L. We iteratively define a family $(\mathsf{Cl}^i_{nd,nd'})$ of indexed collections of sets of sloped relations as shown in fig. 2. We write Cl to denote the indexed collection of sets that is the first fixed point of this iteration. Intuitively, $\mathsf{Cl}_{nd,nd'}$ contains (up to thinning) exactly the compositions R_π corresponding to paths from nd to nd'.

Since the Infinite Descent property is determined by (infinitely) iterated cycles, the criterion need only inspect the sets $\mathsf{Cl}_{nd,nd}$ (i.e. the diagonal of Cl, viewed as a matrix):

$$\mathsf{Cl}^0_{nd,nd'} = \begin{cases} \{\, \mathsf{R}_{nd,nd'} \,\} & \text{if } (nd, nd') \in \mathsf{Edge}, \\ \varnothing & \text{otherwise}, \end{cases}$$

$$\mathsf{Cl}^{i+1}_{nd,nd''} = \mathsf{Dt}\Big(\mathsf{Cl}^i_{nd,nd''} \;\cup \bigcup_{nd' \in \mathsf{Node}} \{\, P \odot Q \mid P \in \mathsf{Cl}^i_{nd,nd'} \wedge Q \in \mathsf{Cl}^i_{nd',nd''} \,\}\Big)$$

Figure 2 Iterative definition of the relation-based abstraction.

the relations in these sets summarize the behaviour of cycles, and $P^{\oplus}(p, p, \searrow)$ expresses a circularly decreasing trace through some iteration of P.

▶ **Definition 10** (Transitive looping). *A sloped graph* SG *satisfies the* Transitive Looping *property if for every* $nd \in \mathsf{Node}$ *and* $P \in \mathsf{Cl}_{nd,nd}$ *there exists* $p \in \mathsf{PosOf}(nd)$ *such that* $P^{\oplus}(p, p, \searrow)$.

▶ **Theorem 11** (Relation-based criterion [8]). *A sloped graph* SG *satisfies Infinite Descent if and only if it satisfies Transitive Looping.*

3.2.2 Isabelle Mechanization

This was the most involved theory in our development (~1.3k LOC). The mechanization of the closure Cl follows the construction described above, using sloped composition and downward thinning. An important technical point about our mechanization is that we cannot overload the '$\leq$' notation to denote the lifting of the ordering on slopes to sloped relations, since this is already being used by Isabelle (via the `Orderings` type class) for the ordering on sloped relations induced by subset inclusion. We therefore define an explicit predicate `leqSl`, capturing the lifted ordering, that we then must prove to be a partial order (i.e., reflexive, transitive and anti-symmetric).

The main proof-engineering issue is that downward thinning is a *non*-monotonic operation. Accordingly, the development proceeds via an auxiliary (over-approximating) closure abstraction `Ccl`, defined by a monotone operation, that supports the standard inductions needed to relate closure elements to path concatenation. We connect the two closures using a predicate `initFrag` on collections of sets of sloped relations, indexed by pairs of nodes in the sloped graph. Specifically, `initFrag X Y` asserts that, for each pair (nd, nd') of nodes, every relation in the closure $\mathtt{Y}_{nd,nd'}$ is bounded from below (w.r.t. `leqSl`) by some relation in $\mathtt{X}_{nd,nd'}$. We then prove that both `initFrag Ccl` Cl and `initFrag` Cl `Ccl` hold, so `Ccl` and Cl induce the same minimal sloped relations. We define analogues of Transitive Looping (theorem 10) with respect to both Cl and `Ccl`, and then prove these equivalent.

