2021

Formal Reasoning for Analyzing Goal Models that Evolve over Time

Alicia M. Grubb  
*Smith College, amgrubb@smith.edu*

Marsha Chechik  
*University of Toronto*

Follow this and additional works at: https://scholarworks.smith.edu/csc_facpubs

Part of the Computer Sciences Commons

**Recommended Citation**

https://scholarworks.smith.edu/csc_facpubs/332

This Article has been accepted for inclusion in Computer Science: Faculty Publications by an authorized administrator of Smith ScholarWorks. For more information, please contact scholarworks@smith.edu
Formal reasoning for analyzing goal models that evolve over time

Alicia M. Grubb1 · Marsha Chechik2

Received: 9 January 2020 / Accepted: 10 March 2021 / Published online: 10 May 2021 © The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2021

Abstract
In early-phase requirements engineering, modeling stakeholder goals and intentions helps stakeholders understand the problem context and evaluate tradeoffs, by exploring possible “what if” questions. Prior research allows modelers to make evaluation assignments to desired goals and generate possible selections for task and dependency alternatives, but this treats models as static snapshots, where the evaluation of the fulfillment of an intention remains constant once it has been determined. Using these techniques, stakeholders are unable to reason about possible evolutions, leaving questions about project viability unanswered when the fulfillment of goals or availability of components is not guaranteed in the future. In this article, we formalize the Evolving Intentions framework for specifying, modeling, and reasoning about goals that change over time. Using the Tropos language, we specify a set of functions that define how intentions and relationships evolve, and use path-based analysis for asking a variety of “what if” questions about such changes. We illustrate the framework using the Bike Lanes example and prove correctness of the analysis. Finally, we demonstrate scalability and effectiveness, enabling stakeholders to explore model evolution.

Keywords GORE · Tropos · Goal evolution

1 Introduction
Goal-oriented requirements engineering (GORE) aims to capture and reason about stakeholder and system goals as part of a requirements engineering process. In the early-phases of a software project, GORE elicits the intentions and social needs of stakeholders and connects them with the technical and non-functional requirements in the software system. Over the years, many approaches have been developed and extended (e.g., NFR [10], Tropos [18], i* (iStar) [52], GRL [3], and KAOS [38]). While each approach has a slightly different focus resulting in different support for analysis (see [32] for a comparison), all of the approaches have support for making tradeoff decisions between alternatives in projects.

Planning for and dealing with changes is important in every part of the software lifecycle [43, 50], especially in early-phase requirements when stakeholders are understanding the problem space prior to development [35, 37]. This is especially important for safety critical systems [30].

In this article, we consider early-phase modeling and reasoning about anticipated changes with GORE, where stakeholders can identify known–unknowns about the future evolution of their project and domain. We consider anticipated changes in whether each intention in the model will become satisfied or denied in the future, and how the relationship between these intentions may change over time. For example, requirements engineers may consider which features to implement based on changing user expectations about security, privacy, or energy usage. They may also prioritize the interests of some stakeholders over others, given their likelihood of being long-term customers or users of the system. In considering anticipated changes, analysts may consider potential changes in regulations or the stability of the underlying software ecosystem or platform on which they develop their application.

When planning for changing project scenarios, stakeholders can, in principle, reason about time in an ad hoc manner, by considering static snapshots of the model before or after the change, but this is insufficient to understand and make decisions about the impacts of these changes in the interim.
In our prior work, we argued that without an explicit notion of time, goal model analysis may lead to incorrect results (i.e., recommending an inferior tradeoff) when the fulfillment of task and dependency of resources is not known in the future [23]. We proposed and evaluated the notion of evolving intention evaluations in iStar [20, 23, 24]. While this work defined the problem and demonstrated feasibility of the approach, it suffered from the fact that iStar did not have formal semantics. As the result, the simulation did not have a unique interpretation (i.e., traceability), leading to confusion among modelers and stakeholders [23]. For example, when a contribution link is resolved with forward analysis, it is not always known which source intention labels caused the destination intention label. The same issue exists with backward analysis.

To allow stakeholders to consider alternatives over time with unique interpretations of solutions, we suggest exploring tradeoffs formally. In this article, we extend our prior work by formally defining how to specify this evolution in Tropos goal graphs [17, 49] for the purpose of automated analysis.

In the absence of well-documented software engineering examples with sufficient documented evolution, in this article, we use two city planning programs as our examples. The first one, the illustrative example, was going through the planning stage when our research was being conducted, allowing us first hand access to the stakeholders, and the second was a historic project with multiple documented revisions. We thus believe that from the early RE perspective, these projects are representative of the modeling and analysis challenges that we aim to support, even though they are not from the domain of software and systems requirements.

1.1 Overview of illustrative example: bike lanes example (BLE)

Consider the bike lanes example (BLE), initially introduced in [21]. Urban roads are often overcrowded leading to collisions between motorists and cyclists. Proponents of cycling argue for the addition of separated cycling lanes (called bike lanes). City of Toronto planners were given a mandate to add bike lanes to a major residential and commercial thoroughfare in the downtown core of the city. In consultation with area residents and businesses, the planners determined that the primary issues of contention are how the bike lanes will interact with parking and road maintenance, as well as whether they should be built in a way that makes them adaptable to seasonal usage (i.e., temporary construction with elements that are removable during the cold Canadian winter to accommodate snow clearing). The planners want to understand how these issues impact the safety of cyclists and public support, as well as the city’s ability to justify the cost of the project.

To describe the trade-offs under consideration by the city planners, we create a goal graph in Tropos [9], shown in Fig. 11. The city planners want to satisfy Have Bike Lanes, while satisfying their root-level goals (e.g., Cyclist Safety, Cost Justification, and Public Support). City planners are considering several trade-offs in the implementation of the bike lanes, which are modeled using or-decomposition links. Specifically, they are considering options for parking and whether the bike lanes should be installed curbside or have parking curbside. Another trade-off they are investigating is whether to invest in Permanent Construction of the lanes (i.e., involving road work) or Temporary Construction (i.e., re-painting lines). The city is considering the impact of these changes on road users.

GORE considers models to be open-world, meaning Fig. 1 is considered incomplete and there may be other ways of decomposing intentions in the model. For example, modelers omitted goals related to road congestion from Fig. 1, because the bike lane project was not believed to either improve or worsen road congestion in this scenario. This is an implicit assumption in the model. Another assumption is that the city planners have already decided that the no parking design option is not feasible because of its negative effect on Access to Parking and Public Support. We include this option for completeness but do not consider it in further trade-off analysis.

Several elements in this model change over time. For example, Have Design becomes satisfied at some point, but Bike Lane Usage may vary depending on external factors, such as seasonal changes in weather. Permanent Construction affects Access to Parking, while the roadway is being reconfigured, whereas Temporary Construction does not have the same effect. In this approach, we consider trade-off decisions in the context of these changes.

Modeler questions The city planners want to understand how changes in their goals over time impact their decisions by exploring the following questions:

EQ1 Can the City delay the design and planning decisions to the future (e.g., choosing between the alternatives for Have Design)?

EQ2 What scenario ensures that Access to Parking is satisfied in six months, even if off-street lots become unavailable?

EQ3 Is a permanent solution (i.e., Permanent Construction) or a temporary solution (i.e., Temporary Construction) most appropriate, given possible changes to cyclist and motor vehicle traffic?

1 We ignore distinctions between intention types and exclude actor boundaries, as the focus of this article is analysis.
EQ4 Are bike lanes or parking most effective on curbside, given seasonal constraints on road operations?

EQ5 How do variations in the satisfaction of Bike Lane Usage over time affect the city’s goals (e.g., Have Bike Lanes and Cyclist Safety)?

EQ6 Can the city eventually satisfy and maintain Cyclist Safety?

**Approaches** Standard forward and backward analysis [33] can be used to explore EQ3 and EQ4, but may not result in the best decision if changing intentions are not considered. For example, we can use forward analysis to answer EQ4 (and similarly EQ3) by creating an initial model before anything is built and additional models for each combination of Bike Lane Curbside and Parking Curbside, but this does not include elapsed time between models, and comparison between propagated models has been shown to be difficult [24]. Forward analysis can also be used for EQ5 by exhaustively propagating evaluation labels, but again this does not say how the individual labels connect. Backward analysis can partially answer EQ2 by providing possible solutions with and without the satisfaction of Off-Street Lots, and EQ6 by setting Cyclist Safety to be satisfied and finding values for the remaining intentions using backward analysis. However, backward analysis is not able to inform stakeholders how (or in what order) to satisfy these goals. Our original iStar approach [23] would be able to answer EQ4, EQ3, EQ5, and EQ6, but would not be able to answer EQ2 or EQ1 because it did not have a notion of absolute time (EQ2). Finally, prior work does not have any mechanism to address EQ1.

**Relation to software and systems** As noted above, we chose to use a city planning example rather than a software intensive system because of the availability of real-world data. Here, we argue that the decisions in this illustrative example are sufficiently equivalent to challenges in software projects. City planners do the work of requirements engineers for designing infrastructure systems. Typical of brown-field software development [50], city planners must elicit and collect elements in the domain and understand
the tradeoffs involved. They have incomplete information and are accountable to stakeholders (including users of the system). GORE approaches have not been extensively used in urban planning, but may act as a visual representation of the tabular approach used by planners in evaluating trade-offs [13].

Requirements engineers consider how the domain and context of their system evolve. EQ1 through EQ6 can be reframed as questions about software systems. For example, EQ2 can be rewritten as “What scenario ensures that Privacy Regulations is satisfied in six months, even if the Anonymous User feature becomes unavailable?” As well, EQ3 can be reframed as “Is Support All Mobile Platforms or Support iOS Platform most appropriate, given changes to the customer base?” Finally, EQ6 can become “Can the Data Storage actor eventually satisfy and maintain Secure Data?”

1.2 Contributions

In this article, we consider the problem of making trade-off decisions (e.g., EQ3) given anticipated changes in stakeholders’ intentions (i.e., goals, tasks, resources, and soft-goals). We hypothesize that goal modeling and, by extension, decision making in early-phase RE can be improved by adding temporal components to accommodate changes in intentions, enabling stakeholders to reason about model evolution. We ask the research question: how can we effectively support users in making trade-off decisions given evolutionary information? We address this research question by providing a complete formalization of the Evolving Intentions framework in Tropos, and demonstrate aspects using an illustrative example (called the bike lanes example). We prove correctness of the analysis, and validate the effectiveness of the approach by exploring a second (larger) example. Finally, we give evidence to the scalability of our analysis with runtime data.

In [26], we showed applicability of the Evolving Intentions framework. In addition to the contributions listed above, this paper is intended to give readers complete details of the formalization to ensure correctness of automated analysis. These details are especially important for the development of safety-critical systems, where explicit traceability between the analysis results and the underlying reasoning is paramount.

The remainder of the article is organized as follows. Section 2 introduces the formal aspects of this article in the context of our broader methodology. Section 3 gives a primer of the formal syntax and semantics for a goal graph, providing the background underlying our analysis. Section 4 defines our representation of time, Evolving Intentions, evolving relationships, and an evolving goal graph. Section 5 defines the notion of an evolving goal graph path and describes how to reason with evolving goal graphs. Section 6 demonstrates the applicability and usefulness of the proposed framework on the illustrative example. Section 7 reports on the implementation, effectiveness, and scalability of our approach. Section 8 contextualizes this work within related research and directly compares our work with a similar approach. We conclude in Sect. 9.

2 Evolving Intentions methodology

In this section, we give a brief overview of a guided procedure that enables stakeholders to generate and reason with goal models in the Evolving Intentions framework. We give forward references to parts of the formalism described in later sections of this article to give the reader intuition about the use of the formal elements. Figure 2 illustrates the process, which consists of three steps: (1) developing the base model; (2) adding evolutionary information; (3) answering questions with the model. Step 1 is the same for the Evolving Intentions framework as for the Tropos modeling language [16, 17, 49]; however, steps 2 and 3 are specific to the Evolving Intentions framework.

Step 1: developing the base model Modelers begin by brainstorming all the actors they wish to represent and adding them to the model. Modelers consider one actor at a time, and answer the following questions for each actor:

- How does the actor interact with the other actors already in the model and those not yet in the model?
- What goals motivate these interactions?
- Can any of the intentions be decomposed?
- Are there other ways of achieving the intentions of this actor?
- Why does this stakeholder want to achieve this goal (i.e., discover other motivations)?

By answering each question, modelers elicit and add intentions and relationships to the model. Repeat this process with each actor (possibly multiple times) until all modelers believe that the model is complete. Modelers should feel free to modify or add to the model in whatever way makes sense to them.
At the end of this step, stakeholders would have completed an untimed goal model, such as the model in Figs. 1 and 3; however, we omit the actor boundaries (e.g., Bike Lanes Project) from these figures because it is not relevant to the analysis formalization, which is the focus of this article. We discuss the formalization of this step in Sect. 3. Stakeholders can use forward and backward analysis to generate scenarios and answer trade-off questions using their untimed goal model.

**Step 2: adding evolutionary information** To add evolutionary information, modelers must first determine granularity and period of time over which they will consider the model, and create the model timeline (see Sect. 4.1 for details). Evolutionary information will be mapped onto this timeline.

Second, modelers identify the elements in the model whose evaluation (i.e., level of satisfaction or fulfillment) changes over time. For each element, modelers should consider three questions:

- How do these evaluations change?
- What assumptions do modelers have about this evolution?
- What alternative patterns are there for each intention?

For expected or known changes, modelers map these intentions to one or more evolving functions, and can specify the behavior of the intention over part or all of the timeline. See Sect. 4.2 for the specification of available evolving functions in the formalism.

Next, modelers determine if there are any relationships that evolve over time. For each of these relationships, modelers should consider the changes, assumptions, and alternatives as posed above. A relationship can exist over only a part of the timeline, or two separate relationship types can be specified over the full timeline. See Sect. 4.3 for the specification of evolving relationships.

Modelers can also create constraints between the evolving functions and relationship specified in the model, and give intentions initial evaluations. At this stage, the initial model is complete and must satisfy Def. 17 (see Sect. 4.4). For example, a fragment of the evolving goal graph of the Bike Lanes example is shown in Fig. 11b, with the formal specification presented in Fig. 11a.

**Step 3: answering questions with the model** At this point, stakeholders can generate scenarios and answer trade-off questions using their evolving goal model. We recommend using the following reasoning loop to sequentially pose questions of the model:

(a) Pose a question about the project.
(b) *Enriching the model* Consider what evolution is required for this question? Add evolving functions, evolving relationships, or constraints to test the particular intentions of interest.
(c) *Running analysis* Automatically create a simulation path (satisfying Def. 22), then review and interpret the results. Consider generating multiple paths to understand the stochastic nature of simulation results.
(d) *Update the Model* After gaining some insight from the analysis, update the model. Ask the question, what would happen if an alternative evolving function was chosen for each of the intentions and relationships?
(e) *Repeat* Return to step (a), until all stakeholders are satisfied with the results of reasoning.

Step 3 allows stakeholders to ask and answer EQ1–EQ6 (i.e., the modeler questions we introduced in Sect. 1.1 for the bike lanes example). In Sect. 6, we further describe how these questions are explored in the context of the formalism presented in Sect. 3–5. We discuss the formalization of this step in Sect. 5 and present a brute force algorithm to automatically find simulation paths in Algorithm 1.

**Summary.** Here, we described the broader methodology of the Evolving Intentions framework. The complete formalization described in Sect. 3 through Sect. 5 is not intended to be directly used by stakeholders; instead, the definitions provide a foundation for automated analysis (see Sect. 7). We return to this methodology in Sect. 9, where we discuss the limitations and broader applicability of the framework.

### 3 Preliminaries

In this section, we describe how we formally specify an untimed goal model in Tropos. This section corresponds to Step 1 of the Evolving Intentions methodology (see Sect. 2). We use the formalism presented by Giorgini et al. [16, 17, 49] as the basis for our work. In this section, we present and
adapt these definitions accordingly. To give new concepts a concrete meaning, we use a fragment of Fig. 1 model of the BLE introduced in Sect. 1.1. The model fragment is shown in Fig. 3.

**Model syntax** We first introduce the formal syntax and semantics of the goal model.

**Definition 1 (Goal Graph)** A goal graph is a tuple \((A, G, R)\), where \(A = \{\}\), \(G = \{(\text{Have Bike Lanes, goal}), (\text{Have Design, goal}), (\text{Build Bike Lanes, goal}), (\text{Temporary Construction, goal}), (\text{Permanent Construction, goal}), (\text{Cyclist Safety, goal}), (\text{Access to Parking, goal}), (\text{Bike Lane Usage, goal})\}\), \((R = \{\text{Have Bike Lanes} \rightarrow S \text{ Cyclist Safety}, \text{(Temp. Const.}, \text{Permanent Const.)} \rightarrow S \text{ Build Bike Lanes, Permanent Construction} \rightarrow S \text{ Access to Parking}\})\). A well-formed goal graph is a graph that has only one incoming n-ary relation, and (2) the graph does not contain any loops.

