Computing Trajectories of Dynamic Systems Using ASP and $\mathcal{F}$LORA-*2*

Daniela Inclezan

Computer Science Department
Texas Tech University
Lubbock, TX 79409 USA
daniela.inclezan@ttu.edu

Abstract. The paper explores the possibility of performing temporal projection in the "Digital Aristotle" reasoning system. In particular, it investigates the relationship between two methods for computing trajectories of dynamic systems: a first method using action languages and the answer set semantics, and a second method using the object-oriented declarative language $\mathcal{F}$LORA-*2*. The former method is sound and complete with respect to the specification. We show that the latter method is also sound, and we identify a sufficient condition for completeness. Based on these results, we present advantages and limitations of the two methods.

1 Introduction

The paper explores the possibility of performing temporal projection in Digital Aristotle, "a reasoning system capable of answering novel questions and solving advanced problems in a broad range of scientific disciplines and related human affairs" [1]. The pilot of the Digital Aristotle uses the object-oriented declarative language $\mathcal{F}$LORA-*2* [2], which relies on the well-founded semantics [3]. $\mathcal{F}$LORA-*2* is a dialect of F-logic [4] with various extensions, including argumentation theory. The inference engine with an identical name, $\mathcal{F}$LORA-*2*, is based on XSB and uses tabling.

Current research goals of the Digital Aristotle focus on performing temporal projection in *discrete dynamic systems*. Such systems can be modeled by transition diagrams whose nodes are possible physical states of the domain and whose arcs are labeled by actions. As the actions of the domain may be non-deterministic, computing trajectories (i.e., paths in the transition diagram) is not a straightforward task. There is a known method [5] of performing temporal projection using action languages and the answer set semantics [6], [7]. Action languages specify discrete dynamic systems in a concise and mathematically accurate manner. We developed a first method of computing trajectories as a refinement of this known method. We specified transition diagrams in a modular action language, $\mathcal{ALM}$ [8], extending action language $\mathcal{AL}$ [9], [10] by means for expressing hierarchies of abstractions that are often necessary for the design of larger knowledge bases.

$\mathcal{AL}$ incorporates in its semantics the *inertia axiom* [11] that says that "*Normally, things stay the same.*" System descriptions written in $\mathcal{AL}$ are translated into logic programs under the answer set semantics; these logic programs are then used in solving various reasoning tasks (trajectory computation, planning, diagnosis, etc.) [5], [12]. We call this first method the "ASP method".

However, the ASP method cannot be used by the Digital Aristotle, as the Digital Aristotle relies on a different formalism, $\mathcal{F}$LORA-*2*. Therefore, we created a method of computing trajectories in discrete dynamic systems using the language $\mathcal{F}$LORA-*2*, and then answered questions using the inference engine with the same name. We compared the $\mathcal{F}$LORA-*2* method with our first method based on action languages and answer set semantics. This comparative mathematical analysis revealed advantages and limitations for the two methods. For the class of $\mathcal{F}$LORA-*2* programs used in computing trajectories, this comparison also provided a *specification*—in terms of transition diagrams and thus independent from the computational technique.

The paper is structured as follows: we start by introducing the syntax and semantics of $\mathcal{AL}$ with defined fluents. The specification of dynamic systems will be given in this language.[1] In the following section we present our two methods for computing trajectories: we start with the ASP method and continue with the $\mathcal{F}$LORA-*2* method. Rather than making use of all the extensions incorporated in $\mathcal{F}$LORA-*2*, we limit ourselves to the well-founded semantics with classical negation and ignore, for example, the argumentation theory. In the next section we give a mathematical analysis of the two methods. We indicate that the ASP method is sound and complete with respect to the specification. We show that the $\mathcal{F}$LORA-*2* method is also sound, and we formulate a sufficient condition for its completeness. Finally, we present advantages and limitations of the two methods.

2 Action Language $\mathcal{AL}$

In this section we quickly present the syntax and semantics of $\mathcal{AL}$ with *defined fluents*. For more details, see [8].