The final correctness proof is organized around giving a correspondence between the sloped relations in the `Ccl` abstraction and the summaries R_π of finite paths π in the sloped graph. For this we define a predicate, `tracksPers R` π, expressing that a sloped relation `R` is the summary of the path π. We then show that `tracksPers` is left- and right-total for `Ccl`: $\mathsf{R}_\pi \in$ `Ccl` for every path π in the sloped graph, and every sloped relation `Ccl` is the summary of some path. Next we prove that, for a sloped relation summmarising a path, there is a descending loop in its transitive closure if and only if the corresponding ω-cycle in the sloped graph is infinitely descending. This yields that Transitive Looping for `Ccl` holds iff so too does Infinite Descent, and thence we obtain the analogue of theorem 11 by the above-mentioned equivalence for the two versions of Transitive Looping.

(a) A sloped graph failing to satisfy SD. **(b)** A sloped graph failing to satisfy XSD.

Figure 3 Sloped graphs demonstrating counterexamples for the SD and XSD criteria, whose nodes are natural numbers i and whose positions $p, p' \in \mathsf{PosOf}(i)$ are indexed by i.

4 Incomplete Criteria for Infinite Descent

This section considers some sufficient criteria for deciding Infinite Descent that are incomplete, but useful in practice. Section 4.1 recalls a criterion (SD) due to Sprenger and Dam [27], which we adapt to the general setting of sloped graphs, and then presents a novel theoretical contribution: an extension that strictly generalizes SD, which we call XSD. Section 4.2 then considers two lightweight cycle-based heuristics used in Cyclone.

4.1 The SD and XSD criteria

4.1.1 Underlying Mathematical Theory

SD Criterion

Sprenger and Dam's SD criterion is an intuitive condition that is sufficient to guarantee Infinite Descent. Essentially it consists in identifying, for each strongly connected subgraph, a single connected trace that witnesses descent at some point. Sprenger and Dam's formulation of this condition was specific to a particular logical inference system (for a variant of first-order μ-calculus), and also requires the graph to be in so-called *cycle normal form* (i.e., one that can be decomposed into a tree with 'back-pointers'). A generic reformulation is given by Brotherston [4, §7.2], who calls it the *trace manifold* condition, but this is again predicated on cycle normal form. Conversely, the formulation of the criterion that we now give is applicable to *arbitrary* sloped graphs.

▶ **Definition 12** (SD Criterion). *Let* $G = (\mathsf{Node}', \mathsf{Edge}')$ *be a subgraph of a sloped graph* $\mathsf{SG} = (\mathsf{Node}, \mathsf{Edge}, \mathsf{PosOf}, \mathsf{R})$. *A* position choice *for* G *is a family* $(p_{nd})_{nd \in \mathsf{Node}'}$ *of positions such that:*

- $\forall nd \in \mathsf{Node}'.\ p_{nd} \in \mathsf{PosOf}(nd)$,
- $\forall (nd, nd') \in \mathsf{Edge}'.\ \exists s \in \mathsf{Slope}.\ \mathsf{R}_{nd,nd'}(p_{nd}, p_{nd'}, s)$.

We say that it is decreasing *if* $\exists (nd, nd') \in \mathsf{Edge}'.\ \mathsf{R}_{nd,nd'}(p_{nd}, p_{nd'}, \searrow)$. *We call* G descending *if it admits a decreasing position choice. We call* SG SD-descending *if every strongly connected subgraph of* SG *is descending.*

The SD criterion guarantees Infinite Descent since the edges traversed infinitely often by an ipath induces a strongly connected subgraph (the 'limit' of the ipath), which is then guaranteed to have a descreasing trace by the criterion.

▶ **Proposition 13.** *A sloped graph* SG *satisfies Infinite Descent if it is SD-descending.*

This criterion is not complete in general: some graphs satisfy Infinite Descent but require visiting multiple positions per node.