**Definition 2 (Evidence Pair)** Let an intention \(g \in G\) be given. An evidence pair is a pair \((s, d)\) where \(s \in \{F, P, \bot\}\) is the level of evidence for and \(d \in \{F, P, \bot\}\) is the level of evidence against the fulfillment of \(g\).

\[ F \ [\text{resp. } P] \text{ means there is full [resp. partial] evidence for or against the fulfillment of an intention, and } \bot \text{ represents null evidence. Figure 5 shows the lattice, on } \geq \text{ ("more true"), over the universe of all evidence pairs \(E\). Intentions can have one of five non-conflicting valuations: (Fully) Satisfied } (F, \bot), \text{ Partially Satisfied } (P, \bot), \text{ Partially Denied } (\bot, P), \text{ (Fully) Denied } (\bot, F), \text{ and None } (\bot, \bot). \text{ This combination produces four conflicting valuations: } (F, F), (F, P), (P, F), \text{ and } (P, P). \]

**Definition 3 (Intention Evaluation)** Let an intention \(g \in G\) be given. The evaluation of \(g\), denoted by \(\llbracket g \rrbracket\), is a mapping \(G \to E \cup \{\bot\}\). The evaluation of each intention is either an evidence pair or is set to a special value \(\bot^2\), meaning “no information.”

For example, \(\llbracket g \rrbracket = (F, \bot)\) means there is full evidence for and no evidence against the fulfillment of \(g\). Similarly, in the bike lanes example, \(\llbracket \text{HaveDesign} \rrbracket = (\bot, F)\) means that there is no evidence for and full evidence against the fulfillment of Have Design.

We define four evidence predicates \(FS(g), PS(g), PD(g),\) and \(FD(g)\) over the evidence pairs to indicate the level of evidence for either the satisfaction or denial of an intention (where * indicates any of \(F, P,\) or \(\bot\)):

\[ FS(g) \iff (\llbracket g \rrbracket = (F, *)) \]
\[ PS(g) \iff (\llbracket g \rrbracket = (F, *) \lor (P, *)) \]
\[ PD(g) \iff (\llbracket g \rrbracket = (*, P) \lor (*, *)) \]
\[ FD(g) \iff (\llbracket g \rrbracket = (*, F)) \]

For example, if \(PS(g)\) is true then there is at least partial evidence for the satisfaction of \(g\). This predicate clearly guarantees the expected ground axioms from [17]:

\[ ^2 \text{ Note the difference between the evidence pair } (\bot, \bot) \text{ meaning None and the special value } \bot \text{ meaning “no information.”} \]
For example, if there is full evidence that $g$ is satisfied, then there is also partial evidence that $g$ is satisfied, since $F \geq P$.

**Relationship semantics and propagation**

We now describe the meaning of the relationships defined in Definition 1. N-ary intention relationships (resp. $(g_1, \ldots, g_n) \rightarrow g$) mean that the fulfillment of $g$ requires all (resp. only one) of $g_1, \ldots, g_n$ to be satisfied. The binary relationship $(g_1, g_2) \rightarrow g$ means the absence of a relationship. The binary relationship $(g_1, g_2) \rightarrow g$ means if $g_1$ is satisfied, then there exists some [resp. full] evidence that $g$ is satisfied. $(g_1, g_2) \rightarrow g$ means that if $g_1$ is denied, then there exists some [resp. full] evidence that $g$ is denied. $(g_1, g_2) \rightarrow g$ combines the +S and +D [resp. ++S and ++D] relationships to propagate some [resp. full] evidence for $g$ when there is any evidence for $g_1$. Similarly, $(g_1, g_2) \rightarrow g$ says that if $g_1$ is satisfied then there exists some [resp. full] evidence that $g$ is denied. The -D, -D, - and - relationships follow the same pattern and structure.

The evaluation of a graph $(A, G, R)$ is a process that assigns each intention $g \in G$ an evidence pair either directly or via propagation. When propagation goes from source nodes to target nodes, it uses rules in lines 1–20 in Fig. 6, and is called **forward propagation**. Backward propagation (lines 21–40 in Fig. 6) propagates evidence from target intentions to source intentions and is a way of obtaining values for target nodes. Backward propagation rules are derived directly from the forward propagation rules, when all rules are taken together.

For example, the forward propagation of a +S relationship between $g_1$ and $g$ (see line 9 in Fig. 6) means that if there is partial evidence for the fulfillment of $g_1$, then there is partial evidence for the fulfillment of $g$. In a different example, if there is full evidence for the satisfaction of $g$ (see lines 21–24 in Fig. 6) then using backward propagation, at least one of the relations must have supplied this full evidence. If there is an and relation between $(g_1, \ldots, g_n)$ and $g$ (see line 23) then all source intentions $g_1, \ldots, g_n$ must have full satisfaction evidence. Alternatively, if there is a ++S (or ++) relationship between $g_i$ and $g$ (see line 23) then the source intention $g_i$ must have full satisfaction evidence.

In the bike lanes example (see Fig. 3), the forward propagation of the + relationship between Permanent Construction and AccessToParking means that if there is partial evidence for the fulfillment of PermanentConstruction, then there is partial evidence for the fulfillment of AccessToParking. This + relationship also means that if there is partial evidence
against the fulfillment of Permanent Construction, then there is partial evidence against the fulfillment of Access to Parking.

**Goal graph evaluation** As mentioned above, in evaluating a goal graph, we assign evidence pairs either directly or via propagation. In order to evaluate a goal graph with the propagation rules, we must first define conflicting values and introduce how stakeholders directly assign evidence pairs.

As mentioned above, the universe of all evidence pairs $E$ contains four conflicting evidence pairs: $(F, F), (F, P), (P, F), (P, P)$. In evaluating a goal graph, we may want to avoid these conflicting values where possible. We specify this avoidance with **conflict levels**.

**Definition 4 (Conflict Levels)** The set Weak _ Conflict = $\{ (F, F), (F, P), (P, F), (P, P) \}$ represents conflicting evidence pairs, i.e., pairs of values that should not be assigned to an intention. In addition, we define subsets of Weak _ Conflict which are often meaningful: Strong _ Conflict = $\{ (F, F) \}$, Medium _ Conflict = $\{ (F, F), (F, P), (P, F) \}$, and None _ Conflict = $\{ \}$.

Medium _ Conflict avoidance requires that an intention cannot be fully satisfiable and (fully or) partially deniable, and vice versa. Strong _ Conflict avoidance requires that an intention cannot be fully satisfiable and fully deniable. In ideal scenarios, stakeholders may avoid all conflicting values (i.e., Weak _ Conflict), but this is not always possible. Strong _ Conflicts should be avoided because they are rarely helpful in decision making. When an intention has full evidence for and against its fulfillment, this is likely due to contribution links, and stakeholders should explore developing the model further. Part of the GORE approach is making tradeoff decisions when the best alternative is not obvious. For this reason, Weak _ Conflict and Medium _ Conflict are generally tolerated because not all soft-goals or non-functional requirements can be satisfied by any alternative.

Goal evaluations can be assigned by users prior to evaluating the graph as a whole. Goal evaluations allow stakeholders to assign desired evidence pairs to intentions for analysis, as well as evidence pair assignments that are known to be true prior to analysis.

**Definition 5 (User Evaluation)** Let a goal graph $\langle A, G, R \rangle$ be given. A set of user evaluations $\text{UEval} = \{ g \in E \mid g \in G \wedge e \in E \}$ is the user’s assignment of evidence pairs to intentions.

For example, a set of user evaluations for the Bike Lanes example could be $\{ (\text{Have Bike Lanes}, (F, \perp)), (\text{Temporary Construction}, (F, \perp)), (\text{Access to Parking}, (L, P)) \}$.

With these inputs, we can now define evaluation over a goal graph.

**Definition 6 (Goal Graph Evaluation)** Let a well-formed goal graph $\langle A, G, R \rangle$, a set of user evaluations $\text{UEval}$, and a conflict level $\text{CFLevel}$ be given. A complete evaluation of $\langle A, G, R \rangle$ is the total mapping $G \rightarrow E$ resulting from the repeated assignment of evidence pairs to intentions and application of the propagation rules in Fig. 6 to all relationships in $R$, given $\text{UEval}$, such that $\forall g \in G : \| g \| \notin \text{CFLevel} \wedge (\forall (g, e) \in \text{UEval} : e = \| g \|)$.

Once a goal graph is evaluated, all goals must have an evaluation, which may include None ($\perp, \perp$). Thus no goal may still have the value $\perp$: “no information”.

Infeasible evaluations occur when the evaluation is over-constrained by one or more of the goal graph relationships, the user evaluations, and the conflict level. Consider the relationship $(\text{Have Design}, \text{Build Bike Lanes}) \rightarrow \text{Have Bike Lanes}$ from the bike lanes example (see Fig. 3). If stakeholders add the following two user evaluations:

$\{ (\text{Have Design}, (\perp, F)), (\text{Have Bike Lanes}, (F, \perp)) \}$

the resulting goal graph evaluation is infeasible, because it is not possible for $\text{Have Bike Lanes}$ to be satisfied (i.e., have the evidence pair $(F, \perp)$), while $\text{Have Design}$ is denied (i.e., with the evidence pair $(\perp, F)$) as this violates the propagation rules of the and relationship.

**4 Modeling Evolving Intentions**

In this section, we describe how we use and extend definitions in Sect. 3 to define how the evaluation of an intention changes over time, as well as the evolution of entire goal graphs over time. This section corresponds to Step 2 of the Evolving Intentions methodology (see Sect. 2). We illustrate them using the extension of the BLE model fragment shown in Fig. 3.

**4.1 Time points and intention evaluations**

In this work, we use discrete time where a tick is a user-specified minimal distance between two observable events. Time is measured in equidistant ticks from a user-defined notion of beginning (zero) to a user-defined notion of end ($\text{maxTime}$). The universe of time is $T = \{ t \mid t \in \mathbb{N}^0 \wedge t \in [0, \text{maxTime}] \}$. This means that users can select an arbitrary unit of time for a tick (e.g., day, month, six months, two years, etc.), and then each tick will map on to real-world events from the user-defined beginning (e.g., today, 01/01/1963, next Tuesday).

**Time Points (of Interest)** are a collection of timed variables $P$. In this paper, elements in $P$ are denoted by $t_i$, where $i$ is the variable name. The range of values for each time point...
of interest in $P$ is $T \cup \{\bot, \text{maxTime} + 1\}$. We use these time points to evaluate goal graphs. Figure 7 illustrates an example mapping between time points and the universe of time (measured in ticks). The top line shows a sequence of ticks starting at zero. The boxes underneath represent the time points in $P$. Upon declaration of a model, time points may be assigned an absolute value or be defined relative to other time points. For example, $t_5$ and $t_8$ do not have time assignments. Throughout this section, time points are introduced to specify how the evaluation of an intention changes, but can also be included on their own as in Definition 16. Note that we include $\{\text{maxTime} + 1\}$ in $P$ with the rationale that we define intervals over time points that are open on the right, and we want to make sure that $\text{maxTime}$ can be included.

We use $t_0 = 0$ to denote the initial time value, and $t_{\text{end}} = \text{maxTime} + 1$ to denote the end of all open intervals where the final value could be $\text{maxTime}$. In the bike lanes example (BLE), the stakeholders chose to represent one month of real-world time as a tick and want to perform analysis over 4 years and 2 months, thus $\text{maxTime} = 50$ and $t_{\text{end}} = 51$.

**Definition 7** (Intention evaluation at a point) Let an intention $g \in G$ and a time point $t \in P$ be given. The evaluation of $g$ at $t$, denoted by $[g[t]]$, is a mapping $G \times P \rightarrow E \cup \{\bot\}$.

If the evaluation of goal $g$ at time $t$ is not specified by the model, then $[g[t]]$ returns $\bot$. Additionally, the evaluation of goal $g$ at time $t$ can be specified to have a value of None, in which case $[g[t]]$ will return $(\bot, \bot)$.

When simulating paths in the next section, all time points will be resolved to be in $T$; thus, in order to enable abstract reasoning about these time points and simplify the presentation, we use the variable and value for a time point interchangeably. For example, using $t_3$ from Fig. 7, $[g[t_3]]$ is functionally equivalent to $[g[12]]$, but $[g[\bot]]$ is nonsensical.

### 4.2 Evolving functions for intentions

In this section, we introduce functions that describe how the evaluation of an intention changes over time.

**Atomic functions** Between two time points, the evaluation of an intention can become more true, or $\text{INCREASE}$; become more false, or $\text{DECREASE}$; remain $\text{CONSTANT}$; or change randomly, displaying a $\text{STOCHASTIC}$ pattern.

We define atomic functions within an interval of time points $[\text{start}, \text{stop})$, where $\text{start}, \text{stop} \in P$, given a special value $x$ denoting a reference evidence pair.

**Definition 8** (Atomic Functions) Let an intention $g \in G$, a set of time points $P$, a reference evidence pair $x \in E \land x \notin \text{Weak Conflict}$, and time points $\text{start}, \text{stop} \in P$ (where $\text{start} < \text{stop}$) be given. An atomic function defines how the evaluation of $g$ changes over the time points in $P$ within an interval $[\text{start}, \text{stop})$, based on four types.

(a) **CONSTANT** says that the evidence pair assignment of an intention $g$ remains equal to $x$.

\[
\forall t_i \in P : t_i \in [\text{start}, \text{stop}) \implies \[g[t_i]\] = x;
\]

(b) **INCREASE** means that values in the evidence pair assignment become “more true”, up to a maximum value $x$, as time progresses.

\[
\forall t_i, t_j \in P : t_i, t_j \in [\text{start}, \text{stop}) \land t_i < t_j \implies \[g[t_i]\] \leq \[g[t_j]\] \leq x;
\]

(c) **DECREASE** means that values in the evidence pair assignment become “less true”, down to a minimum value $x$, as time progresses.

\[
\forall t_i, t_j \in P : t_i, t_j \in [\text{start}, \text{stop}) \land t_i < t_j \implies \[g[t_i]\] \geq \[g[t_j]\] \geq x;
\]

(d) **STOCHASTIC** means that values in the evidence pair assignment are stochastic or random, but do not include conflicting values.

An atomic function is the tuple $(\text{type}, x, \text{start}, \text{stop})$, that can be associated with an intention, where type is one of **CONSTANT**, **INCREASE**, **DECREASE** or **STOCHASTIC**.

Suppose $\text{Have Design}$ is assigned the atomic function $(\text{CONSTANT}, (\bot, 1), t_0, t_{\text{design}})$, where $t_{\text{design}} = 15$. This would constrain the evaluation of $\text{Have Design}$ to be $(\bot, 1)$ at all time points from $t_0$ up to but not including $t_{\text{design}} = 15$, including initially undefined time points (i.e., $\bot$) who are assigned as less than 15 through analysis (see Definition 21). Similarly, **INCREASE** and **DECREASE** constrain the evaluation of an intention over the entire interval $[\text{start}, \text{stop})$, despite not prescribing valuations prior to analysis.

In Definitions 10 and 11, we combine atomic functions. We identify the $\text{start}$ and $\text{stop}$ elements of a specific function using bracket notation. For a given atomic function $a_i(t)$, $a_i(\text{start})$ denotes the start time point in the atomic function; similarly, $a_i(\text{stop})$ denotes the stop time point.

**Generalized function** When evaluating intentions, modelers can use atomic functions on their own or can create compound functions by defining an ordered list of atomic functions over an ordered set of time intervals.
is the tuple $\perp$

Marks the beginning of summer and the transition to a t function with maxi-
t...
The time when a decision between

Marks the beginning of fall and the transition to a Increase $\perp$
ta
Graph constraint time point 7 maxTime
Marks the beginning of spring and the transition to an constant
The time when
The time when the bike lanes opened and the end of the negative impact of $\perp$
t⟩ $\bigcirc$
t

Table 1 Initial time points associated with the bike lanes example The following time points are used to describe formal concepts and explain the bike lanes example

<table>
<thead>
<tr>
<th>Sections</th>
<th>Time points</th>
<th>Description</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>$t_0$</td>
<td>Start of analysis</td>
<td>0</td>
</tr>
<tr>
<td>4.1</td>
<td>$t_{end}$</td>
<td>$maxTime$ is the final point of analysis. This time point ensures that $maxTime$ can be selected as a point in the last interval of each evolving function</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>$t_{use}$</td>
<td>Marks the end of the $\text{CONSTANT } (L, F)$ period for Bike Lane Usage, which is the time when the construction of the bike lanes is finished. Also marks the beginning of the repeating period, and the first $\text{STOCHASTIC}$ function</td>
<td>10</td>
</tr>
<tr>
<td>4.2</td>
<td>$t_{spring}$</td>
<td>Marks the beginning of spring and the transition to an $\text{INCREASE}$ function in Bike Lane Usage</td>
<td>$\perp$</td>
</tr>
<tr>
<td>4.2</td>
<td>$t_{summer}$</td>
<td>Marks the beginning of summer and the transition to a $\text{CONSTANT}$ function at $(F, \perp)$ in Bike Lane Usage</td>
<td>$\perp$</td>
</tr>
<tr>
<td>4.2</td>
<td>$t_{fall}$</td>
<td>Marks the beginning of fall and the transition to a $\text{DECREASE}$ function in Bike Lane Usage</td>
<td>$\perp$</td>
</tr>
<tr>
<td>4.2</td>
<td>$t_{design}$</td>
<td>The time when Have Design became satisfied and a design has been created</td>
<td>$\perp$</td>
</tr>
<tr>
<td>4.3</td>
<td>$t_{dec}$</td>
<td>The time when a decision between Permanent Construction and Temporary Construction is made</td>
<td>5</td>
</tr>
<tr>
<td>4.4</td>
<td>$t_{stop}$</td>
<td>Graph constraint time point</td>
<td>7</td>
</tr>
<tr>
<td>4.4</td>
<td>$t_{5}$</td>
<td>Graph constraint time point</td>
<td>14</td>
</tr>
<tr>
<td>4.4</td>
<td>$t_{3}$</td>
<td>Graph constraint time point</td>
<td>34</td>
</tr>
</tbody>
</table>

In this example, $maxTime$ is assigned by the stakeholder to be 50

Definition 9 (Disjoint Neighboring Intervals) Intervals $[t_1, t_2), [t_3, t_4), [t_5, t_6] \ldots$ are disjoint neighboring intervals if $t_2 = t_3 \wedge t_4 = t_5$.