2.1 Syntax of $\mathcal{AL}$

A *system description* of $\mathcal{AL}$ consists of a sorted *signature* and a collection of *axioms*. The signature contains names for primitive *sorts*, a *sorted universe* (nonempty sets of object constants assigned to primitive sorts), and names for *actions* and *fluents*. The fluents are partitioned into *static*, *inertial*, and *defined fluents*. The truth values of static fluents cannot be changed by actions. Inertial fluents can be changed by actions and are subject to the law of inertia. Defined fluents

[1] In the actual work, dynamic systems were specified in the modular action language $\mathcal{ALM}$. As system descriptions of $\mathcal{ALM}$ are mapped into equivalent system descriptions of $\mathcal{AL}$, we abstracted away from $\mathcal{ALM}$ for the purpose of this paper.

are non-static fluents that are defined in terms of other fluents; they can be changed by actions but only indirectly.

An atom is a string of the form $p(\bar{x})$ where p is a fluent and $\bar{x}$ is a tuple of primitive objects. A (*static, inertial* or *defined*) *literal* is an atom or its negation. Direct causal effects of actions are described in $\mathcal{AL}$ by *dynamic causal laws* – statements of the form:

$$a \textbf{ causes } l \textbf{ if } p \tag{1}$$

where l is an inertial literal, a is an action name, and p is a collection of arbitrary literals. (1) says that if action a were executed in a state satisfying p then l would be true in a state resulting from this execution. Dependencies between fluents are described by *state constraints* — statements of the form:

$$l \textbf{ if } p \tag{2}$$

where l is a literal and p is a set of literals. (2) says that every state satisfying p must satisfy l. *Executability conditions* of $\mathcal{AL}$ are statements of the form:

$$\textbf{impossible } a_1, \ldots, a_k \textbf{ if } p \tag{3}$$

Statement (3) says that actions $a_1 \ldots a_k$ cannot be executed simultaneously in a state that satisfies p. An $\mathcal{AL}$ axiom with variables is understood as a shorthand for the set of all its ground instantiations.

2.2 Semantics of $\mathcal{AL}$

We define the semantics of $\mathcal{AL}$ by defining the transition diagram $\mathcal{T}(\mathcal{D})$ for every system description $\mathcal{D}$ of $\mathcal{AL}$. Some preliminary definitions: a set σ of literals is called *complete* if for any fluent f either f or $\neg f$ is in σ; σ is called *consistent* if there is no f such that $f \in \sigma$ and $\neg f \in \sigma$. Our definition of the transition relationship $\langle \sigma_0, a, \sigma_1 \rangle$ of $\mathcal{T}(\mathcal{D})$ *will be based on the notion of an answer set of a logic program* [7]. We will construct a program $\Pi_S(\mathcal{D})$ consisting of logic programming encodings of statements from $\mathcal{D}$. The answer sets of the union of $\Pi_S(\mathcal{D})$ with the encodings of a state σ_0 and an action a will determine the states into which the system can move after the execution of a in σ_0.

Let $\mathcal{D}$ be a system description of $\mathcal{AL}$. The signature of $\Pi_S(\mathcal{D})$ consists of: (a) names from the signature of $\mathcal{D}$; (b) a new sort: *step* with constants 0 and 1; and (c) the relations: $holds(fluent, step)$ ($holds(f, i)$ says that fluent f is true at step i); $occurs(action, step)$ ($occurs(a, i)$ says that action a occurred at step i); and $fluent(fluent\_type, fluent)$ ($fluent(t, f)$ says that f is a fluent of type t). If l is a literal formed by a non-static fluent, $h(l, i)$ will denote $holds(f, i)$ if $l = f$ or $\neg holds(f, i)$ if $l = \neg f$. Otherwise $h(l, i)$ is simply l. If e is a set of actions, $occurs(e, i) = \{occurs(a, i) : a \in e\}$. If p is a set of literals, $h(p, i) = \{h(l, i) : l \in p\}$.

The logic program $\Pi_S(\mathcal{D})$ is constructed as follows:

1. For every static causal law "l **if** p" from $\mathcal{D}$, $\Pi_S(\mathcal{D})$ contains:

$$h(l, I) \leftarrow h(p, I). \tag{4}$$

2. For every dynamic causal law "a **causes** l **if** p" from $\mathcal{D}$, $\Pi_S(\mathcal{D})$ contains:

$$\begin{array}{rl} h(l, I+1) \leftarrow & h(p, I), \\ & occurs(a, I). \end{array} \tag{5}$$