▶ **Example 14.** *The sloped graph in fig. 3a, consisting of a single strongly connected subgraph, satisfies Infinite Descent. However, any descending trace must alternate between the two positions, p_1 and p_1', and so there is no choice of a single position that witnesses descent in the strongly connected subgraph. Consequently the graph is not SD-descending.*

XSD Criterion

The source of incompleteness highlighted in theorem 14 for the SD criterion suggests a way of extending it in order to capture more sloped graphs satisfying Infinite Descent. Namely, rather than choosing a single position for each node of a strongly connected subgraph, we instead capture a *set* of positions for each node. Additionally, we provide a function witnessing how these positions can be connected (via the sloped relations along edges) to build a descending trace for each ipath inducing that subgraph. This results in our novel *extended* Sprenger-Dam (XSD) criterion.

In the following, we fix a sloped graph $\mathsf{SG} = (\mathsf{Node}, \mathsf{Edge}, \mathsf{PosOf}, \mathsf{R})$.

▶ **Definition 15** (Position-change Set-choice). *Let $G = (\mathsf{Node}', \mathsf{Edge}')$ be a subgraph of SG. A* position-change set-choice *for G is a family $(P_{nd})_{nd \in \mathsf{Node}'}$ of sets of positions, together with a family of functions $(f_{nd,nd'} : P_{nd} \to P_{nd'})_{(nd,nd') \in \mathsf{Edge}'}$ such that the following hold:*

- $\forall nd \in \mathsf{Node}'.\ \varnothing \subsetneq P_{nd} \subseteq \mathsf{PosOf}(nd)$,
- $\forall (nd, nd') \in \mathsf{Edge}'.\ \forall p \in P_{nd}.\ \exists s \in \mathsf{Slope}.\ \mathsf{R}_{nd,nd'}(p, f_{nd,nd'}(p), s)$.

To express our condition guaranteeing that these set-choices can be used to construct decreasing traces, we relate them to certain subgraphs in a refinement of the sloped graph.

▶ **Definition 16** (Slices). *We define the* extension *of SG to be the (directed) graph $\mathsf{Ext}(\mathsf{SG})$ having nodes (nd, p) for $nd \in \mathsf{Node}$ and $p \in \mathsf{PosOf}(nd)$, and edges $((nd, p), (nd', p'))$ for each $(nd, nd') \in \mathsf{Edge}$ and p, p' such that $\mathsf{R}_{nd,nd'}(p, p', s)$ for some slope s. Take a position-change set-choice $\Pi = ((P_{nd})_{nd \in \mathsf{Node}'}, (f_{nd,nd'})_{(nd,nd') \in \mathsf{Edge}'})$ for some subgraph of SG. We call a subgraph $(\mathcal{N}, \mathcal{E})$ of $\mathsf{Ext}(\mathsf{SG})$ a* slice *of Π when the following hold.*

- *For all $((nd, p), (nd', p')) \in \mathcal{E}$, we have $(nd, nd') \in \mathsf{Edge}'$ and $f_{nd,nd'}(p) = p'$.*
- *For all $(nd, nd') \in \mathsf{Edge}'$, there is $((nd, p), (nd', p')) \in \mathcal{E}$ for some p and p'.*

We say that the slice is decreasing *if $\mathsf{R}_{nd,nd'}(p, p', \searrow)$ for some edge $((nd, p), (nd', p')) \in \mathcal{E}$, and that Π is* descending *when every strongly connected slice of Π is decreasing.*

We now state our extension of the SD criterion.

▶ **Definition 17** (XSD Criterion). *SG is called XSD-descending if every strongly connected subgraph of SG has a descending position-change set-choice.*

To see that the XSD criterion is sufficient to guarantee Infinite Descent, consider an ipath in the sloped graph. The criterion guarantees, for its limit, a descending position-change set-choice $((P_{nd})_{nd \in \mathsf{Node}'}, (f_{nd,nd'})_{(nd,nd') \in \mathsf{Edge}'})$. For any ipath $nd_0, nd_1, \ldots$ inducing this limit (i.e., visiting each edge infinitely often), we can then construct a trace for any tail $nd_k, nd_{k+1}, \ldots$ consisting only of nodes in the limit by picking some position $p \in P_{nd_k}$, and following the functions $f_{nd_i,nd_{i+1}}$ (for each $i \geq k$) in the set-choice. Moreover, this trace corresponds to an ipath $(nd_k, p), (nd_{k+1}, f_{nd_k,nd_{k+1}}(p)), (nd_{k+2}, f_{nd_{k+1},nd_{k+2}}(f_{nd_k,nd_{k+1}}(p))), \ldots$ visiting each edge of some strongly connected slice infinitely often. Thus, since every such slice is decreasing, so too is the trace.