Definition 10 (Compound Function) Let a sequence of atomic functions $A$ be given, where $a_i \in A$ is the tuple $\langle \text{type}, x, \text{start}, \text{stop} \rangle$ and $\forall a_i \in A : a_i(\text{stop}) = a_{i+1}(\text{start})$. A compound function is a sequence of atomic functions over disjoint neighboring intervals (see Definition 9) that define how the evaluation of an intention changes over time.

For example, Bike Lane Usage (see Fig. 3) can be represented by the compound function illustrated in Fig. 8, and specified by the sequence

$$\{ \langle \text{STOCHASTIC}, (L, \perp) \rangle, t_0, t_{spring} \}, \langle \text{INCREASE}, (F, \perp), t_{spring}, t_{summer} \rangle, \langle \text{CONSTANT}, (F, \perp), t_{summer}, t_{fall} \rangle, \langle \text{DECREASE}, (\perp, F), t_{fall}, t_{end} \rangle \}.$$  

This consists of four atomic functions, each defining one of the four seasons (starting with winter at $t_0$): a STOCHASTIC function over $[t_0, t_{spring})$; an INCREASE function with maximum value $(F, \perp)$ over $[t_{spring}, t_{summer})$; a CONSTANT function at $(F, \perp)$ over $[t_{summer}, t_{fall})$; and a DECREASE function at $(\perp, F)$ over $[t_{fall}, t_{end})$. The intuition behind choosing these functions is that usage is dependent on the season. Depending on the mildness of winter, residents use bike lanes to varying degrees (STOCHASTIC). There is a steady increase in usage throughout spring (INCREASE) and a steady decrease in usage throughout fall (DECREASE), with consistently high (or satisfied) usage in summer (CONSTANT). All time points used to specify the bike lanes example are presented together in Table 1 for reference.

Definition 11 (Repeating Compound Function) Let a sequence of atomic functions $A$ be given, where $a_i \in A$ is the tuple $\langle \text{type}, x, \text{start}, \text{stop} \rangle$ and $\forall a_i \in A : a_i(\text{stop}) = a_{i+1}(\text{start})$. A repeating compound function is the tuple $\langle \text{REPEAT}, \text{repNum}, \text{absTime}, A \rangle$, where REPEAT indicates a repeating function, $A$ is the given sequence of atomic functions, $\text{repNum} \in \mathbb{N}^*$ is the number of repeats for the entirety of $A$, and $\text{absTime} \in \mathbb{N}^0$ is the absolute time for each function or zero if no value is specified. A repeating compound function is a sequence of atomic functions over disjoint

Fig. 8 Sequence of atomic functions over time points $t_0 \ldots t_{end}$
neighboring intervals that repeat repNum to define how the evaluation of an intention \( g \) changes over time.

For example, the compound function in Fig. 8 is repeated three times in Fig. 9. New time points are generated to define the repeat. For example, in Fig. 9 \( r_1\_t_{spring} \) identifies the first instance of \( t_{spring} \) in the first repeat and \( r_2\_t_0 \) identifies the beginning of the second repeat. The last time point in the sequence \( A \) is paired with the first time point for the next repeat (i.e., \( r_1\_t_{end} = r_2\_t_0 \)). We model the first three years of Bike Lane Usage with this repeating function, where we model each new winter (until \( t_{spring} \)) with a Stochastic function. The repeating version of Bike Lane Usage is specified by the tuple

\[
\langle \text{REPEAT}, 3, 0, \rangle \\
\{ \langle \text{STOCHASTIC}, (\perp, \perp), t_0, t_{spring} \rangle, \langle \text{INCREASE}, (F, \perp), t_{spring}, t_{summer} \rangle, \langle \text{CONSTANT}, (F, \perp), t_{summer}, t_{fall} \rangle, \langle \text{DECREASE}, (\perp, F), t_{fall}, t_{end} \rangle \}. 
\]

**Definition 12 (User-Defined Function)** Let an intention \( g \in G \) be given. A User-Defined function is an arbitrary composition of compound and repeating compound functions (Definition 10 and 11) over disjoint neighboring intervals (one function per interval) that define how the evaluation of an intention \( g \) changes over time.

To model Bike Lane Usage with a User-Defined function most accurately, we should include a period of no usage at the beginning of the function until the bike lanes are built (full evidence against the satisfaction of Bike Lane Usage). The final version of the function for Bike Lane Usage is specified by the tuple

\[
\{ \langle \text{CONSTANT}, (\perp, F), t_0, t_{use} \rangle, \langle \text{REPEAT}, 3, 0, \rangle \\
\{ \langle \text{STOCHASTIC}, (\perp, \perp), t_{use}, t_{spring} \rangle, \langle \text{INCREASE}, (F, \perp), t_{spring}, t_{summer} \rangle, \langle \text{CONSTANT}, (F, \perp), t_{summer}, t_{fall} \rangle, \langle \text{DECREASE}, (\perp, F), t_{fall}, t_{end} \rangle \}. 
\]

where \( t_{use} \) is defined as when the construction of the bike lanes is finished.
Common compound functions. Given the User-Defined function, we define, for convenience, the functions listed in Table 2, which we call common compound functions. For example, in Table 2, the CONSTANT-STOCHASTIC function definition consists of two atomic functions: first, a CONSTANT function over the interval \([e_0, e_1]\) with a constant value of \(x\), and second, a STOCHASTIC function over the interval \([e_1, e_2]\). \((\bot, \bot)\) is used as the reference evidence pair in the interval \([e_1, e_2]\) because STOCHASTIC does not require a value.

For the bike lanes example (see Fig. 3), we define Have Design as having a Denied-Satisfied function: \(\langle\text{Denied-Satisfied}, (F, \bot), \{0, t_{\text{design}}, \text{maxTime}\}\rangle\), which is the same as the User-Defined function
\[
\{\langle\text{Constant}, (\bot, F), t_0, t_{\text{design}}\rangle, \\
\langle\text{Constant}, (F, \bot), t_{\text{design}}, t_{\text{end}}\rangle\},
\]
where \(t_{\text{design}}\) is the time when Have Design becomes satisfied and a design has been created. So the value of Have Design is \((\bot, F)\) until a design is accepted at \(t_{\text{design}}\), and then it is \((F, \bot)\). The intuition behind this function is that a task starts as not completed and remains so until its completion time.

### 4.3 Evolving functions for relationships

In addition to having Evolving Intentions, we allow stakeholders to specify evolving relationships.

**Definition 13** (N-ary evolving relationship) Let intentions \((g_1, \ldots, g_n)\), \(g \in G\), relationships \(r_1, r_2 \in \{\text{and}, \text{or}, \text{no}\}\), and a set of time points \(P\), and a reference time point \(t_{\text{Ref}} \in P\) be given. An n-ary relationship \((g_1, \ldots, g_n) \xrightarrow{r_{Ref}} g\) is evolving with respect to \(t_{\text{Ref}}\) if
\[
\forall t_i \in P \begin{cases} (g_1, \ldots, g_n) \xrightarrow{r_1} g, & \text{if } t_i < t_{\text{Ref}} \\ (g_1, \ldots, g_n) \xrightarrow{r_2} g, & \text{otherwise.} \end{cases}
\]
An n-ary relationship is constrained by \(r_1\) prior to time point \(t_{\text{Ref}}\) and \(r_2\) on or after \(t_{\text{Ref}}\).

This allows stakeholders to change the nature of a decomposition relationship. For example, consider the case where the initial intention is to fulfill the target intention, but after a time, both are required. In the bike lanes example, we see this as a change in government regulations. Prior to a new regulation, bike lanes could be identified by either painted lines or flexible bollards (i.e., posts), but after the change both are required to identify the bike lanes. This bike lanes decomposition relationship can be represented with an \(\xrightarrow{t_1}\) link, where over the same intentions (i.e., Paint Lines and Install Bollards), an or relationship holds until some \(t_1\), and then an and relationship holds.

**Definition 14** (Binary evolving relationship) Let intentions \(g_1, g_2 \in G\), relationships \(r_1, r_2 \in \{\text{and}, \text{or}, \text{no}\}\), and a set of time points \(P\), and a reference time point \(t_{\text{Ref}} \in P\) be given. A binary relationship \(g_1 \xrightarrow{r_{Ref}} g_2\) is evolving with respect to \(t_{\text{Ref}}\) if
\[
\forall t_i \in P \begin{cases} g_1 \xrightarrow{r_1} g_2, & \text{if } t_i < t_{\text{Ref}} \\ g_1 \xrightarrow{r_2} g_2, & \text{otherwise.} \end{cases}
\]
A binary relationship is constrained by \(r_1\) prior to time point \(t_{\text{Ref}}\) and \(r_2\) on or after \(t_{\text{Ref}}\).

Similarly, the strength or type of a contribution relationship may change over time. This evolution changes the impact one goal may have on another. For example, choosing a design with Permanent Construction impacts Access to Parking but only while the construction is taking place. After construction is complete and the bike lanes become open (at \(t_{\text{BLOpen}}\)), Permanent Construction has no impact on Access to Parking. We can model the relationship between Permanent Construction and Access to Parking with an evolving no relationship. This relationship, denoted by

\[
\text{Permanent Construction} \xrightarrow{\text{no}} \text{Access to Parking},
\]
allows us to model the negative impact of construction on Access to Parking, where significant road work is required until \(t_{\text{BLOpen}}\), which denotes the time when the construction is complete and the bike lanes become open. Prior to time point \(t_{\text{BLOpen}}\), if Permanent Construction is fully (or partially) satisfied, then there exists full (or partial) evidence that Access to Parking is denied, and after \(t_{\text{BLOpen}}\) no evidence is propagated.

**NotBoth function** We added a new relationship function specifically for the purpose of showing when a mutually exclusive decision is made in the graph over time. This relationship guarantees that only one alternative is chosen in any analysis procedure, where prior to making the decision, each alternative did not impact other intentions. We have two variations that specify the evidence pair for the unchosen alternative: NotBoth-Denied (NBD) where the unchosen alternative is assigned Denied \((\bot, F)\), and NotBoth-None (NBN) where the unchosen alternative is assigned None \((\bot, \bot)\).

**Definition 15** (NotBoth Evolving Function) Let intentions \(g_1, g_2 \in G\), a set of time points \(P\), and a reference time point \(t_{\text{Ref}} \in P\) be given, where \(g_1\) and \(g_2\) are not elements of any atomic functions. \(x\) is a special evidence pair used to define the final value, where \(x = (\bot, F)\) for NBD and \(x = (\bot, \bot)\) for NBN. A NotBoth evolving function between intentions \(g_1\) and \(g_2\) says that the relationship in Fig. 10 holds for given
values of $t_{\text{Ref}}$ and $x$. We denote the **NotBoth Evolving Function** as $g_1 \leftarrow t_{\text{Ref}} \rightarrow g_2$, and $g_1 \leftarrow t_{\text{Ref}} \rightarrow g_2$.

For example, we can represent the mutually exclusive decision between Permanent Construction and Temporary Construction by modifying the graph in Fig. 3 with the NotBoth evolving function

\[
\text{Permanent Construction} \xrightarrow{t_{\text{decision}}} \text{Temporary Construction},
\]

where $t_{\text{decision}}$ denotes the time when a decision between Permanent Construction and Temporary Construction is made. Prior to the time point $t_{\text{decision}}$, no decision has been made, and both intentions have the evidence pair $(\bot, \bot)$. After $t_{\text{decision}}$, only one intention has the evidence pair $(F, \bot)$, and the other has $(\bot, \bot)$. This relationship means that only one of the alternatives Permanent Construction and Temporary Construction can be chosen. The chosen alternative becomes satisfied and can impact other intentions, while the other alternative does not impact other intentions.

The **NotBoth function** contains the evaluations of two intentions. If either of these intentions are assigned atomic or User-Defined functions, then the graph may become over constrained preventing the computation of a complete graph evaluation path (see Definition 22). In this formalization, we do not prohibit additional evolving functions for intentions constrained by a NotBoth function, but tool designers may want to consider these restrictions.

### 4.4 Specifying evolving models

Stakeholders can specify constraints between time points and give absolute time assignments to time points.

**Definition 16** (Graph Constraints) Let time points $t_1, t_2 \in P$ be given. A graph constraint is a relationship between $t_1$ and $t_2$, such that the graph constraint is of the form $t_1 < t_2$, $t_1 = t_2$, $t_1 = k$, or $t_1 \neq k$, where $k \in \mathbb{N}^0 \land k \in [0, \text{maxTime})$. 

![Fig. 10](image1.png) Specification of NotBoth evolving function (Definition 15)

The Bike Lanes Goal Graph is $\langle A, G, R, EF, MC, \text{maxTime} \rangle$ where, $A = \{\}$,

$G = \{\langle \text{Have Bike Lanes}, \text{goal} \rangle, \langle \text{Build Bike Lanes}, \text{goal} \rangle, \langle \text{Temporary Construction}, \text{goal} \rangle, \langle \text{Permanent Construction}, \text{goal} \rangle, \langle \text{Cyclist Safety}, \text{goal} \rangle, \langle \text{Access to Parking}, \text{goal} \rangle, \langle \text{Bike Lane Usage}, \text{goal} \rangle\}$,

$R = \{\langle \text{Have Design}, \text{Build Bike Lanes} \rangle \rightarrow \langle \text{Have Bike Lanes} \rangle, \langle \text{Temporary Construction}, \text{Permanent Construction} \rangle \rightarrow \langle \text{Build Bike Lanes} \rangle, \langle \text{Have Bike Lanes} \rangle \rightarrow \langle \text{Cyclist Safety} \rangle, \langle \text{Permanent Construction} \rangle \rightarrow \langle \text{Access to Parking} \rangle, \langle \text{Bike Lane Usage} \rangle \rightarrow \langle \text{Temporary Construction} \rangle\}$,

$EF = \{\langle \text{Have Design}, \langle \text{DENIED-SATISFIED}, (F, \bot) \rangle, \{t_0, t_{\text{design}}, t_{\text{end}}\} \rangle, \langle \text{Bike Lane Usage}, \{\langle \text{CONSTANT}, (\bot, F), t_0, t_{\text{use}}\rangle, \langle \text{REPEAT}, 3, 0, \langle \text{STOCHASTIC}, (\bot, \bot), t_{\text{use}}, t_{\text{spring}}\rangle, \langle \text{INCREASE}, (F, \bot), t_{\text{spring}}, t_{\text{summer}}\rangle, \langle \text{CONSTANT}, (F, \bot), t_{\text{summer}}, t_{\text{fall}}\rangle, \langle \text{DECREASE}, (\bot, F), t_{\text{fall}}, t_{\text{end}}\rangle \rangle \rangle \}$,

$MC = \{t_{\text{design}} < t_{\text{decision}}, t_{\text{use}} = t_{\text{BLOpen}} = 10, t_{\text{decision}} = 5, t_{a1} = 7, t_{a2} = 14, t_{a2} = 34\}$, and 

$\text{maxTime} = 50$.

![Fig. 11](image2.png) Graph fragment and specification of bike lanes example with evolving elements.

(a) Specification of a fragment of Bike Lanes example goal graph with evolving elements.

(b) Fragment of Bike Lanes example goal graph with evolving elements.
In Table 1, we described $t_{design}$, $t_{decision}$, $t_{BLOpen}$, and $t_{use}$, which are all used in the bike lanes example. We can define constraints between these times, such that $t_{design} < t_{decision}$, $t_{use} = t_{BLOpen} = 10$, and $t_{decision} = 5$. Graph constraints do not need to be associated with evolving elements in the graph. Users can name time points and assign absolute values to them. For example, in Table 1, we added the constraints $t_{a0} = 7$, $t_{a1} = 14$, and $t_{a2} = 34$. Graph constraints cannot be added inside any repeating segment of a User-Defined function because these repeating time points are not explicitly part of the goal graph specification. We return to this point in Sect. 4.2.

We extend the definition of a goal graph (see Definition 1) to include evolutionary concepts.

**Definition 17** (Evolving goal graph) An evolving goal graph is a tuple $(A, G, R, EF, MC, maxTime)$, where $A$ is a set of actors, $G$ is a set of intentions, $R$ is a set of relationships over intentions (including evolving relationships and NotBoth evolving functions), $EF$ is a set of mappings between elements in $G$ and User-Defined functions (see Definition 12), $MC$ is a set of graph constraints over the evolving functions and relationships, and $maxTime \in \mathbb{N}^+$ is the maximum absolute time over which any evolving function is defined. A well-formed evolving goal graph is a graph where: (1) each intention has at most one incoming n-ary relation, and (2) the graph does not contain any loops.

From this point on, when we say “goal graph,” we mean a well-formed evolving goal graph. With this new definition, we update the goal graph from Fig. 3 with the evolving elements and present the new goal graph in Fig. 11b, with the specification in Fig. 11a. Each element in the visual graph maps to a portion of the specification. The intentions and relationships are listed first. Notice the specification of the relationships $R$ in Fig. 11a now has the evolving relationship and the NotBoth evolving functions (discussed in Sect. 4.3). This graph has only two evolving functions (see $EF$ in Fig. 11a), one for Have Design and a much longer one for Bike Lane Usage (both explained in detail in Sect. 4.2). $MC$ lists the constraints added between time points in the graph.