3. For every executability condition "**impossible** $a_1, \ldots, a_k$ **if** p" from $\mathcal{D}$, $\Pi_S(\mathcal{D})$ contains:

$$\neg occurs(a_1, I) \vee \ldots \vee \neg occurs(a_k, I) \leftarrow h(p, I). \tag{6}$$

4. $\Pi_S(\mathcal{D})$ contains the Inertia Axioms:

$$\begin{array}{rll} holds(F, I+1) & \leftarrow & fluent(\mathbf{inertial}, F), \\ & & holds(F, I), \\ & & \text{not } \neg holds(F, I+1). \\ \neg holds(F, I+1) & \leftarrow & fluent(\mathbf{inertial}, F), \\ & & \neg holds(F, I), \\ & & \text{not } holds(F, I+1). \end{array} \tag{7}$$

5. $\Pi_S(\mathcal{D})$ contains the CWA for defined fluents:

$$\begin{array}{rl} \neg holds(F, I) \leftarrow & fluent(\mathbf{defined}, F), \\ & \text{not } holds(F, I). \end{array} \tag{8}$$

6. $\Pi_S(\mathcal{D})$ contains the constraint:

$$\begin{array}{rl} \leftarrow & fluent(\mathbf{inertial}, F), \\ & \text{not } holds(F, I), \\ & \text{not } \neg holds(F, I). \end{array} \tag{9}$$

7. For every static fluent f, $\Pi_S(\mathcal{D})$ contains the constraint:

$$\begin{array}{rl} \leftarrow & fluent(\mathbf{static}, f), \\ & \text{not } f, \\ & \text{not } \neg f. \end{array} \tag{10}$$

Let $\Pi_{Sc}(\mathcal{D})$ be a program constructed by rules (4), (8), (9), and (10) above. For any set σ of literals, σ_{nd} denotes the collection of all literals of σ formed by inertial and static fluents. $\Pi_{Sc}(\mathcal{D}, \sigma)$ is obtained from $\Pi_{Sc}(\mathcal{D}) \cup h(\sigma_{nd}, 0)$ by replacing I by 0.

Definition 1 (State). A set σ of literals is a *state* of $\mathcal{T}(\mathcal{D})$ if $\Pi_{Sc}(\mathcal{D}, \sigma)$ has a unique answer set, A, and $\sigma = \{l : h(l, 0) \in A\}$.

Now let σ_0 be a state and e a collection of actions.

$$\Pi_S(\mathcal{D}, \sigma_0, e) =_{def} \Pi_S(\mathcal{D}) \cup h(\sigma_0, 0) \cup occurs(e, 0) \ .$$

Definition 2 (Transition). A *transition* $\langle \sigma_0, e, \sigma_1 \rangle$ is in $\mathcal{T}(\mathcal{D})$ iff $\Pi_S(\mathcal{D}, \sigma_0, e)$ has an answer set A such that $\sigma_1 = \{l : h(l, 1) \in A\}$.

As an illustration of this definition we consider:

Example 1. [Lin's Briefcase]([13])
The system description defining this domain consists of: (a) a signature containing the sort name *latch*, the sorted universe $\{l_1, l_2\}$, the action $toggle(latch)$, the inertial fluent $up(latch)$ and the defined fluent *open*, and (b) the following axioms:

$toggle(L)$ **causes** $up(L)$ **if** $\neg up(L)$
$toggle(L)$ **causes** $\neg up(L)$ **if** $up(L)$
$open$ **if** $up(l_1), up(l_2)$.

One can use our definitions to check that the system contains transitions
$\langle \{\neg up(l_1), up(l_2), \neg open\}, toggle(l_1), \{up(l_1), up(l_2), open\}\rangle$,
$\langle \{up(l_1), up(l_2), open\}, toggle(l_1), \{\neg up(l_1), up(l_2), \neg open\}\rangle$, etc.

Note that a set $\{\neg up(l_1), up(l_2), open\}$ is not a state of our system.

The semantics of system descriptions that do not contain defined fluents is equivalent to the semantics described in [14] and [15]. As far as we know, [14] is the first work which uses ASP to describe the semantics of action languages.

3 Computing Trajectories

A system description $\mathcal{D}$ of $\mathcal{AL}$ specifies the *entire* transition diagram $\mathcal{T}(\mathcal{D})$ representing a dynamic domain. In order to reason about a *specific trajectory* of an agent in this domain, we need to add a recorded history, i.e., a collection of observations made by the agent together with a record of its own actions. Trajectories can then be computed by combining together:

1. the recorded history
2. a logic program encoding the system description and
3. a logic program connecting the recorded history to the transition diagram.