▶ **Proposition 18.** *If* SG *is XSD-descending then it satisfies Infinite Descent.*

It is not too difficult to see that the SD criterion is simply a special instance of this generalisation, in which each position-change set-choice consists of a singleton set of positions.

The XSD criterion captures the graph in fig. 3a, by taking $P_1 = \{p_1, p_1'\}$, $f_{1,1}(p_1) = p_1'$, and $f_{1,1}(p_1') = p_1$ (note that there is only one strongly connected subgraph). However, it is still incomplete for Infinite Descent.

▶ **Example 19.** *The sloped graph in fig. 3b satisfies Infinite Descent but is not XSD-descending. For Infinite Descent, note that it only has three types of ipath: for those ending in* $1, 2, 1, 2, \ldots$*, we have a trace* $p_1, p_2, p_1, p_2, \ldots$*; for those ending in* $1, 3, 1, 3, \ldots$*, we have a trace* $p_1', p_3, p_1', p_3, \ldots$*; and, lastly, we have a trace* $(p_1, p_2)^{m_1}(p_1', p_3)^{n_1}(p_1, p_2)^{m_2}(p_1', p_3)^{n_2} \ldots$ *for those alternating infinitely between the two cycles. However, there is no set-choice for the strongly connected subgraph consisting of the whole graph: no values for either* $f_{1,3}(p_1)$ *or* $f_{1,2}(p_1')$*, and thus no (non-empty) sets* P_1*, satisfy the requirements of the criterion.*

It seems possible that our extension of the Sprenger-Dam criterion is related to the condition that underlies the recently developed *stack-controlled* style of cyclic proofs [29]. The latter extends an alternative characterisation of the Sprenger-Dam criterion based on the notion of *induction orders*, which are certain partial orders on the basic cycles. Our XSD condition may be the corresponding Sprenger-Dam style counterpart to these extended induction orders. Again, we note that stack-controlled proofs and induction orders are predicated on graphs in cycle normal form, but our XSD condition is expressed in terms of arbitrary sloped graphs.

4.1.2 Isabelle Mechanization

We mechanized the SD and XSD criteria inside the `Sloped_Graph` locale, reusing standard directed-graph infrastructure from our `Directed_Graphs` theory including subgraphs (`subgr`), strongly connected graphs (`scg`), and strongly connected subgraphs (`scsg`).

The SD Criterion

The SD criterion quantifies over strongly connected subgraphs. For each, it requires a *single* position choice per node, with these positions being connected (at least once via $\searrow$) by the sloped relations along each edge. In Isabelle, we package this data into a predicate `decreasingPCC`. A deliberate design choice is to specify the strongly connected subgraph purely in terms of the edge relation, since additionally specifying the set of nodes is redundant. This allows the same lemmas to be reused uniformly for each strongly connected subgraph. The criterion itself, mechanized as the predicate `SDdescending`, quantifies over strongly connected subgraphs and requires the `decreasingPCC` predicate to hold of them. We prove the soundness theorem (cf. theorem 13) as follows: starting from an arbitrary ipath, we pass to its limit using the directed-graph metatheory, obtain the corresponding position choice witnessing the `decreasingPCC` predicate, and then invoke a witness-to-trace lemma to obtain a decreasing trace on a suffix. We also mechanized theorem 14, showing incompleteness of the criterion.