**4.5 Incompleteness**

In the previous subsections, we described how to model with Evolving Intentions framework. But the set of function types is not complete for describing all possible evolutions. The following list gives examples that cannot be expressed in the framework:

- Atomic functions for intentions with presence conditions.
- Repeating functions with an arbitrary number of repeats.
- Repeating functions that should repeat until an independent event (e.g., an intention is assigned a specific evidence pair).
- Repeating functions with absolute time points specified during the repeating portion, or absolute time periods for individual segments in the repeat.
- Evolving relationships with more than two relationships.
- Mixing n-ary and binary evolving relationship types.
- More than two intentions involved in a single NotBoth relationship.

Although we are aware of these limitations, we have not encountered concrete examples that require these additions. Future work may extend this formalism for completeness.

**5 Reasoning with Evolving Intentions**

In the previous section, we provided a formal definition of an evolving goal graph (see Definition 17) that enables modelers to specify changes in intentions and relationships within a goal graph. In this section, we define how goal graphs can be evaluated over time creating path-based simulations for the purpose of answering time-based trade-off questions. This section corresponds to Step 3 of the Evolving Intentions methodology (see Sect. 2).

**Designating graph time points** As introduced in Sect. 4.1, $P$ is a collection of time point variables representing absolute time values (in terms of ticks). Throughout Sect. 4, we used these time points to define events within functions for Evolving Intentions and relationships. Time points are also defined in terms of absolute and relative model constraints. Thus, within the definition of a goal graph, (see Definition 17) elements in $P$ are defined as part of MC, EF, and R. In this section, we create a time point path by collecting all the time points defined in the goal graph and assigning them absolute time values. We begin by defining this collection.

**Definition 18** (Goal Graph Time Points) Let a goal graph $M = \langle A, G, R, EF, MC, maxTime \rangle$ be given. Let $TP(M)$, called the graph time points, be the set of all time point variables named in the specification of $M$. $TP(M)$ returns the time points with unrolled repeating segments.

From the specification of the bike lanes example in Fig. 11a, we list all relevant constraints for time points in Table 3, as well as the origin of each constraint. For example, Row Set 1 lists the constraints associated with the evolving function for Have Design, where $t_0 < t_{design} < t_{end}$. Row Set 5 in Table 3 lists all the constraints from the $MC$ specification in Fig. 11a.

Row Set 2 in Table 3 lists the partial order over the time points associated with the evolving function for Bike
Lane Usage. In its specification (see Fig. 11a), Bike Lane Usage contains a Constant function, followed by four atomic functions that repeat three times. Prior to performing any analysis on the goal graph, we first unroll repeating segments and create new time points for each repeat of the form \( r_{i,j} \) for each repeat \( i \) and time point \( t_j \). In the example, the first repeat starts with \( r_{1,1} \) and ends with \( r_{1,3} \). The end of each repeat is set to be equal to the start of the next repeat, so \( r_{1,3} = r_{2,1} \) and so on. Notice that the original time points within the repeating portion (see \( t_{\text{spring}}, t_{\text{summer}}, \) and \( t_{\text{fall}} \) in Fig. 11a) do not appear in Row Set 2 of Table 3. Since these time points are internal to the repeat, they are removed, and are not allowed in constraints between time points (i.e., Row Set 5, see also Sect. 4.4). Thus, using \( t_{\text{spring}} \) or any repeating time point (e.g., \( r_{2,1} \)) in a graph constraint is not allowed. With the constraints listed in Table 3, we can create a partial order over all the time points in the graph.

Given the specification of the bike lanes example (see Fig. 11(a)), \( TP(M) \) would return the time points listed in Tables 3 and 4 (Time Points (Variable) column). Prior to analysis, the only time points assigned absolute values had been \( t_0, t_{\text{end}}, \) and graph constraints of the form \( t_i = k \) (see Definition 4.4 and Table 3 Row Set 5).

User evaluations We define what it means to evaluate an intention at a time point by extending Definitions 5 and 6. Evidence pairs can be assigned to intentions through propagation or user assignments. Stakeholders can assign an evidence pair to any intention in the graph at any time point in the graph time points (see Definition 18). Stakeholders assign known evidence pairs at the current time point \( (t_0) \). Additionally, stakeholders can assign desired values of intentions at other time points to constrain simulation results. However, by doing so, stakeholders run the risk of over-constraining the analysis resulting in no valid path.

Table 3 List of the constraints between time points for each constraint origin in the bike lanes example

<table>
<thead>
<tr>
<th>Row Set 1: Defined-Satisfied function: Have Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_0 &lt; t_{\text{design}} &lt; t_{\text{end}} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Set 2: User-Defined function: Bike Lane Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_0 &lt; t_{\text{use}} &lt; t_{\text{season}} &lt; t_{\text{summer}} &lt; t_{\text{fall}} &lt; t_{\text{spring}} )</td>
</tr>
<tr>
<td>( \leq t_{\text{end}} &lt; t_{\text{season}} &lt; t_{\text{summer}} &lt; t_{\text{fall}} &lt; t_{\text{spring}} &lt; t_{\text{end}} )</td>
</tr>
<tr>
<td>( \leq t_{\text{season}} &lt; t_{\text{summer}} &lt; t_{\text{fall}} &lt; t_{\text{spring}} &lt; t_{\text{end}} &lt; t_{\text{season}} )</td>
</tr>
<tr>
<td>( \leq t_{\text{summer}} &lt; t_{\text{fall}} &lt; t_{\text{spring}} &lt; t_{\text{end}} &lt; t_{\text{season}} )</td>
</tr>
<tr>
<td>( \leq t_{\text{fall}} &lt; t_{\text{spring}} &lt; t_{\text{end}} &lt; t_{\text{season}} )</td>
</tr>
<tr>
<td>( \leq t_{\text{spring}} &lt; t_{\text{end}} &lt; t_{\text{season}} )</td>
</tr>
<tr>
<td>( \leq t_{\text{end}} &lt; t_{\text{season}} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Set 3: ( \neg \text{S/NO} ) relationship between Permanent Construction &amp; Access to Parking</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_0 &lt; t_{\text{BLOpen}} &lt; t_{\text{end}} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Set 4: NBT relationship between Permanent Construction &amp; Temporary Construction</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_0 &lt; t_{\text{design}} &lt; t_{\text{end}} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Set 5: Graph Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{design}} &lt; t_{\text{decision}} )</td>
</tr>
<tr>
<td>( t_{\text{use}} = t_{\text{BLOpen}} = 10, )</td>
</tr>
<tr>
<td>( t_{\text{decision}} = 5, t_{\text{ab}} = 7, t_{a_1} = 14, t_{a_2} = 34 )</td>
</tr>
</tbody>
</table>

In this example, \( maxTime \) is assigned arbitrarily to be 50, so \( t_{\text{end}} = 51 \) (and, as always, \( t_0 = 0 \))

Definition 19 (User evaluation at a time point)
Let a goal graph \( M = (A, G, R, EF, MC, maxTime) \) be given. A set of user evaluations \( UEval = \{ (g, t, e) \mid g \in G \land t \in TP(M) \land e \in E \} \) is the user’s assignment of evidence pairs to intentions at time points.

In the bike lanes example, a set of user evaluations is \( \{ \langle \text{HaveDesign}, t_{a_2}, (F, \bot) \rangle, \langle \text{CyclistSafety}, t_0, (\bot, P) \rangle \} \).

User evaluations can be any combination of intentions and time points. For example, we could include all intentions at \( t_0 \) to assign the current values that the user knows to be true in the real world. The user can also assign desired values (e.g., root-level goals to achieve in the future), such as saying Cyclist Safety will be satisfied at \( t_{a_2} \), as in \( \langle \text{CyclistSafety}, t_{a_2}, (F, \bot) \rangle \). However, this same value can be assigned with an evolving function. For example, assigning Cyclist Safety a Stochastic-Constant that is constant with the evidence pair \( (F, \bot) \) at \( t_{a_2} \) would achieve the same result. It is possible for users to add user affirmed intentions that make the evaluation of the goal graph inconsistent. For example, if we added the user evaluation \( \langle \text{CyclistSafety}, t_{a_2}, (F, \bot) \rangle \) in conjunction with the Stochastic-Constant function, this would result in an inconsistency in the value of Cyclist Safety at \( t_{a_2} \).

Evaluating an evolving goal graph at a time point
With the introduction of user evaluations and graph time points, we can now evaluate a goal graph at a time point. In Sect. 3, we defined a goal graph evaluation (see Definition 6). We extend this definition to add timing information that allows goal graphs to exist at a time point, which defines our notion of a state. A set of goal graph evaluations at time points are connected to create a path that describes the evolution of a goal graph in Sect. 5.
**Definition 20** (Goal Graph Eval. at a Time Point) Let a well-formed goal graph $M = (A, G, R, EF, MC, \text{maxTime})$, a set of user evaluations $UEval$, a reference time point $t_{Ref} \in TP(M)$, and a conflict level $\text{CFL}$ be given. A complete evaluation of $M$ at $t_{Ref}$ is the total mapping $G \times t_{Ref} \rightarrow E$ resulting from the repeated application of Definitions 8–16 to all intentions, assignment of evidence pairs to intentions, and application of the propagation rules in Fig. 6 to all relationships in $R$, given $UEval$ such that $\forall g \in G : \|g[t_{Ref}]\| \notin \text{CFL} \land \|g[t_{Ref}]\| \neq \bot \land (\forall (g, t_{Ref}, e) \in UEval : e = \|g[t_{Ref}]\|)$. When this is not possible, the evaluation of the goal graph is said to be infeasible.

We show the evaluation of the Bike Lanes example at $t_{\text{decision}}$ in Fig. 12 and $t_{\text{End}}$ in Fig. 13. These time points were selected to showcase both a time point where there is a transition in an evolving function (i.e., $t_{\text{decision}}$ is defined in Have Design) and an absolute time point (i.e., $t_{\text{End}}$ is specified in $MC$). For both examples, we did not have user evaluations, and the CFL was set to Weak_Conflict. In Fig. 12,

![Image](image1.png)

**Fig. 12** Evaluation of the Bike Lane example at $t_{\text{decision}} = 5$

![Image](image2.png)

**Fig. 13** Evaluation of the Bike Lane example after $t_{\text{End}} = 14$

**Table 4** Graph evaluation path result for the bike lanes example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>$⊥, F$</td>
<td>$⊥, F$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, F$</td>
</tr>
<tr>
<td>$t_{\text{design}}$</td>
<td>2</td>
<td>$⊥, F$</td>
<td>$F, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
</tr>
<tr>
<td>$t_{\text{decision}}$</td>
<td>5</td>
<td>$⊥, F$</td>
<td>$F, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
</tr>
<tr>
<td>$t_{\text{End}}$</td>
<td>7</td>
<td>$⊥, F$</td>
<td>$F, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
<td>$⊥, ⊥$</td>
</tr>
<tr>
<td>$t_{\text{End}}/\text{Bike Lanes}$</td>
<td>10</td>
<td>$⊥, P$</td>
<td>$(P, ⊥)$</td>
<td>$⊥, P$</td>
<td>$⊥, P$</td>
<td>$⊥, P$</td>
<td>$⊥, P$</td>
<td>$⊥, P$</td>
<td>$⊥, P$</td>
</tr>
<tr>
<td>$r_{1\text{Summer}}$</td>
<td>17</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
</tr>
<tr>
<td>$r_{1\text{Fall}}$</td>
<td>19</td>
<td>$(P, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(F, ⊥)$</td>
</tr>
<tr>
<td>$r_{1\text{End}/r_{2\text{Spring}}}$</td>
<td>24</td>
<td>$(⊥, P)$</td>
<td>$(F, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(P, ⊥)$</td>
</tr>
<tr>
<td>$r_{2\text{Spring}}$</td>
<td>28</td>
<td>$(P, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
</tr>
<tr>
<td>$r_{2\text{Summer}}$</td>
<td>32</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
</tr>
<tr>
<td>$r_{2\text{Fall}}$</td>
<td>34</td>
<td>$(P, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
</tr>
<tr>
<td>$r_{2\text{Fall}}/r_{3\text{Spring}}$</td>
<td>36</td>
<td>$(⊥, P)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
</tr>
<tr>
<td>$r_{3\text{Spring}}$</td>
<td>40</td>
<td>$(⊥, F)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
</tr>
<tr>
<td>$r_{3\text{Fall}}$</td>
<td>45</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(F, ⊥)$</td>
</tr>
<tr>
<td>$r_{3\text{Fall}}/r_{3\text{Spring}}$</td>
<td>48</td>
<td>$(P, ⊥)$</td>
<td>$(F, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(P, ⊥)$</td>
<td>$(P, ⊥)$</td>
</tr>
</tbody>
</table>

Each element of the time point column is associated with a time and an evidence pair for each goal in the goal graph. Notice that there is no row for $t_{\text{End}}/r_{3\text{Spring}}$ because the graph is not evaluated at this time point (see Definition 21)
\( t_{\text{decision}} = 5 \). Have Design’s \text{Denied-Satisfied} function has already transitioned to \((F, \bot)\) because \( t_{\text{design}} < t_{\text{decision}} \), and Bike Lane Usage is \((\bot, F)\) because \( t_{\text{decision}} < t_{\text{use}} \). At \( t_{\text{decision}} \), a decision is made between Permanent Construction and Temporary Construction. In this evaluation, Permanent Construction is chosen and assigned \((F, \bot)\), and Temporary Construction remains \((\bot, \bot)\). The values for the rest of the intentions in the graph are then propagated from these evidence pairs. Access to Parking has the value \((\bot, F)\) because of the \( \rightarrow \text{S} \) relationship with Permanent Construction, which has not evolved yet because \( t_{\text{decision}} \). In this evolution, 

\text{Permanent Construction} \rightarrow \text{Parking from Access to Parking}.

\text{Construction has transitioned to function, and the evolving relationship between \text{Permanent} and \text{Temporary Construction} remains \((\bot, \bot)\). In this evaluation, \text{Access to Parking} is non-deterministic and in this evaluation has the value \((\bot, \bot)\). The remainder of the intentions have the same evidence pairs as in Fig. 12 at \( t_{\text{decision}} \). From these two time points, we see how the evaluation of the goal graph changes between \( t_{\text{decision}} \) and \( t_{\text{use}} \).

Both of the evaluations at \( t_{\text{decision}} \) and \( t_{\text{use}} \) are complete (see Definition 20). Suppose instead we added the user evaluation \( \langle \text{Have Design}, t_{\text{use}}, \bot, P \rangle \). This would result in an infeasible graph evaluation at \( t_{\text{use}} \) because the value of Have Design is over-constrained. The \text{ Denied-Satisfied} function defines the value of Have Design as \((F, \bot)\) after \( t_{\text{design}} \), and \( t_{\text{use}} \) is after \( t_{\text{design}} \), so the user cannot assign the value of Have Design to be \((\bot, P)\). Infeasible graph evaluations help the user understand which evaluations are not possible and debug the graph when errors are made.

Generating simulation paths A state is an evaluation of a goal graph at a given time \( t \). A path is a sequence of states where time is increasing and all time points with the same time have identical evaluations of the goal graph. A path is an ordering of existing constraints over the time points in the graph. A path is only generated once all repeating segments of \text{User-Defined} functions have been unrolled (see Definition 18), because this results in the creation of additional time points.

As stated in Sect. 4.1, in the previous section, we simplified the presentation of timepoints by using the variable and value for a time point interchangeably. In the next definition, we use \( t \) to denote a time point variable (e.g., time points (variable) column in Table 4) and \( \parallel t \parallel \) to denote the evaluation of a time point, which returns an absolute time value in terms of ticks (e.g., Abs. Time (Ticks) column in Table 4).

**Definition 21** (Time point path) Let a goal graph \( M \), and a set of goal graph time points in \( M \) (i.e., \( TP(M) \)) be given. A time point path \( \Pi \) is a partial order over \( TP(M) \) satisfying the graph constraints in \( M \), where \( \Pi = \{ (t, \parallel t \parallel) \mid t \in TP(M) \land \parallel t \parallel \neq \bot \land 0 \leq \parallel t \parallel \leq \maxTime \} \).

A time point path is created by assigning an absolute value to each time point in the goal graph \( TP(M) \). For example, we can assign an absolute time to each of the time points listed in Table 3 for the bike lanes example. A time point path is shown in the first two columns in Table 4. Other time point paths can exist by assigning different absolute values to each time point.

Recall that \( t_{\text{end}} = \maxTime + 1 \) is included in \( TP(M) \) with the rationale that we defined intervals over time points that are open on the right, and we want to make sure that \( \maxTime \) can be included (see Sect. 4.1). However, the time point path explicitly does not include \( t_{\text{end}} \) because it is greater than \( \maxTime \).

With a time point path, we can evaluate the graph over each of the time points in the path to create a simulation of the evolution of the graph.

**Definition 22** (Graph evaluation path) Let a goal graph \( M \), a time point path \( \Pi \), a set of user evaluations \( UEval \), and a conflict level \( CFLevel \) be given. A complete graph evaluation path is the complete evaluation of \( M \) at each time point in \( \Pi \), given \( UEval \) and \( CFLevel \).

Formally, \( \forall t \in \Pi : G[t] \rightarrow E \) such that \( \forall g \in G : \parallel g[t] \parallel \notin CFLevel \land \parallel g[t] \parallel \neq \bot \) and the rules in Fig. 6 and Definitions 8–16 are respected.

Once computed, the graph evaluation path consists of two parts: first, the time point path \( \Pi \) with each time point having an absolute value assigned to it; and second, the set of intentions in the graph have an assigned evidence pair for each time in the time point path (i.e., for each time point in \( \Pi \)).
Algorithm 1 Compute Evolving Graph Evaluation Path

Require:

Goal Graph $\langle A, G, R, EF, MC, maxTime \rangle$ \Comment{Satisfying Def. 17.}
Time Point Path $\Pi$ \Comment{Ordered set of times satisfying Def. 21.}
Set of User Evaluations $UEval$ \Comment{Satisfying Def. 19.}
Set of Conflicting Evidence Pairs $CFEvidence$ \Comment{Satisfying Def. 4.}

Ensure:

Evaluations of the graph intentions (satisfying Def. 20) at each time point in $\Pi$
or null indicating infeasibility.

1: $aGoals = \emptyset$ \Comment{Set of assigned intentions and evidence pairs. Type: $\langle g, t, [g[t]] \rangle$.}
2: $uGoals = \emptyset$ \Comment{Set of unassigned intentions and possible evidence pairs. Type: $\langle g, t, \emptyset \rangle$.}
3: for all $t \in \Pi, g \in G$ do
4: if $(g, t) \in UEval$ then
5: if $UEval(g, t) \in CFEvidence$ then return null
6: $aGoals.add((g, t, UEval(g, t)))$
7: else
8: $uGoals.add((g, t, \emptyset \setminus CFEvidence))$ \Comment{Adds the set of non-conflicting evidence pairs.}
9: if $\neg(ASSIGN_CONSTANT(EF, \Pi, aGoals, uGoals))$ then return null \Comment{See Algo. 4 in App. A.}
10: $uGoals.sort(\Pi)$ \Comment{Sorts the unassigned values by time point.}
11: return $SOLVE(aGoals, uGoals, R, EF, \Pi)$ \Comment{Recursive procedure to pick values and check.}
12: function $SOLVE(aGoals, uGoals, R, EF, \Pi)$
13: if $uGoals = \emptyset$ then
14: return $aGoals$
15: oldAGoals = $aGoals$, oldUGoals = $uGoals$ \Comment{Create copies for backtracking.}
16: $curGoal = uGoals.head.goal$
17: $curTime = uGoals.head.time$
18: $curEvals = uGoals.head.evals$
19: for all $val \in curEvals$ do \Comment{Iterate over possible evaluations.}
20: $aGoals.add(curGoal, curTime, val)$ \Comment{Make assignments.}
21: $uGoals.remove(curGoal, curTime)$
22: $(aGoals, uGoals) = PROPAGATE(G, R, aGoals, uGoals, t)$
23: validGraph = CHECK_PROP($G, R, II, aGoals$) \wedge CHECK_ELFUNC($EF, \Pi, aGoals$) \wedge CHECK_NOTBoth($R, II, aGoals$) \Comment{See Algo. 3, 5, and 6 in App. A.}
24: if validGraph then
25: result = $SOLVE(aGoals, uGoals, R, EF, \Pi)$ \Comment{Call $SOLVE$ to select the next assignment.}
26: if result $\neq$ null then return result \Comment{Return successful result.}
27: $aGoals = oldAGoals$
28: $uGoals = oldUGoals$
29: return null \Comment{If no evaluations remain, the graph is infeasible.}

Graph evaluation path Algorithm (Algorithm 1). We present Algorithm 1, a reference algorithm that finds a complete Graph Evaluation Path (satisfying Definition 22) if one exists. Algorithm 1 is a brute force algorithm that returns a complete graph evaluation path (satisfying Definition 22) or null indicating infeasibility. A complete graph evaluation path is a complete graph evaluation at each time point in the given time point path. Algorithm 1 can be used to answer stakeholder questions by generating simulations showing the evaluations of an evolving goal graph over time.

Algorithm 1 has four inputs: an evolving goal graph (satisfying Definition 17), a time point path (satisfying Definition 21), a set of user evaluations (satisfying Definition 19), and a set of conflicting evidence pairs (satisfying Definition 4). Algorithm 1 consists of two parts: initialization of the inputs (see Lines 1–10) and the $SOLVE$ function (see Lines 12–29). $SOLVE$ is a recursive function that
incrementally assigns evidence pairs to intentions at time points. When \texttt{Solve} is first called on Line 11, it contains only the user evaluations in \texttt{aGoals} and assignments made by the \texttt{Assign CONSTANT} function, and the remainder of the intentions in \texttt{G} are stored in \texttt{uGoals} with non-conflicting evidence pairs. On Line 11, the \texttt{Assign CONSTANT} function (see Algorithm 4 in App.) assigns evidence pairs to each intention defined by a \texttt{CONSTANT} function over specific time points in \texttt{G}.

For each iteration of the main loop in the \texttt{Solve} function (see Lines 19–28), \texttt{Solve} considers one additional assigned evidence pair. The new assignment, \texttt{val} for \texttt{curGoal} at \texttt{curTime}, is added to \texttt{aGoals} and removed from \texttt{uGoals}. Then, the algorithm attempts to assign additional evidence pairs to intentions by calling the \texttt{Propagate} function on Line 22 (see Algorithm 2 in App. A.1 for details). Next, on Line 23 the algorithm checks the constraints over the graph elements by invoking the \texttt{Check Prop} function (see Algorithm 3 in Appendix "A.1") to see if any relationships were violated, followed by the \texttt{Check EL Func} and \texttt{Check Not Both} (see Algorithm 5 and Algorithm 6 in App.) functions to determine if the assigned evidence pair violates any evolving functions. If any of these checks return \texttt{false} (meaning the graph is inconsistent and \texttt{validGraph = false}), then the algorithm jumps to Line 27 to backtracking. If the checks all return \texttt{true}, then a recursive call to \texttt{Solve} is made on Line 25 with the additional assignment. If the call to \texttt{Solve} returns \texttt{null}, then backtracking is required; otherwise, the call was successful and a set of \texttt{aGoals} is returned on Line 26. \texttt{Solve} backtracks by reverting \texttt{aGoals} and \texttt{uGoals} back to their old values and begins the next iteration of the main loop, where the next \texttt{val} is selected. If the main loop completes and all elements in \texttt{curEval} had been tested, then no solution exists, and \texttt{Solve} returns \texttt{null} indicating infeasibility. \texttt{Solve} exhaustively checks every evidence pair for every unassigned intention at every time point, in the case of infeasibility.

\textit{Correctness and termination of Algorithm 1.} Next, we prove properties of Algorithm 1.

\textbf{Theorem 1 (Termination)} \texttt{Algorithm 1 terminates.}

\textbf{Proof} Let \(k\) be the size of \texttt{uGoals} at Line 11 in Algorithm 1. The maximum number of possible evidence pairs is \(|E| = 9\) for an intention at a given time, if \texttt{UEval} and \texttt{C FEvidence} are the empty set. So each \(k\) has at most nine possible evidence pair assignments. Overall, \texttt{Solve} works by picking evidence pair assignments for intentions and backtracking when an assignment violates one or more of the constraints in the model. The recursive \texttt{Solve} procedure selects the tuple at the head of \texttt{uGoals} and then begins to iterate over possible evidence (\texttt{curEval}) for the selected intention (\texttt{curGoal}) at Line 19. If the selected assignment (\texttt{val}) and propagated values do not violate any of the checks on Line 23, then \texttt{Solve} is called again with the assignment held and a new head for \texttt{uGoals} on Line 25. If the assignment is not valid, then \texttt{Solve} will select the next evidence pair in \texttt{curEval} (Line 19) and repeat. If none of the values in \texttt{curEval} are found to be valid, then there is no valid solution, and \texttt{Solve} returns \texttt{null}. In the worst case, no additional values are found through the \texttt{Propagate} and \texttt{Assign CONSTANT} procedures, and \texttt{Solve} will try every combination of evidence pairs for every unassigned value, and will change the graph assignment \(9^n\) times (where \(n = |G|\) and \(m = |\Pi|\)) before returning \texttt{null}.

Therefore, Algorithm 1 is guaranteed to terminate with a runtime of \(\mathcal{O}(9^n)\).

\textbf{Theorem 2 (Completeness given Complete Path)} \texttt{Algorithm 1 will result in a complete graph evaluation path if a complete path exists.}

\textbf{Proof} Proof is by contradiction: suppose a complete graph evaluation path exists and Algorithm 1 returns \texttt{null} indicating that the path is infeasible. The complete assignment is \((\{g_0, t_0, (F, \bot)\}, \ldots, \{g_m, t_m, (F, \bot)\})\). Since all user evaluations are known, and the \texttt{Solve} procedure tries every combination of evidence pairs for unassigned intentions, Algorithm 1 would have tried every combination, including this combination, which is a contradiction. Therefore, Algorithm 1 will result in a complete graph evaluation path if a complete path exists.

\textbf{Theorem 3 (Soundness)} \texttt{If Algorithm 1 results in a complete graph evaluation path then the path satisfies Definition 22.}

\textbf{Proof} Definition 22 states that a complete graph evaluation path is the complete evaluation of the graph at each time point in the time point path, given the user evaluations and CFLevel. Definition 20 states that a complete evaluation of the graph at a time point is correct if all the user evaluations (at the time point) have their given values and all relationships satisfy the propagation rules in Fig. 6 and the evolving functions (see Definition 8–16) such that \(\forall g \in G : [g] \not\in \text{CFLevel} \land [g] \neq \bot\).

If Algorithm 1 results in a complete evaluation of the goal graph, the final call to \texttt{Check Prop}, \texttt{Check EL Func}, and \texttt{Check Not Both} would have returned \texttt{true} so we know that the evaluation satisfied the rules in Fig. 6 for all relationships in \(R\), and Definition 8–16 for each time point. Given the user evaluations, the assignment of only non-conflicting evidence to unassigned intentions (see Lines 3–8 in Algorithm 1) satisfies the requirement that \(\forall t \in P : G[t] \rightarrow E\) such that \(\forall g \in G : [g[t]] \not\in \text{CFLevel} \land [g[t]] \neq \bot\). Therefore, the complete evaluation of the goal graph is correct by construction.
The Bike Lanes Goal Graph is \( \langle A, G, R, EF, MC, \text{maxTime} \rangle \) where,

\[
A = \{ \},
\]

\[
G = \{ \langle \text{Have Bike Lanes, goal} \rangle, \langle \text{Have Design, goal} \rangle, \langle \text{Build Bike Lanes, goal} \rangle, \langle \text{Plan Build, goal} \rangle,
\langle \text{Temporary Construction, goal} \rangle, \langle \text{Permanent Construction, goal} \rangle, \langle \text{Commercial Access, goal} \rangle,
\langle \text{Cyclist Safety, goal} \rangle, \langle \text{Access to Parking, goal} \rangle, \langle \text{Bike Lane Usage, goal} \rangle, \langle \text{Public Support, goal} \rangle,
\langle \text{Road Maintenance, goal} \rangle, \langle \text{Operate Bike Lane, goal} \rangle, \langle \text{Remove Bollards, goal} \rangle, \langle \text{No Parking, goal} \rangle,
\langle \text{Separate Snow Removal, goal} \rangle, \langle \text{Costs Justification, goal} \rangle, \langle \text{Construct Medians, goal} \rangle,
\langle \text{Install Bollards, goal} \rangle, \langle \text{Paint Lines, goal} \rangle, \langle \text{Road Restructuring, goal} \rangle, \langle \text{Off Street Parking, goal} \rangle,
\langle \text{Temporary Construction Plan, goal} \rangle, \langle \text{Permanent Construction Plan, goal} \rangle, \langle \text{Parking, goal} \rangle,
\langle \text{Prevent Doorico Incident, goal} \rangle, \langle \text{Prevent Unloading in Bike Lane, goal} \rangle,
\langle \text{Bike Lane Curbside, goal} \rangle, \langle \text{Parking Curbside, goal} \rangle \},
\]

\[
R = \{ \langle \text{Have Design, Build Bike Lanes, Plan Build, Operate Bike Lane} \rangle \xrightarrow{\text{and}} \langle \text{Have Bike Lanes} \rangle,
\langle \text{Temporary Construction, Permanent Construction} \rangle \xrightarrow{\text{or}} \langle \text{Build Bike Lanes} \rangle,
\langle \text{Temporary Construction Plan, Permanent Construction Plan} \rangle \xrightarrow{\text{or}} \langle \text{Plan Build} \rangle,
\langle \text{Remove Bollards, Separate Snow Removal} \rangle \xrightarrow{\text{or}} \langle \text{Road Maintenance} \rangle,
\langle \text{Bike Lane Curbside, Parking Curbside} \rangle \xrightarrow{\text{or}} \langle \text{Parking, No Parking} \rangle \xrightarrow{\text{or}} \langle \text{Have Design} \rangle,
\langle \text{Paint Lines, Install Bollards} \rangle \xrightarrow{\text{and}} \langle \text{Temporary Construction} \rangle,
\langle \text{Paint Lines, Road Restructuring, Construct Medians} \rangle \xrightarrow{\text{and}} \langle \text{Permanent Construction} \rangle,
\langle \text{Have Bike Lanes} \rangle \xrightarrow{+s} \langle \text{Cyclist Safety, Separate Snow Removal} \rangle \xrightarrow{-s} \langle \text{Costs Justification} \rangle,
\langle \text{Remove Bollards} \rangle \xrightarrow{-s} \langle \text{Costs Justification} \rangle, \langle \text{Prevent Unloading in Bike Lane} \rangle \xrightarrow{+} \langle \text{Cyclist Safety} \rangle,
\langle \text{Bike Lane Curbside} \rangle \xrightarrow{+} \langle \text{Cyclist Safety, Prevent Doorico Incident} \rangle \xrightarrow{+} \langle \text{Cyclist Safety} \rangle,
\langle \text{Parking Curbside} \rangle \xrightarrow{+} \langle \text{Prevent Unloading in Bike Lane, No Parking} \rangle \xrightarrow{-s} \langle \text{Public Support} \rangle,
\langle \text{No Parking} \rangle \xrightarrow{-s} \langle \text{Access to Parking, Off Street Parking} \rangle \xrightarrow{+s} \langle \text{Access to Parking} \rangle,
\langle \text{Access to Parking} \rangle \xrightarrow{+} \langle \text{Public Support, Commercial Access} \rangle \xrightarrow{+} \langle \text{Public Support} \rangle,
\langle \text{Bike Lane Usage} \rangle \xrightarrow{+} \langle \text{Public Support, Bike Lane Usage} \rangle \xrightarrow{+} \langle \text{Costs Justification} \rangle,
\langle \text{Temporary Construction Plan} \rangle \xrightarrow{-s} \langle \text{Costs Justification} \rangle,
\langle \text{Permanent Construction Plan} \rangle \xrightarrow{-s/\text{no}} \langle \text{Costs Justification, Road Maintenance} \rangle \xrightarrow{+} \langle \text{Operate Bike Lane} \rangle,
\langle \text{Permanent Construction} \rangle \xrightarrow{+s/\text{no}} \langle \text{Access to Parking} \rangle,
\langle \text{Bike Lane Curbside} \rangle \xrightarrow{\text{NBN}} \langle \text{Parking Curbside} \rangle,
\langle \text{Temporary Construction Plan} \rangle \xrightarrow{\text{NBD}} \langle \text{Permanent Construction Plan} \rangle,
\langle \text{Temporary Construction Plan} \rangle \xrightarrow{+s} \langle \text{Temporary Construction} \rangle,
\langle \text{Permanent Construction Plan} \rangle \xrightarrow{+s} \langle \text{Permanent Construction} \rangle \},
\]

\[
\text{maxTime} = 50.
\]

Fig. 14 Specification of the full bike lanes example goal graph with evolving elements

Illustration of Algorithm 1 with BLE

We conclude this section by briefly illustrating the graph evaluation path for the bike lanes example graph fragment specified in Fig. 11, as created by Algorithm 1. First, we find a time point path given all the time points from the graph. The repeating USER-DEFINED function for Bike Lane Usage has already been unrolled, giving the partial order

\[
t_{\text{use}} < r_1 \rightarrow s_{\text{spring}} < r_1 \rightarrow s_{\text{summer}} < r_1 \rightarrow s_{\text{fall}} <
\]
\[
(r_1 \rightarrow e_{\text{end}} = r_2 \rightarrow s_{\text{use}} < r_2 \rightarrow s_{\text{spring}} < r_2 \rightarrow s_{\text{summer}} <
\]
\[
r_2 \rightarrow f_{\text{fall}} < (r_2 \rightarrow e_{\text{end}} = r_3 \rightarrow s_{\text{use}} < r_3 \rightarrow s_{\text{spring}} <
\]
\[
r_3 \rightarrow s_{\text{summer}} < r_3 \rightarrow f_{\text{fall}} < (r_3 \rightarrow e_{\text{end}} = t_{\text{end}}).
\]

The absolute value assignments to time points (see Table 1) also give two partial orders:
In the BLE, we let the algorithm choose between trade-offs. Using the NorBoth function, we identify mutually exclusive decisions and can ask if these decision points can be delayed: "Can the City delay the design and planning decisions to the future?"

The BLE has two mutually exclusive decisions: Temporary Construction Plan vs. Permanent Construction Plan for which we assigned an NBN function, and Bike Lane Curbside vs. Parking Curbside for which we assigned an NBD function (see Fig. 14). By including these relationships in the goal graph specification, each generated simulation path will defer each decision and choose an alternative in the NorBoth relationship (see Definition 15); thus, we can delay decision making until the transitions in the NorBoth functions, answering EQ1 positively.