The XSD Criterion

The XSD criterion was mechanised as the following predicate.

```
definition XSDdescending :: bool where
"XSDdescending ≡ ∀Node1 edge1. scsg Node1 edge1 ⟶
    (∃lab f. wfLabFS Node1 lab ∧ descending_PCSC_sliced Node1 edge1 lab f)"
```

This ensures that each strongly connected subgraph (`scsg Node1 edge1`) of the sloped graph has a descending position-change set-choice, using the `descending_PCSC_sliced` predicate: the parameter `lab` denotes the set of positions chosen for each node, and the parameter `f` denotes the family of functions $(f_{nd,nd'})_{(nd,nd')\in\texttt{edge1}}$, cf. theorem 15. This uses the auxiliary predicate `RRSetChoice`, expressing the conditions of theorem 15, and additionally encodes the conditions of theorem 16 using auxiliary predicates `is_slice` and `decreasing_slice`. These predicates, in turn, depend on defintions of the nodes (`ExtG_Nodes`) and edges (`ExtG_Edges`) of the extension of the sloped graph.

The proof of the XSD soundness theorem (cf. theorem 18) is substantially more involved than that for SD. Specifically, an explicit recursive construction (using `rec_nat`) is required in order to build a trace for a given an ipath using the functions `f` from the appropriate position-change set-choice. Relating this to a strongly connected slice then proceeds, via a pigeonhole-style repetition argument, from an appropriate index-based decomposition of the trace into a sequence of finite segments. Compared with the soundness proof of the SD criterion, which made elegant use of high-level LTL operators, the arguments needed for the XSD criterion required explicit arithmetic reasoning about indices, bounds, and segment lengths. We also mechanized theorem 19, showing incompleteness of the criterion.

4.2 Cycle Based Criteria

We next recall two criteria used by the Cyclone verifier [10]: a sufficient condition for *failure* of Infinite Descent (flat cycles), and a complete characterization for the special case of *unicycles* graphs.

4.2.1 Underlying Mathematical Theory

Flat Cycles Criterion

A graph fails Infinite Descent if it admits an infinite path with no forced strict decreases. This is witnessed by a *flat* cycle: one whose edges never relate any positions via a $\searrow$ step. Let $\mathsf{FlatEdge} = \{(nd, nd') \in \mathsf{Edge} \mid \forall p \in \mathsf{PosOf}(nd), q \in \mathsf{PosOf}(nd'). \neg \mathsf{R}_{nd,nd'}(p, q, \searrow)\}$ denote the set of all such edges in the sloped graph. Then a cycle is *flat* if all its edges lie in $\mathsf{FlatEdge}$.

▶ **Theorem 20** (Flat Cycles Criterion [10]). *If a sloped graph has a flat cycle, then it does not satisfy Infinite Descent.*

Descending Unicycles Criterion

The descending unicycles criterion applies when the sloped graph is a *unicycles graph*, meaning that is has no overlapping cycles (i.e., each strongly connected component consists of a single basic cycle).

▶ **Definition 21** (Unicycles Graph). *A directed graph is a* unicycles graph *if for any two basic cycles c, c', whenever c reaches c' and c' reaches c, then they contain the same set of nodes.*

The Descending Unicyles criterion amounts to checking that there is a descending trace for each simple cycle. To express this, we reuse the notion of the extension of the sloped graph, from theorem 16.

▶ **Definition 22** (Simply descending). *Let $c = (nd_0, \ldots, nd_k)$ (with $nd_0 = nd_k$) be a basic cycle in a sloped graph* SG*, and consider the restriction,* $\mathsf{Ext}_c(\mathsf{SG})$*, of the extension of* SG *to the cycle c: the subgraph of* $\mathsf{Ext}(\mathsf{SG})$ *consisting of just the edges of the form $((nd, p), (nd', p'))$ such that $nd = nd_i$ and $nd' = nd_{i+1}$ for some $0 \leq i < k$. We say that the cycle c is* descending *if there exists a basic cycle in* $\mathsf{Ext}_c(\mathsf{SG})$ *containing an edge $((nd_i, p), (nd_{i+1}, p'))$ such that $\mathsf{R}_{nd_i, nd_{i+1}}(p, p', \searrow)$. We call* SG simply descending *if every basic cycle in* SG *is descending.*

Interestingly, as well as being a sufficient criterion, unicycles graphs satisfying Infinite Descent are also necessarily simply descending.