### 6.2 EQ2

What scenario ensures Access to Parking is satisfied in six months, even if off-street lots become unavailable? The phrase ‘ensures in six months’ means that Access to Parking must be satisfied at 6 months. Recall from Sect. 4.1 that the stakeholders chose to represent one month of real world time as a tick. We assign Access to Parking a Stochastic-Constant function \( \langle \text{AccessstoParking}, \langle \text{STOCHASTIC-CONSTANT}, FS, \{t_0, t_{acs}, t_{end}\} \rangle \rangle \), with the constraint \( t_{acs} = 6 \), representing six months. We give the values for \( EF \) and \( MC \) required to answer EQ2 in Fig. 15.

This returns a result where the Temporary Construction Plan is chosen, because otherwise the construction will not be completed in six months given the constraint that \( t_{plan} = 5 \) (i.e., the time when a construction plan is chosen). This result helped us debug the model and realize that both the absolute values selected for Bike Lane Usage and the constraints within \( MC \) contained unrealistic values. To answer EQ2 and ensure Access to Parking is satisfied in 6 months, the City must either select Temporary Construction Plan or make guarantees that permanent construction will be complete in 6 months.

### 6.3 EQ3

Looking more broadly at the differences between temporary and permanent construction, we ask “Is a permanent solution or a temporary solution most appropriate, given
possible changes to cyclist and motor vehicle traffic?” We used the evolving functions $E^F$ listed above but removed all the absolute constraints in $MC$.

In a simulation result for this question, we found a missing link between Install Bollards and Remove Bollards, as well as between Construct Medians and Separate Snow Removal. Once these links have been added, the simulation indicated that Permanent Construction and permanent construction maintenance (via Separate Snow Removal) have a negative impact on Cost Justification over the winter period when Bike Lane Usage is unknown (i.e., Stochastic). In the end, the City decides that Permanent Construction is deemed to be the better construction plan.

### 6.4 EQ4

Another tradeoff is considered with the question “Are bike lanes or parking most effective on curbside, given seasonal constraints on road operations?” Road operations are represented by the node Road Maintenance in this model (see Fig. and specification in Fig.); so, we investigate how Road Maintenance is impacted by this trade-off. Given the NorBot (None) evolving function between Bike Lane Curbside and Parking Curbside, we can investigate this question with the same simulation results as EQ2 and EQ3. This simulation does not produce interesting results because we did not add any evolving functions for the elements that impact Road Maintenance in connection with Bike Lane Curbside or Parking Curbside. This is an example of how the analysis helps stakeholders to debug their model and ensure that important concepts are accurately represented.

### 6.5 EQ5

**How do variations in the satisfaction of Bike Lane Usage over time affect the City’s goals?** We have already discussed the result of having Bike Lane Usage change with the season above, so here we change the evolving function for Bike Lane Usage to be stochastic over the full simulation, $\langle\text{BikeLaneUsage}, \{\text{STOCHASTIC}, \bot, t_0, t_{\text{end}}\}\rangle$. We observe Bike Lane Usage changing stochastically in the resulting simulation path, which impacts Costs Justification and Public Support. In this goal graph, Cyclist Safety is not impacted by Bike Lane Usage. With this information, the City reevaluates their root-level goals. Since Cyclist Safety is paramount, the City needs to find a solution where Cyclist Safety can be satisfied and maintained, and then explore the values for Costs Justification and Public Support.

### 6.6 EQ6

**Can we eventually satisfy and maintain Cyclist Safety?** The phrase ‘eventually satisfy and maintain’ is modeled with a Stochastic-Constant function. To do this, we add $\langle\text{CyclistSafety}, \{\text{STOCHASTIC-CONSTANT, FS, } t_0, t_{\text{safe}}, t_{\text{end}}\}\rangle$ to the set $E^F$ in the goal graph. Using this addition, the generated solution has conflicting values for Cyclist Safety until $t_{\text{safe}}$ then becomes satisfied. Next, to avoid conflicting values, we model Cyclist Safety, using the function $\langle\text{CyclistSafety}, \{\text{INCREASE, FS, } t_0, t_{\text{end}}\}\rangle$, and add the user evaluations $UEval = \{\langle\text{CyclistSafety, } t_0, PS\rangle, \langle\text{CyclistSafety, } t_{\text{safe}}, FS\rangle\}$. This second function ensures that Cyclist Safety is always at least $PS$, but the model does not result in a complete solution because it cannot be partially satisfied at $t = 0$. By removing the user evaluation at $t_0$, we satisfy and maintain Cyclist Safety, but only if the bike lanes are built.

### 6.7 Summary

By answering EQ1–EQ6, we explored the impact of changes in the evaluation of intentions in the BLE. We concluded that the city should build the bike lane curbside with a temporary construction plan.

### 7 Implementation and validation

The formalism introduced earlier in this article enables automated analysis. Stakeholders do not directly interact with the definitions; instead, stakeholders create models using the visual syntax (as described in Sects. 1 and 2) and use tooling to perform analysis built on top of the formalism. In this section, we first describe our implementation and then validate the effectiveness (see Sect. 7.2) and scalability (see Sect. 7.3) of the Evolving Intentions framework.

#### 7.1 Tooling and encoding

Our formalism is implemented as a browser-based modeling tool, BloomingLeaf [25]. It allows stakeholders to create goal graphs, add evolutionary information, and generate simulations (satisfying Definition 22). Since Algorithm 1 (see Sect. 5) is a brute force reference algorithm and is not efficient at finding paths, BloomingLeaf encodes graph evaluation paths as a constraint satisfaction problem (CSP) [47].

**Definition 23** (Constraint satisfaction problem) A constraint satisfaction problem is defined as a triple $(V, D, Q)$, where $V = \{v_1, \ldots, v_n\}$ is a set of variables, $D = \{d_1, \ldots, d_n\}$ is a set of the respective domains of values, and $Q = \{q_1, \ldots, q_m\}$ is a set of constraints [47].
Recall that a graph evaluation path is the complete evaluation of a goal graph $M = (A, G, R, EF, MC, maxTime)$ at each time point in $\Pi$, given $UEval$ and $CFLevel$ (see Definition 22). When constructed as CSP, our goal is to find an evaluation for each intention in the graph ($g \in G$) at each time point ($t \in \Pi$). Thus, CSP variables $V$ are a set of intentions at all time points (i.e., $|V| = |G| = |\Pi|$). Initially, the domain $d$ of each variable $v$ is the set of evidence pairs ($\mathcal{E}$). Each user ($g, t, e$) (see Definition 19) reduces the domain for the variable ($g, t$) to a domain of $e$. Each element in $R$ and $EF$ forms a constraint $q$ over the elements in $V$. Once all the constraints are added to the problem, BloomingLeaf invokes JaCoP constraint solver [36] to find satisfying assignments to each variable, such that each variable assignment is an element of the variable’s domain and all the assignments taken together satisfy all constraints in the problem [47]. See the associated thesis for the complete encoding [22].

### 7.2 Effectiveness: the Spadina Expressway study

In this subsection, we validate the analysis capabilities of the Evolving Intentions framework. We demonstrated the applicability of our framework via a case study of the evolution of the Spadina Expressway project [26]. We now focus on effectiveness and scalability, by determining the extent to which our formalization and analysis can answer questions over real-world complex models. Using the online supplement from [26], we created a version of the Spadina Expressway model (known as the Spadina graph in this paper) and created the formal specification of the evolving goal graph $M = (A, G, R, EF, MC, maxTime)$ (see Definition 17). The graph consists of eleven actors, 124 intentions, and 161 relationships (i.e., $|A| = 11$, $|G| = 124$, $|R| = 161$, $|EF| = 58$, $|MC| = 58$, $maxTime = 76$). The $maxTime$ of 76 maps onto the period July 1947 to July 1985, with a $tick$ measuring a six-month period [26].

**Reasoning questions** To evaluate the reasoning capabilities of the Evolving Intentions framework, we explore the following questions over the Spadina graph:

- **VQ1** Can we generate the project timeline?
- **VQ2** How will road construction affect property values in York?
- **VQ3** Over what period did the project receive funding?
- **VQ4** Is it possible to fulfill Get Network Funding at time point 30 (i.e., July 1962), given the funding information for the Spadina Expressway project discussed in the previous question?

The questions were chosen to showcase various aspects of the formalism and Algorithm 1.

<table>
<thead>
<tr>
<th>Time point</th>
<th>34</th>
<th>37</th>
<th>57</th>
<th>58</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintain Property Values</td>
<td>$(F, \perp)$</td>
<td>$(\perp, P)$</td>
<td>$\cdots$</td>
<td>$(\perp, P)$</td>
</tr>
<tr>
<td>Build Lawrence to Eglinton</td>
<td>$(\perp, P)$</td>
<td>$(P, \perp)$</td>
<td>$\cdots$</td>
<td>$(P, \perp)$</td>
</tr>
<tr>
<td>Build Eglinton to Davenport</td>
<td>$(L, F)$</td>
<td>$(L, F)$</td>
<td>$\cdots$</td>
<td>$(L, F)$</td>
</tr>
</tbody>
</table>

**Table 5 Simulation results for VQ2**

Prior to the first call to Solve, Oppose Crosstown and Elect Reformer Council were assigned the values $(F, \perp)$ and $(\perp, F)$, respectively, at time point 0. When Propagate (Line 22, see Algorithm 2 in App. A.1 for function details) is called on the first iteration of Solve, $(F, P)$ is assigned to Support Metro as a result of the propagation rule in Fig. 6. Note this is a conflicting value but it does not violate our condition to avoid Strong_Conflict $(F, F)$. After additional elements are added to $uGoals$, the constraints in the graph are checked on Line 23 to determine if valid assignments were made. If so, then Solve is recursively called on Line 25. Thus, Solve is called for each unassigned element at time-point 0. In the same manner, the algorithm iterates through
each unassigned value, in the order of timepoints, to create the path. When the last value is assigned, uGoals is empty and Solve returns the set of aGoals, which is the evolving graph evaluation path. Given the full historical timing information specified in this example, Algorithm 1 is able to produce a path that approximates the historical timeline of events in the Spadina Expressway project. Thus, we were able to generate the project timeline. We answer the remaining questions with respect to this timeline. We create alternative timelines by relaxing constraints in the model and removing evolutionary information from select intentions.

VQ2 property values Next, we look at an example with backtracking based on violations of relationships and evolving functions. To do this, we look at the question: How will road construction affect property values in York? In this graph, the goal to maintain property values is defined as whether property values maintain at least medium market growth relative to the rest of the Toronto housing market.

Again, we discuss answering this question in the context of Algorithm 1. To consider this question, we look at the relationships and functions involved:

Build Lawrence to Eglinton \overset{\text{Build Eglinton to Davenport}}{\longrightarrow} \text{Maintain Property Values},

Build Lawrence to Eglinton and Build Eglinton to Davenport are both User-Defined functions that consist only of Constant functions.

The function that describes Build Lawrence to Eglinton is

\[
\langle \text{Maintain Property Values}, 37, \{ (P, \bot), (P, P) \} \rangle,
\]

and we assign the first value \((F, \bot)\) to val (see Line 19). Build Lawrence to Eglinton and Build Eglinton to Davenport have the values \((P, \bot)\) and \((\bot, F)\), respectively (see Table 5). Check Prop (Line 23) returns false because assigning \((F, \bot)\) to Maintain Property Values violates the \(-S\) relationship from Build Lawrence to Eglinton (see Lines 13–14 of Fig. 6), which has the value \((P, \bot)\). At this point, the algorithm backtracks (Lines 27–28) and tries the next value in val on Line 19, which is \((P, \bot)\). Again, \((P, \bot)\) results in a violation of the \(-S\) relationship when the algorithm calls Check Prop (Line 23), resulting in backtracking and selection of the next value in val on Line 19. At this point, \((\bot, P)\) is selected for Maintain Property Values at time point 37, which is a valid evidence pair and which does not violate any relationships.

<table>
<thead>
<tr>
<th>Absolute time point</th>
<th>0</th>
<th>1</th>
<th>27</th>
<th>29</th>
<th>30</th>
<th>47</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get funding</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(F, \bot)</td>
<td>(F, \bot)</td>
<td>(\bot, F)</td>
</tr>
<tr>
<td>Get funding toronto-york</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
</tr>
<tr>
<td>Get funding from toronto</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
</tr>
<tr>
<td>Get funding from york</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
</tr>
<tr>
<td>Get funding metro-province</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
</tr>
<tr>
<td>Get funding from province</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
</tr>
<tr>
<td>Metro approve funding</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
</tr>
<tr>
<td>OMB approval</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
<td>(\bot, F)</td>
</tr>
</tbody>
</table>
At each time point from 37 up to and including time point 57, Algorithm 1 repeats this process to assign \((\perp, P)\) to maintain property values. Finally, we consider time point 58, where the head of \(uGoals\) is
\[
\langle \text{MaintainPropertyValues}, 58, \{(F, \perp) \ldots (P, P)\}\rangle,
\]
and assign the first value \((F, \perp)\) to \(val\) (see Line 19). Since \text{Build Lawrence to Eglinton} is now assigned the evidence pair \((\perp, F)\) (see Table 5), \((F, \perp)\) does not violate the \(-S\) relationship when the algorithm calls \text{CHECK_PROP}, and \((F, \perp)\) is selected.

Considering the results of Algorithm 1 for this question, if road construction takes place (i.e., is either partially or totally executed), this causes denial of maintain property values, that is, we expect property values to drop relative to the Toronto market. Since the original model from [26] considered maintain property values relative to market rates, we cannot explicitly say to what extent the property values would change or the direction of such a change; but our analysis was able to predict the stability of maintain property values.

VQ3 funding over time Next, we consider the question, “Over what period did the project receive funding?” In this case, we look at the results of simulation for \text{Get Funding} in the Spadina actor. The relationships of interest in this question are described below. The first one is
\[
\langle \text{Get Funding Toronto-York}, \text{Get Funding Metro-Province} \rangle \rightarrow \neg \rightarrow \text{Get Funding}.
\]
\text{Get Funding Toronto-York} consists of the relationship
\[
\langle \text{Get Funding From Toronto}, \text{Get Funding From York} \rangle \rightarrow \neg \rightarrow \text{Get FundingToronto} \rightarrow \neg \rightarrow \text{York},
\]
while \text{Get Funding Metro-Province} is composed of
\[
\langle \text{Get Funding From Metro}, \text{Get Funding From York} \rangle \rightarrow \neg \rightarrow \text{Get FundingMetro} \rightarrow \neg \rightarrow \text{Province}, \text{as well as}
\langle \text{Metro Approves Funding}, \text{OMB Approval} \rangle \rightarrow \neg \rightarrow \text{Get FundingFromMetro}.
\]
For demonstration purposes, we simplify the decomposition of \text{Get Funding Metro-Province} relationships as
\[
\langle \text{GetFundingFromProvince, MetroApprovesFunding, OMB Approval} \rangle \rightarrow \neg \rightarrow \text{Get Funding Metro-Province}.
\]
Each of the child intentions is assigned an evolving function. \text{Get Funding From York} is denied \((\perp, F)\) over the whole period via a \text{Constant} function, while \text{Get Funding From Toronto} is fulfilled \((F, \perp)\) at time point 1 and remains fulfilled via its \text{Constant} function over the remainder of the path. The function that describes \text{Get Funding From Province} is
\[
\{\langle \text{Constant}, (\perp, \perp), t_0, t_a \rangle, \langle \text{Constant}, (F, \perp), t_a, t_b \rangle, \langle \text{Constant}, (\perp, F), t_b, t_{end} \rangle\}.
\]
The function for \text{Metro Approves Funding} is
\[
\{\langle \text{Constant}, (\perp, \perp), t_0, t_a \rangle, \langle \text{Constant}, (\perp, F), t_a, t_b \rangle, \langle \text{Constant}, (F, \perp), t_b, t_{end} \rangle\},
\]
and the function for \text{OMB Approval} is specified as
\[
\{\langle \text{Constant}, (\perp, \perp), t_0, t_a \rangle, \langle \text{Stochastic}, (\perp, \perp), t_a, t_b \rangle, \langle \text{Constant}, (F, \perp), t_b, t_{end} \rangle\}.
\]
The absolute values for these are provided in the model constraints \(MC\), which includes \(t_a = 27, t_b = 30, t_m = 29, t_s = 48\).

With these relationships and evolving functions, we use Algorithm 1 to find a path that specifies the period over which the project received funding. Algorithm 1 iterates over each time point assigning the first element in \(curEval\) to \(val\) and then calls \text{PROPAGATE} (see Line 22). A value for \text{Get Funding} is found on the first call. For example, at time point 1, every child of \text{Get Funding Metro-Province} has the evidence pair \((\perp, \perp), which results in the propagation of \((\perp, \perp)\) to \text{Get Funding Metro-Province}. \text{Get Funding From Toronto} is \((F, \perp)\) and \text{Get Funding From York} is \((\perp, F)\), so \((\perp, F)\) is assigned to \text{Get Funding Toronto-York} via the \text{PROPAGATE} function. These values are then propagated to \text{Get Funding} which is assigned \((\perp, F)\).

The exception to the computation described in the previous paragraph is the time points in the interval \([t_a = 27, t_b = 30]\) because the function assigned to \text{OMB Approval} is \text{Stochastic}, meaning that it was not assigned by \text{ASSIGN_CONSTANT} prior to the first call to \text{SOLVE}. In this case, values are not assigned to \text{Get Funding Metro-Province} and \text{Get Funding} until \text{OMB Approval} is at the head of \(uGoals\), as in
\[
\langle \text{OMB Approval}, 27, \{(F, \perp) \ldots (P, P)\}\rangle
\]
for time point 27. In the original graph [26], \text{OMB Approval} is not a leaf intention and has two children, which constrains the evaluation over the interval \([27, 30]\). For this reason, values are tried until an evidence pair is selected \((\perp, F)\) in the case of time point 27.

Table 6 lists the results of a single simulation path for the intentions that make up \text{Get Funding}. We only list the timepoints where the value of one of the selected intentions change. From Table 6, we see that \text{Get Funding} is fulfilled \((F, \perp)\) from 30 to 47 in absolute timepoints (i.e., the period of 1962–1971). Thus, \text{Get Funding} is fulfilled between timepoints 30 and 47.

VQ4 Network Funding. Finally, we ask the question, “Is it possible to fulfill \text{Get Network Funding} at time point 30 (i.e., July 1962), given the funding information for the Spadina Expressway project discussed in the previous question?”
Algorithm 1 finds the first available path given the inputs but more than one path may exist. We can search for an alternative path by updating one or more of the constraints in the graph. To answer this question, we update Get Network Funding with the following evolving function:

\[
\{(\text{STOCHASTIC}, (\bot, \bot), t_0, t_f), \\
\quad (\text{CONSTANT}, (F, \bot), t_f, t_{\text{end}})\},
\]

where \(t_f = 30\). This function does not specify the evaluation of Get Network Funding prior to time point 30 but guarantees that it will be fulfilled at and after this time. Get Network Funding has two child intentions, described by the and decomposition

\[
\text{(Get Road Funding, Get Startup Funding)} \quad \text{and} \quad \text{Get Network Funding}.
\]

For this question, Algorithm 1 returns \textit{null} because no such path exists, which we demonstrate here. We start by analyzing the first call to \textsc{Solve} at time point 30, where the head of \textit{uGoals} is an arbitrary element such as \((\text{Support Cooperation}, 30, (F, \bot) \ldots (P, P))\). In this case, the first value in \textit{curEval} is assigned to \textit{val} and then \textsc{Propagate} is called (see Line 22). An evaluation is propagated to \textsc{Get Startup Funding} using the \textit{CONSTANT} evaluations assigned prior to the analysis. At this point, \textsc{Get Startup Funding} is assigned \((\bot, F)\) and \textsc{Get Network Funding} is assigned \((F, \bot)\), meaning \textsc{check\_prop} (Line 23) returns false in accordance with the and decomposition rules in Fig. 6, causing the algorithm to backtrack on Lines 27–28, and select the next \textit{val} in \textit{curEval} on Line 19 of Algorithm 1. Since this violation is not connected to the assignment of an evidence pair to \textit{Support Cooperation} on Line 20, a similar process continues until \((P, P)\) is tried, and again \textsc{check\_prop} returns false resulting in backtracking. This completes the for loop on Line 19, at which point \textsc{Solve} returns null on Line 29. Ultimately, Algorithm 1 returns null because no satisfying assignment can be made at time point 30 resulting in the fulfillment \((F, \bot)\) of Get Network Funding. Therefore, the answer is no, it is not possible to fulfill Get Network Funding at time point 30.

\textbf{Summary.} In summary, the Evolving Intentions framework with path-based analysis is able to fully reproduce the timeline of actual events as documented. The analysis was found to be effective at answering questions we asked. As with all GORE approaches, the questions that can be answered with analysis depend on what is included in the model. In an iterative process, the requirements engineer selects which questions to explore and what elements about the domain to capture in the model. In the next subsection, we give runtime information for these questions to demonstrate scalability of our analysis.

\section{7.3 Scalability}

We demonstrate the scalability of the CSP-based analysis for generating simulation paths using actual run-time data of the examples discussed in this paper and on randomly generated examples. All scalability tests were completed on a Silicon Mechanics Rackform iServ R331.v4 with two 12-core Intel E5-2697v2 CPUs and 128G of RAM.

Using BloomingLeaf, we calculated runtimes for each of the examples discussed in this paper. The bike lanes example is a relatively small graph. Runtimes for EQ1–EQ6 varied between 9 and 66 ms (<1 s), whereas runtimes for the Spadina graph (VQ1–VQ4) varied between 1881–3568 ms (1.8–3.6 s), except for VQ4, where the graph was found to be infeasible. The runtime for VQ4 was under 1 ms. BloomingLeaf automatically generates a time point path Π, given a goal graph. In the case of the Spadina graph, the solver finds assignments for 2832 Boolean variables (of the 17950 Boolean and integer variables created to define the model). The resulting path gives an evaluation of the graph (see Definition 20) for each time point in Π.

Given the run-times for the goal graphs discussed above, we consider whether the number of intentions in a graph affects the computation time. To evaluate variations in the number of intentions, we created a set of models by linking all intentions in a tree structure and varying the number of intentions in the models as follows: 25, 51, 75, 101, 125, 151, 175, and 201. Note that the Spadina graph was 124 intentions. For each model, we assigned atomic functions (see Sect. ) to portions of the model intentions. We calculated the mean run-time (in seconds) with min/max bars for generating paths of length 5, 10, 25, 50, 75, 100, 150, and

![Fig. 16 Computation times for graph evaluation paths with varying path lengths](image-url)
Figure 16 presents the results of this evaluation, and shows that both model size and path length impact the runtime. If we assume an upper bound of 30 seconds for human-in-the-loop computation, then our approach is scalable up to models of 150 intentions (i.e., larger than the Spadina graph). In the future, we aim to investigate further efficiencies in our encoding for BloomingLeaf to improve computation times for models with more than 150 nodes, though we have not encountered models of this size in practice.

8 Related work

In this section, we compare our approach with related work.

Changing Intentions We begin by further exploring the connection between this article and our prior work using iStar [23, 27]. The core idea in both approaches is to vary evaluation labels of intentions within a goal model over time to make trade-off decisions given future evolutions. There are two key differences between Tropos and iStar. First, iStar uses different contribution links (e.g., makes) and has dependency links. This change impacts binary evolving relationships, where in iStar the relationships are makes, helps, hurts, breaks, and depends. Second, iStar uses seven qualitative evaluation labels (i.e., Fully Satisfied, Partially Satisfied, Partially Denied, Fully Denied, Conflict, Unknown, and None) resulting in different semantics for evaluation at a time point and all functions for intentions, as well as the definition of an evolving goal graph. There is no notion of a conflict prevention level, but instead, Conflict and Unknown are assigned only via propagation. We are able to represent both absolute and time points, whereas our prior work on iStar only uses relative time points [23]. This also enables us to include constraints over time points and the NotBoth evolving function.

Aprajita and Mussbacher provide a quantitative framework (called TimedGRL) that enables stakeholders to consider possible evolutions and trends through visualizations [7]. In the associated thesis, Aprajita provides a detailed example of how TimedGRL can be applied to review the performance of a project [5]. In subsequent work, called TimedURN, Aprajita et al. add support for feature models [6] and provide extensive tooling [41]. If we consider , TimedGRL can directly address EQ5 and EQ2 by specifying how the elements change in different contexts and then evaluating the model at each time point for a given context. TimedGRL partially supports answering other questions, such as EQ4, and EQ3, by creating different scenarios and manually comparing results. To fully answer the remaining questions, TimedURN requires additional change types to determine values nondeterministically, for which it was not originally intended. TimedGRL allows for the addition and deletion of intentions in the model with deactivation changes, and has additional model management features for testing different evolving functions (i.e., dynamic contexts).

Incremental results The analysis presented in this paper generates complete simulation results from \( r = 0 \). Nguyen et al. studied how to minimize total effort in finding incremental solutions when evolution occurs in the form of requirements changes [44, 45]. We have not investigated generation of incremental solutions based on previous or partial paths, but the analysis described in this paper can be extended to do that.

Our framework explored making decisions in the context of known future changes. Ernst et al. looked at incrementally updating prior decisions as requirements evolve [14]. Allkaf et al. studied change impact analysis for intention evaluations in GRL models [2, 1]. Others have investigated capturing change histories [28] and understanding model stability [29]. We envision connecting our work with these downstream efforts in the future.

Representation of time Our choice of representation allows for analyzing tradeoffs given changes in sociotechnical systems. An alternative approach would be system dynamics, a methodology of scientific prediction used to model emergent properties of complex systems behavior, where changes are nonlinear [15, 51]. In , system dynamics would allow us to model traffic patterns of motorists and cyclists given different configurations, but would not allow us to make trade-off decisions in project planning. Creating system dynamics simulations requires significant contextual information that is not usually known in the early- phases of RE. The intended purposes of goal modeling and system dynamics are fundamentally orthogonal. System dynamics is suitable for simulating physical resources, but its applicability to this article is limited. Work of Zhang and Liu [53] already considered connecting goal modeling and system dynamics, where causal loop diagrams were mixed with basic goal models to refine top-level goals into causal loop relationships for modeling environmental impacts of systems. Future work can investigate connecting framework with this work.

Choice of logical representation In Sect. 4.1, we introduced our notion of time as ticks, time points and intervals between time points. This choice of representation was motivated by sequential logic from digital circuits and microprocessor design [42]. As we developed these ideas, we chose to use first-order logic, and prioritized the readability and simplicity of our representation over the expressive power. We also considered using LTL and CTL [8, 11], as well as regular expressions [31].

Prior work in goal modeling used LTL for presenting temporal goals and timing properties [19, 40]. LTL and CTL allow for the specification of constraints over paths and trees, respectively. These would not have been suitable for specifying Evolving Goal Graphs (see Definition 17),
but would be suitable for specifying strategies over Graph Evaluation Paths (see Definition 22). For example, using our reasoning stakeholders consider properties over intermediate and final states of intentions, and can write these as LTL formulas where “eventually” an intention has a state. CTL can be used to express constraints over all possible simulations for a graph. For example, if stakeholders want to review all possible paths where an intention is eventually fulfilled. Future work can investigate using LTL and CTL as a way of automatically encoding strategies and stakeholder questions in order to check properties over simulations.

Alternatively, we can imagine specifying evolving functions through a condensed text-based notation that would allow for the use of regular expressions to expand and create paths over individual intentions. This approach was not chosen because it required an additional level of abstraction but still used standard forward and backward analysis algorithms for propagation of labels.

9 Summary, limitations, and future work

This paper presents a complete formalization of the Evolving Intentions framework built on top of the Tropos goal modeling language. We conclude that this formalization satisfies our initial research question to understand how we can effectively support users in making trade-off decisions given evolutionary information. By providing formal semantics, we give a unique interpretation for analysis results over simulation paths, giving stakeholders confidence in their analysis. We conclude that this approach is scalable for sufficiently large examples.

We now discuss the limitations of the Evolving Intentions framework, starting with those inherent in our framework. To start with, we are not able to represent all possible behaviors of model intentions. For example, we constrain evolving functions for relationships to only two time intervals (see Sect. 4.3), and thus, cannot represent relationships that evolve over three or more intervals. See Sect. 4.5 for a complete list of the limitations in the expressive power of the modeling aspects of the formalism.

Furthermore, goal models in general are considered to be open-world artifacts. This means that it is assumed that a decomposition relationship can have an additional source, i.e., child, that is not yet present in the model. To enable automated analysis, the Evolving Intentions framework considers all models to be closed-world artifacts.

As with other GORE approaches, the ability of an approach to generate meaningful answers to trade-off decisions requires the elicitation of model elements at the right level of abstraction. We acknowledge that this is a difficult and time consuming task. The Evolving Intentions framework requires additional information in the specification of the evolving functions, and is therefore limited by the modelers’ ability to express anticipated changes.

We now consider limitations that can be mitigated through future improvements. The Evolving Intentions framework only captures qualitative information, and not all decisions can be reached with only qualitative information. Other approaches, such as TimedGRL [5], consider both qualitative and quantitative changes in intentions over time. Our framework can be easily extended to include a quantitative approach because the formal specification of Tropos [16, 17, 49] already includes the formal foundations for quantitative evaluation.

The framework does not allow for identifying and analyzing project obstacles and hazards, such as in the work of Letier [39]. Future work could create a methodology for obstacle analysis in the context of evolution. We could also look at how goals and obstacles affect risk management more broadly as in the work of Islam and Houmb [34]. Furthermore, we
could develop a methodology to connect our approach with scenario models to help stakeholders discover goals and evolution in their domains (as discussed in Sect. 8).

The methodology presented in Sect. 2 assumes the perspective of a single modeler, and we did not investigate adapting this technique for multiple modelers. We did not formally define the concept of actor availability (i.e., presence conditions), which limits stakeholders’ ability to consider the evolution of actors. Future work will consider merging multiple view points [48] and resolving conflicts between stakeholders. As well, to fully support dialog between stakeholders, we can enable actor-centric reasoning, where analysis methods either prioritize or isolate a subset of actors in the model and consider only their intentions and dependencies. We hope this will improve the versatility of our analysis, and mitigate lengthy analysis run-times over larger models, when not all actors are relevant to the query.

Since we aim to deal with the intentions of multiple stakeholders, we must look broadly at assumptions, beliefs, and design-time uncertainty. Stakeholders have tacit assumptions about the domain and intended users of the system, as well as the intentions of other stakeholders. These assumptions become embedded in the models throughout the modeling process. With respect to reasoning, modelers make assumptions about why the evolving functions and evaluations they selected are appropriate. In KAOS, an assumption is a statement to be satisfied by the environment [38]. These are explicit assumptions made by the system. In GRL, a belief is a fact that is important to a stakeholder [4]. Future work will explore and classify the different types of assumptions made both in the model, and in the modeling and reasoning process. We will develop analysis techniques that will expose and challenge the implicit assumptions of stakeholders. For example, we may connect scenario models with the Evolving Intentions framework as in Rolland et al. [46] to generate additional goals and expose assumptions. We will also consider documenting these assumptions and representing them in the model, creating traceability links for stakeholders. Dhaouadi et al. created a mechanism for connecting heterogeneous models (including Tropos models) using design-time uncertainty (i.e., missing information in how to design the model) [12]. We envision extending this approach in order to specify and resolve uncertainty in evolving functions selected by modelers.

In Sect. 2, we proposed a methodology for using the Evolving Intentions framework, which we developed alongside the framework and our validation efforts. At present, we have not empirically validated the methodology in Sect. 2, and we cannot make claims about real-world stakeholders’ ability to use the evolving functions and path-based analysis defined in this formalism. In a similar method for the iStar goal modeling language, we propose strategies to connect analysis procedures to stakeholder questions [23]. Further research will explore creating similar strategies for the Evolving Intentions framework, as well as empirically validating the methodology.

In Sect. 7.2, we showed how the formalization can explore questions using a historical case study. We hope to further validate the Evolving Intentions framework with a significant case study from software/systems engineering. With a longitudinal study, stakeholders can observe each other’s intentions and their requirements evolving.

Requirements engineers and systems analysts can use the Evolving Intentions framework in the context of domain analysis to understand how stakeholders and the proposed system may be impacted by changes in the socio-technical landscape and software ecosystem. The framework can also be used to model existing systems and demonstrate how past changes impact top-level goals, as well as how proposed changes in the future may impact existing relationships.

In addition to modeling project evolution, we anticipate using the Evolving Intentions framework to model trade-offs in ethical decision making. For example, in the context of autonomous vehicles, we could explore implications for the requirements of a collision avoidance system in choosing between harming occupants of the autonomous vehicle, occupants of other vehicles, and unprotected road users (e.g., cyclists and pedestrians). These ethical decisions affect non-users of autonomous systems and their designs must broadly consider the intentions of all stakeholders and society as a whole.

Appendix: Algorithm 1 helper functions

Propagation algorithms

As mentioned in Sect. 3, when evidence pairs are propagated from source nodes to target nodes, this is called forward propagation. We first consider an adapted version of the forward propagation algorithm introduced by Giorgini et al. [17]. Recall that an evidence pair is a pair (s, d), where s ∈ {F, P, ⊥} is the level of evidence for and d ∈ {F, P, ⊥} is the level of evidence against the fulfillment of g, where F > P > ⊥. In the following algorithms, we use Sat(g) to refer to the s value of an evidence pair and Den(g) to refer to the d value of an evidence pair. The forward propagation rules are listed on lines 1–32 in Fig. 6. To make the propagation algorithm easier to understand, in Table 7, we restate these rules in terms of (s, d) pairs (i.e., Sat(g) and Den(g)) for the source and target intentions. For example, the first row gives the Sat and Den rules for decomposition (i.e., and & or) and directly connects to the rules on Lines 1–4 for and and Lines 5–8 for or of Fig. 6.
**UPDATE_LABEL function (Algorithm 2).** Suppose that a target has more than one incoming link. Using Table 7 is insufficient because it does not specify how to combine values from multiple links. The UPDATE_LABEL function (see Algorithm 2) specifies how the evidence pair for an intention \( g \) is updated based on the links for which \( g \) is the target intention. UPDATE_LABEL takes as input an intention \( g \), a time point \( t \), a set of relationships \( \text{targetRel} \), and a set of tuples \( \text{Old} \) mapping intentions to evidence pairs, and returns a new evidence pair assignment for the input goal \( g \) at time point \( t \). Line 18 assigns the old evidence pair to \( \text{sat} \) and \( \text{den} \). Then, the main loop (Lines 19–21) iterates through the relationships in \( \text{targetRel} \), and invokes \( \text{Apply_Rules_Sat} \) and \( \text{Apply_Rules_Den} \) returning new \( \text{sat} \) and \( \text{den} \) values, respectively, for the selected relationship based on the rules in Table 7. After iterating over all the incoming relationships the updated \( \text{sat} \) and \( \text{den} \) values for the goal \( g \) at time \( t \) are the maximum values (i.e., strongest evidence) that was propagated from one of the incoming links.