▶ **Theorem 23** (Descending Unicycles Criterion [10]). *A unicycles sloped graph satisfies Infinite Descent iff it is Simply Descending.*

4.2.2 Isabelle Mechanization

The mechanization of the cycle-based criteria are structured so that each criterion is exposed through a small predicate-level interface along with a short list of reusable elimination or introduction rules. This was done so that later proofs can invoke the criterion without replaying the construction details.

Flat Cycles Criterion

We formalize the Flat Cycles criterion as a predicate `FlatCycle`, which captures the notion of a graph containing a directed cycle consisting entirely of edges in `FlatEdges`, the set of flat edges in the sloped graph. The soundness theorem follows the a step structure: from a witness of the `FlatCycle` predicate we construct an ω-word that is a valid ipath, and then show that every candidate trace along that ipath is forced to be nondecreasing. We package the resulting contradiction as an elimination rule that is then used directly to conclude $\neg$`InfiniteDescent` for any sloped graph exhibiting a flat cycle.

We instantiate an example from Cyclone [10, fig. 4b] to confirm that our mechanization can indeed validate failure of Infinite Descent from concrete witnesses.

Descending Unicycles Criterion

We first developed relevant infrastructure in the `Directed_Graph` context, defining the notion of a basic cycle as a cycle with no intermediate repeated nodes. We then used this to state the unicycles graph property, as well as a lasso decomposition lemma (`unicycle_lasso`): every ipath in a unicycles graph eventually repeats a basic cycle. The proof constructs the limit of the ipath and, using the fact that this forms a strongly connected subgraph, exploits the unicycles property to obtain a unique successor property for paths inside the limit, forcing deterministic repetition of a single basic cycle.

Within the `Sloped_Graph` context, rather than directly constructing a representation of the cycle-restricted extension (cf. theorem 22), we opt for a more implicit approach. We first define an explicit representation (`descentPath`) of descending traces along *finite* path segments. We then define a descending cycle (`cycleDescends`) to be one that can be repeated (some finite number of times) such that the resulting path segment has a descending trace starting and ending with the same position. The predicate `SimplyDescendingGraph` then requires all basic cycles in a sloped graph to witness `cycleDescends`. This choice keeps the criterion compatible with the rest of our list and stream infrastructure.

The equivalence proof (cf. theorem 23) proves the bi-implication in the following way. For `InfiniteDescent` implies `SimplyDescendingGraph`, we instantiate `InfiniteDescent` on c^ω for each basic cycle c and then extract a witness for `cycleDescends` using finiteness of positions. For `SimplyDescendingGraph` implies `InfiniteDescent`, we infer, using `unicycle_lasso`, that ipaths are ultimately periodic over a basic cycle, and then derive a descent on each successive finite segment from the fact that (by assumption) the cycle witnesses `cycleDescends`.

We instantiate positive and negative examples (cf. [10, fig. 6]) to confirm that our mechanization can indeed validate (failure of) Infinite Descent from concrete witness of the Descending Unicycles criterion.

5 Conclusion

We presented an Isabelle/HOL mechanization of *Infinite Descent* in a locale-based framework of sloped graphs. Within this setting, we verified soundness of the incomplete criteria used in practice (SD, our extension XSD, and Cyclone's cycle heuristics), and mechanized two complete characterizations: an automata-based reduction to Büchi language inclusion (cf. the vertex-language and slope-language criteria) and a relation-based algebraic criterion. All results are fully machine-checked and replayable from the accompanying artifact.