For example, suppose \( \text{Old} \) contains the tuples \( \langle \text{HaveDesign}, 0, (F, \bot) \rangle \), \( \langle \text{HaveBikeLanes}, 0, (\bot, \bot) \rangle \), and \( \langle \text{BuildBikeLanes}, 0, (F, F) \rangle \). When \( \text{UPDATE\_LABEL} \) is called for \( \text{Have Bike Lanes} \), the returned value for \( \text{Have Bike Lanes} \) will be \( (P, F) \), based on the \( \text{P} \) and \( \text{F} \) relationship between them.

**Algorithm 2 PROPAGATE: Forward Graph Evaluations for Partially Evaluated Goal Graphs**

```
1: function PROPAGATE(G, R, aGoals, uGoals, t)  \>
2:     do  \>
3:         size = |aGoals|  \>
4:         for all \( g \in G \) do  \>
5:             if aGoals.getVal(g, t) \neq null then continue  \>
6:                 targetRel = R.getLinkSetForTargetAtTime(g, t) \>
7:                     \text{get all relationships at time point } t \text{ with } g \text{ as a target intention.}  \>
8:                 for all \( r \in \text{targetRel} \) do  \>
9:                     sGoals = r.getSourceSet  \>
10:                    if sGoals \& aGoals.getGoalsAtTime(t) \neq \emptyset then goto Line 14  \>
11:                    targetGoalEval = UPDATE\_LABEL(g, t, targetRel, aGoals)  \>
12:                    if targetGoalEval \in uGoals.getPossibleEvidence(g, t) then  \>
13:                        aGoals.add(g, t, targetGoalEval)  \>
14:                    uGoals.remove(g, t)  \>
15:         while |aGoals| \neq size  \>
16:     return (aGoals, uGoals)
```

**PROPAGATE function (Algorithm 2).** This function propagates values from source nodes to target nodes. The PROPAGATE function (see Algorithm 2) takes as input goals \( G \), relationships \( R \), and a single time point \( t \), as well as two sets, \( uGoals \) and \( aGoals \), to represent unassigned and assigned values, respectively. Each element of \( uGoals \) is of the form \( \langle g, t, \emptyset \rangle \) and may be specified by the user. Each element of \( aGoals \) is of the form \( \langle g, t, \langle g[t]\rangle \rangle \) and may be specified by the user. Each element of \( aGoals \) is of the form \( \langle g, t, \emptyset \rangle \), where the third element in the tuple is a set of possible evidence pairs in \( E \). PROPAGATE returns an updated copy of \( aGoals \) and \( uGoals \). The main loop (see Lines 2–15) iterates until a fix-point is found, where no additional evidence pairs have been assigned to intentions (i.e., \( |aGoals| = size \), on Line 15). The updated sets are returned on Line . The inner loop (see Lines 7–9) evaluates each relationship \( r \in R \). If the target of \( r \) is already assigned, then the relationship is skipped (see 5). If the target of \( r \) is not already assigned and all of the source intentions of \( r \) are assigned (Line 9), then UPDATE_LABEL is called to assign an evidence pair for the target intention. Finally, the target intention is removed from \( uGoals \) and added to \( aGoals \) with the assigned evidence pair (Lines 12 and 13).
A value is only assigned to a target intention when all source intentions have an assigned value in \( \mathcal{E} \). For example, consider the Bike Lanes example fragment in Fig. 1 (with \( R \) specified in Sect. 3) and the following values for \( aGoals \) and \( uGoals \) (assuming Weak_Conflict avoidance at timepoint 0): \( aGoals = \{ \langle \text{PermanentConstruction}, 0, (F, \bot) \rangle, \langle \text{HaveDesign}, 0, (P, \bot) \rangle, \langle \text{TemporaryConstruction}, 0, (\bot, F) \rangle \} \), \( uGoals = \{ \langle \text{HaveBikeLanes}, 0, \{(F, \bot), (P, \bot), (\bot, P), (\bot, F)\} \rangle, \langle \text{CyclistSafety}, 0, \{(F, \bot), (P, \bot), (\bot, P), (\bot, F), (\bot, \bot)\} \rangle, \langle \text{HaveBikeLanes}, 0, \{(F, \bot), (P, \bot), (\bot, P), (\bot, F), (\bot, \bot)\} \rangle \} \), are used as inputs to Algorithm 2. The resulting output value is \( aGoals = \{ \langle \text{ PermanentConstruction}, 0, (F, \bot) \rangle, \langle \text{ BikeLaneUsage}, 0, (P, \bot) \rangle, \langle \text{ TemporaryConstruction}, 0, (\bot, F) \rangle, \langle \text{ AccessToParking}, 0, (P, \bot) \rangle, \langle \text{ BuildBikeLanes}, 0, (F, \bot) \rangle \} \). Have Bike Lanes and Cyclist Safety are not assigned values because Have Design does not have a value and one cannot be generated through forward analysis.

\[
\text{Algorithm 3: CHECK_PROP: Checks Propagation Rules for All Assigned Relationships}
\]

1: function CHECK_PROP(G, R, \Pi, aGoals) \>
2: for all \((g, t, epa) \in aGoals\) do \>
3: \(\text{linkEvalSet} = \emptyset\) \> Set of pairs of link type and source intention evidence pair set. Type: \((\text{type}, \{[g_1], \ldots, [g_n]\})\)
4: \(\text{targetRel} = R \text{. getLinkSetForTarget}(g)\) \> Get relationship set that has this intention as a target.
5: for all \(r \in \text{targetRel}\) do \>
6: \(sEvals = \emptyset\) \>
7: for all \(s \in r \text{. sourceGoalSet}\) do \>
8: if \(aGoals \text{. getEval}(s, t) \neq \emptyset\) then \>
9: \(sEvals.\text{add}(aGoals \text{. getEval}(s, t))\) \> Add evidence pair to set.
10: if \(sEvals \neq \emptyset\) then \>
11: if \(r \text{. areEvolving} \land t \geq r \text{. tRef}\) then \>
12: \(\text{linkEvalSet} \text{. add}(r \text{. evolveType}, sEvals)\) \>
13: else \>
14: \(\text{linkEvalSet} \text{. add}(r \text{. type}, sEvals)\)
15: if \(\text{linkEvalSet} = \emptyset\) then continue \>
16: if \(-\text{SAT_TEST}(epa, \text{linkEvalSet})\) then return \text{false} \> See rules in Fig. 6.
17: return \text{true}\]

\(\text{CHECK_PROP function (Algorithm 3).}\) The \text{CHECK_PROP function (see Algorithm 3)} determines whether any intention evaluations in a goal graph violate the propagation rules for relationships (see Fig. 6). This function is used to verify that assignments made by \text{PROPAGATE} (see Algorithm 2) satisfy all propagation rules. The inputs to \text{CHECK_PROP} are \(G\): the set of goals, \(R\): the set of relationships, \(\Pi\): the set of time points, and \(aGoals\): a set of tuples that map intentions to evidence pairs ((\(g, r, \{[g_1], \ldots, [g_n]\}\))). The main loop (see Lines 2–16) iterates over each tuple in \(aGoals\) and checks the relationships where the goal is a target node. The relationship loop (see Lines 5–14) iterates over each relationship where \(g\) is the target goal. For each relationship, the function loops over the source intentions to determine if they are assigned (see Lines 7–9), meaning that they are elements of \(aGoals\). Evidence pairs for source goals are stored in \(sEvals\). If the source goal has an assignment \(sEvals \neq \emptyset\), then the relationship is added to the \(\text{linkEvalSet}\) to be checked. When all relationships have been added to \(\text{linkEvalSet}\), \text{SAT_TEST} is called on Line 16, which checks the assignments against the rule in Fig. 6. If a violation is found, \text{CHECK_PROP} returns \text{false}. Once all intentions have been checked, no violations have been discovered and \text{CHECK_PROP} returns \text{true}.

For example, \text{CHECK_PROP} would return \text{true} for the output of the \text{PROPAGATE} example discussed above, \text{where} \(aGoals = \{\langle \text{PermanentConstruction}, (F, \bot)\rangle, \langle \text{BikeLaneUsage}, (P, \bot)\rangle, \langle \text{TemporaryConstruction}, (\bot, F)\rangle, \langle \text{AccessToParking}, (P, \bot)\rangle, \langle \text{BuildBikeLanes}, (F, \bot)\rangle\}\).

However, only the \(+\) relationship for \text{Build Bike Lanes} and the \(+\) relationship for \text{Access to Parking} would have been checked by \text{SAT_TEST} on Line 16.
Algorithm 4 ASSIGN_CONSTANT: Assigned evidence pairs for intentions with CONSTANT functions.

1: function ASSIGN_CONSTANT(EF, II, aGoals, uGoals)
2:   for all item \( \in EF \) do
3:     for all \( f \in \text{item.function} \) do
4:         if \( f.\text{type} = \text{CONSTANT} \) then
5:             \( g = \text{item.goal} \), \( x = f.x \),
6:             for all \( t \in II \land t \in [f.\text{start}, f.\text{stop}] \) do
7:                 if \( aGoals(g, t) \neq \text{null} \land aGoals(g, t) \neq x \) then return false
8:             else
9:                 \( aGoals.add(g, t, x) \)
10:                uGoals.remove(g, t)
11:     return true

Algorithm 5 CHECK_EL_FUNC: Checks Evolving Functions for Intentions

1: function CHECK_EL_FUNC(EF, II, aGoals)
2:   for all item \( \in EF \) do
3:     \( g = \text{item.goal} \)
4:     for all \( f \in \text{item.function} \) do
5:         type = \( f.\text{type} \), \( x = f.x \), \( \text{start} = f.\text{start} \), \( \text{stop} = f.\text{stop} \)
6:         if \( \text{type} = \text{CONSTANT} \) then
7:             for all \( t \in II \land t \in [\text{start}, \text{stop}] \) do
8:                 if \( aGoals(g, t) \neq \text{null} \land aGoals(g, t) \neq x \) then return false
9:             else if \( \text{type} = \text{INCREASE} \) then
10:                for all \( t \in II \land t \in [\text{start}, \text{stop}] \) do
11:                    if \( aGoals(g, t) \neq \text{null} \land aGoals(g, t) > x \) then return false
12:                for all \( c \in II \land c \in [\text{start}, \text{stop}] \) do
13:                    if \( aGoals(g, t) \neq \text{null} \land aGoals(g, c) \neq \text{null} \land t > c \land aGoals(g, t) < aGoals(g, c) \) then
14:                        return false
15:                else if \( \text{type} = \text{DECREASE} \) then
16:                    for all \( t \in II \land t \in [\text{start}, \text{stop}] \) do
17:                        if \( aGoals(g, t) \neq \text{null} \land aGoals(g, t) < x \) then return false
18:                    for all \( c \in II \land c \in [\text{start}, \text{stop}] \) do
19:                        if \( aGoals(g, t) \neq \text{null} \land aGoals(g, c) \neq \text{null} \land t > c \land aGoals(g, t) > aGoals(g, c) \) then
20:                            return false
21:   return true

Algorithm 6 CHECK_NOTBOTH: Checks Evidence Pair for Intentions with NOTBOTH Functions

1: function CHECK_NOTBOTH(R, II, aGoals) \( \triangleright \) Check NotBOTH functions.
2:   for all \( r \in R \) do
3:     if \( r.\text{type} = \text{NB} \lor r.\text{type} = \text{NBD} \) then
4:         if \( r.\text{type} = \text{NB} \) then \( fVal = (\bot, \bot) \)
5:         else \( fVal = (\bot, F) \)
6:     \( g_1 = r.g_1, g_2 = r.g_2, t_{Ref} = r.\text{timePoint} \)
7:     for all \( t \in II \) do
8:       if \( t < t_{Ref} \) then
9:         for each \( g \in \{g_1, g_2\} \) do
10:            if \( aGoals(g, t) \neq \text{null} \) then
11:               if \( aGoals(g, t) \neq (\bot, \bot) \) then return false
12:           else if \( t = t_{Ref} \) then
13:               if \( aGoals(g_1, t) \neq \text{null} \land aGoals(g_2, t) \neq \text{null} \) then
14:                 if \( \neg (aGoals(g_1, t) = (F, \bot) \land aGoals(g_2, t) = fVal) \lor
15:                   (aGoals(g_1, t) = fVal \land aGoals(g_2, t) = (F, \bot)) \) then return false
16:           else if \( t > t_{Ref} \) then
17:               if \( aGoals(g_1, t_{Ref}) \neq \text{null} \land aGoals(g_2, t_{Ref}) \neq \text{null} \) then
18:                 for each \( g \in \{g_1, g_2\} \) do
19:                   if \( aGoals(g, t) \neq \text{null} \) then
20:                     if \( aGoals(g, t) \neq aGoals(g, t_{Ref}) \) then return false
21:   return true

\( \odot \) Springer
Evolving function algorithms

In this subsection, we present three helper functions (i.e., ASSIGN_CONSTANT, CHECK_EL_FUNC, and CHECK_NOTBOTH) used in Algorithm 1 to check the evolving functions in the evolving goal graph.

ASSIGN_CONSTANT function (Algorithm 4). The purpose of the ASSIGN_CONSTANT function (see Algorithm 4) is to increase the efficiency of Algorithm 1 by assigning all evidence pairs that are known prior to reasoning because they are constrained by a CONSTANT function. ASSIGN_CONSTANT iterates over each intention with a USER-DEFINED function (see Line 2), and again for each atomic function contained within (see Line 3). If the function \( f \) is a CONSTANT function, then Algorithm 4 assigns evidence pairs over the specified period (see Lines 9–10). Prior to this assignment, Algorithm 4 also checks that the proposed assignment is not in conflict with any User Evaluations in UEVal. If a conflict is found, the algorithm returns false meaning the graph is infeasible (see Line 7).

In the bike lanes example, ASSIGN_CONSTANT selects evidence pairs for Have Design over the entire path \((\perp, F)\) at \( t = 0 \) and \((F, \perp)\) after, and for Bike Lane Usage at \( t = \{17, 32, 45\} \) when it is satisfied \((F, \perp)\) in the summer.

CHECK_EL_FUNC function (Algorithm 5). The purpose of the CHECK_EL_FUNC (see Algorithm 5) function is to check that no evolving functions for intentions have been violated while assigning evidence pairs. This Boolean function takes as input a set of goal assignment \( aGoals \), the time point path \( \Pi \), and \( \mathcal{E}F \) which consists of mappings between intentions and USER-DEFINED functions (see Definition 12). Each mapping, made up of atomic functions (see Definition 8), takes the form

\[ \langle g, \{(\text{type}_0, x_0, \text{start}_0, \text{stop}_0), \ldots, (\text{type}_n, x_n, \text{start}_n, \text{stop}_n)\} \rangle. \]

CHECK_EL_FUNC returns true if no atomic functions have been violated. The main loop (see Lines 2–20) iterates over each item in the set \( \mathcal{E}F \). The function loop (see Lines 4–20) then iterates over each atomic function (initializing values on Line 5) and checks each value of the time point path \( \Pi \) within the range \([\text{start}, \text{stop}]\) depending on the type (i.e., CONSTANT, INCREASE, DECREASE). For example, the check for CONSTANT (see Lines 6–8) verifies that each assigned evidence pair (i.e., having a value in \( aGoals \)) is equal to \( x \). If all checks pass and the main loop completes, CHECK_EL_FUNC returns true on Line 21, but if a single check fails, false is returned.

As mentioned when discussing Definition 6 (see Sect. 5) if stakeholders add the user evaluation in conjunction with an evolving function, this may result in an inconsistency. These inconsistencies are found by CHECK_EL_FUNC. For example, if Have Bike Lanes has the evolving function \( \langle \text{CONSTANT}, (F, \perp), tBLopen, tend \rangle \), and is accidentally assigned the user evaluation \( \langle \text{HaveBikeLanes}, t_{u3}, (\perp, F) \rangle \), where \( tBLopen \) is before \( t_{u3} \), then this inconsistency would be found by CHECK_EL_FUNC on Line 8 returning false.

CHECK_NOTBOTH function (Algorithm 6). Similarly, CHECK_NOTBOTH inspects the assignments for both intentions in each NOTBOTH relationship function in the evolving goal graph (see Algorithm 6). CHECK_NOTBOTH returns true iff expected values of all assignments are consistent with Definition 15. NOTBOTH relationships are stored in the set of relationships \( R \); thus, the main loop (see Lines 2–20) iterates over each relationship and only investigates further for NBN and NBD relationships. After initializing relevant values on Lines 7–20, the time point loop (see Lines 8–11) iterates over each time point in \( \Pi \). For time points before the decision \((t < tRef)\), CHECK_NOTBOTH verifies that assigned intentions (i.e., intentions having values in \( aGoals \)) are \((\perp, \perp)\) (see Lines 12–15). For the decision time point \( tRef \), CHECK_NOTBOTH verifies that one intention is \((F, \perp)\) and the other has \( fVal \), the value specified by the type, where \( fVal \) is \((\perp, \perp)\) for NBN and \((\perp, F)\) for NBD (see Lines 12–15). For time points after the decision \((t > tRef)\), CHECK_NOTBOTH verifies that assigned values are the same as those at the time when the decision was made (at \( tRef \)) (see Lines 16–20). In the bike lanes example, CHECK_NOTBOTH would test the value of the NBN relationship between Permanent Construction and Temporary Construction.

References

7. Aprajita, Mussbacher, G (2016) TimedGRL: specifying goal models over time. In: Proceedings of the the sixth international workshop on model-driven requirements engineering (MoDRE’16)


Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.