A primary motivation for this formalization was to bridge the gap between automated discovery and trusted validation. Consequently, an immediate line of future work is to instrument automated provers, such as Cyclist, to export cyclic proof objects that can be reconstructed and checked within our Isabelle/HOL locale. Combined with the methods already implemented, this could lead to a Sledgehammer-like tool [23] for checking Infinite Descent. Furthermore, we intend to investigate the definition of a new object logic within Isabelle that treats Infinite Descent as a first-class inference rule. By embedding our verified criteria into the kernel of such a logic, we can support cyclic tactics natively, relying on the certified automata-theoretic or relational checks to discharge the associated soundness obligations automatically.

References

1 Linear temporal logic on streams. part of the isabelle library. `https://isabelle.in.tum.de/dist/library/HOL/HOL-Library/Linear_Temporal_Logic_on_Streams.html`, 2022.

2 Andreas Abel and Thorsten Altenkirch. A Predicative Analysis of Structural Recursion. *Journal of Functional Programming*, 12(1):1–41, 2002. `doi:10.1017/S0956796801004191`.

3 Clemens Ballarin. Locales: A Module System for Mathematical Theories. *J. Autom. Reason.*, 52(2):123–153, 2014. `doi:10.1007/s10817-013-9284-7`.

4 James Brotherston. *Sequent Calculus Proof Systems for Inductive Definitions*. PhD thesis, University of Edinburgh, November 2006. URL: `https://era.ed.ac.uk/handle/1842/1458`.

5 James Brotherston and Alex Simpson. Sequent Calculi for Induction and Infinite Descent. *J. Log. Comput.*, 21(6):1177–1216, 2011. `doi:10.1093/LOGCOM/EXQ052`.

6 Julian Brunner. Büchi Complementation. *Archive of Formal Proofs*, October 2017. `https://isa-afp.org/entries/Buchi_Complementation.html`, Formal proof development.

7 Julian Brunner. Transition Systems and Automata. *Archive of Formal Proofs*, October 2017. `https://isa-afp.org/entries/Transition_Systems_and_Automata.html`, Formal proof development.

8 Liron Cohen, Adham Jabarin, Andrei Popescu, and Reuben N. S. Rowe. The Complex(ity) Landscape of Checking Infinite Descent. *Proc. ACM Program. Lang.*, 8(POPL), jan 2024. doi:10.1145/3632888.

9 Liron Cohen and Reuben N. S. Rowe. Non-Well-Founded Proof Theory of Transitive Closure Logic. *ACM Trans. Comput. Logic*, 21(4), August 2020. doi:10.1145/3404889.

10 Liron Cohen, Reuben N. S. Rowe, and Matan Shaked. Cyclone: A Heterogeneous Tool for Verifying Infinite Descent. In Arie Gurfinkel and Marijn Heule, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 336–354, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-90643-5_18.

11 Anupam Das. On The Logical Complexity of Cyclic Arithmetic. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:1)2020.

12 Christian Dax, Martin Hofmann, and Martin Lange. A Proof System for the Linear Time μ-Calculus. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006. doi:10.1007/11944836_26.

13 Agda Developers. Agda. URL: https://agda.readthedocs.io/.

14 Amina Doumane. Constructive Completeness for the Linear-time μ-calculus. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005075.

15 The Idris Community. *The Idris Reference*, jan 2016. Release 0.9.19.1. URL: https://docs.idris-lang.org/_/downloads/en/v0.10/pdf/.

16 Eddie Jones, C.-H. Luke Ong, and Steven Ramsay. Cycleq: An Efficient Basis for Cyclic Equational Reasoning. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2022, pages 395–409, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519939.3523731.

17 Florian Kammüller, Markus Wenzel, and Lawrence C. Paulson. Locales - A Sectioning Concept for Isabelle. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin-Mohring, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings*, volume 1690 of *Lecture Notes in Computer Science*, pages 149–166. Springer, 1999. doi:10.1007/3-540-48256-3_11.

18 Alexander" Krauss. Certified Size-Change Termination. In Frank" Pfenning, editor, *Automated Deduction – CADE-21"*, pages 460–475", Berlin, Heidelberg", 2007". Springer Berlin Heidelberg". doi:10.1007/978-3-540-73595-3_34.

19 Alexander Krauss, Christian Sternagel, René Thiemann, Carsten Fuhs, and Jürgen Giesl. Termination of Isabelle functions via Termination of Rewriting. In Marko van Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk, editors, *"Interactive Theorem Proving*, pages 152–167, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-22863-6_13.

20 Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The Size-change Principle for Program Termination. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 81–92. ACM, 2001. doi:10.1145/360204.360210.

21 Rodolphe Lepigre and Christophe Raffalli. Practical Subtyping for Curry-Style Languages. *ACM Trans. Program. Lang. Syst.*, 41(1):5:1–5:58, 2019. doi:10.1145/3285955.

22 Cesar A. Muñoz, Mauricio Ayala-Rincón, Mariano M. Moscato, Aaron M. Dutle, Anthony J. Narkawicz, Ariane Alves Almeida, Andréia B. Avelar da Silva, and Thiago M. Ferreira Ramos. Formal Verification of Termination Criteria for First-Order Recursive Functions. *Journal of Automated Reasoning*, 67(4):40, 2023. doi:10.1007/s10817-023-09669-z.

23 Lawrence C. Paulson and Jasmin Christian Blanchette. Three Years of Experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *The 8th International Workshop on*

the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011, volume 2 of *EPiC Series in Computing*, pages 1–11. EasyChair, 2010. doi:10.29007/36dt.

24 Luigi Santocanale. A Calculus of Circular Proofs and Its Categorical Semantics. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2002. doi:10.1007/3-540-45931-6_25.

25 Alex Simpson. Cyclic Arithmetic Is Equivalent to Peano Arithmetic. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 283–300, 2017. doi:10.1007/978-3-662-54458-7_17.

26 Christoph Sprenger and Mads Dam. A Note on Global Induction in a mu-calculus with Explicit Approximations. In Zoltán Ésik and Anna Ingólfsdóttir, editors, *Fixed Points in Computer Science, FICS 2002, Copenhagen, Denmark, 20-21 July 2002, Preliminary Proceedings*, volume NS-02-2 of *BRICS Notes Series*, pages 22–24. University of Aarhus, 2002. URL: https://www.brics.dk/NS/02/2.

27 Christoph Sprenger and Mads Dam. On the Structure of Inductive Reasoning: Circular and Tree-Shaped Proofs in the μ-calculus. In Andrew D. Gordon, editor, *Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS 2003 Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2620 of *Lecture Notes in Computer Science*, pages 425–440. Springer, 2003. doi:10.1007/3-540-36576-1_27.

28 René Thiemann and Jürgen Giesl. The Size-Change Principle and Dependency Pairs for Termination of Term Rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 16(4):229–270, September 2005. doi:10.1007/s00200-005-0179-7.

29 Dominik Wehr. *Cyclic Proof Theory*. Doctoral thesis, University of Gothenburg, Gothenburg, Sweden, December 2025. URL: https://hdl.handle.net/2077/89733.

30 Jamie Wright, Andrei Popescu, Reuben N. S. Rowe, and Liron Cohen. Certified Infinite Descent Criteria in Isabelle/HOL – ITP 2026 Software Artifact, 2026. doi:10.5281/zenodo.20323854.

31 Jamie Wright, Andrei Popescu, Reuben N. S. Rowe, and Liron Cohen. Certified Infinite Descent Criteria in Isabelle/HOL (Browseable HTML), 2026. URL: https://jamierwright.github.io/InfiniteDescent/.