This general method can be adapted to different non-monotonic formalisms by ensuring that the logic programs in points 2 and 3 obey specific syntactic requirements. Our ASP and $\mathcal{F}$LORA-*2* methods follow this pattern. As they rely on different formalisms, the only part the two methods have in common is the recorded history. Let us now define it formally.

Definition 3 (History). *By the* history Γ_n *of a system description* $\mathcal{D}$ *up to time step* n *we mean a collection of observations, i.e., facts of the form:*
1. $observed(f, true/false, i)$ *- fluent* f *was observed to be true/false at step* i*, where* $0 \leq i \leq n$.
2. $happened(a, i)$ *- action* a *was observed to happen at step* i*, where* $0 \leq i < n$.

Example 2. [History] For our Example 1, a possible history may be:
$\Gamma_1 = \{\ observed(up(l_1), false, 0),$
$observed(up(l_2), true, 0),$
$happened(toggle(l_1), 0)\ \}$
This says that latch l_1 was initially down, latch l_2 was initially up and that the agent toggled l_1 at time step 0.

If l is a literal, by $obs(l, i)$ we will denote $observed(f, true, i)$ if $l = f$ and $observed(f, false, i)$ if $l = \neg f$. If p is a set of literals, $obs(p, i) = \{obs(l, i) : l \in p\}$. If for every fluent f in the signature either $observed(f, true, 0) \in \Gamma_n$ or $observed(f, false, 0) \in \Gamma_n$, then we say that *the initial situation of Γ_n is complete*. The semantics of a history Γ_n is given by the following definition:

Definition 4 (Model of a History). *(adapted from [16]) Let Γ_n be a history of a system description $\mathcal{D}$ up to time step n.*

(a) A trajectory $\langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$ of $\mathcal{T}(\mathcal{D})$ is a model of Γ_n *if:*
1. $a_i = \{a : happened(a, i) \in \Gamma_n\}$, $\forall\ 0 \leq i < n$
2. if $obs(l, i)$ then $l \in \sigma_i$, $\forall\ 0 \leq i \leq n$
(b) Γ_n is consistent *if it has a model.*
(c) A literal l holds *in a model M of Γ_n at time $i \leq n$ if $l \in \sigma_i$; Γ_n* entails *$h(l, i)$ if, for every model M of Γ_n, $M \models h(l, i)$.*
(We use $M \models h(l, i)$ to denote that l holds in model M, and $\Gamma_n \models h(l, i)$ to denote that Γ_n entails *$h(l, i)$.)*

Example 3. [Model of a History] The history Γ_1 in Example 2 is consistent. Its model is the trajectory:
$M = \langle \{\neg up(l_1), up(l_2), \neg open\},\ toggle(l_1),\ \{up(l_1), up(l_2),\ open\} \rangle$.
As well, $\Gamma_1 \models \neg holds(open, 0)$, $\Gamma_1 \models holds(open, 1)$, etc.

Note that a consistent history may have more than one model if non-deterministic actions are involved.

Now that we have presented the common part for the two methods, we continue presenting each one of them in detail.

3.1 Computing Trajectories Using ASP

The ASP method requires, besides the recorded history, two logic programs under the answer set semantics: one for encoding the system description of $\mathcal{AL}$ and another one to express the connection between the recorded history and the system description. We use the logic program Π_S described in section 2.2 to encode system descriptions of $\mathcal{AL}$. We now introduce a program, Ω_S, to represent the connection between observations in the history and the hypothetical relations *occurs* and *holds* in Π_S.

Definition 5. *(from [16]) If Γ_n is a history of system description $\mathcal{D}$ up to time step n, then by Ω_S we denote the program constructed as follows:*

1. *For every action a such that $happened(a, i) \in \Gamma_n$, Ω_S contains:*
   $occurs(a, i) \leftarrow happened(a, i)$.
2. *For every literal l such that $obs(l, 0) \in \Gamma_n$, Ω_S contains:*
   $h(l, 0) \leftarrow obs(l, 0)$.
3. *For every literal l such that $obs(l, i) \in \Gamma_n$, Ω_S contains the* reality check *axiom:*
   $\leftarrow obs(l, i),$
   $\quad not\ h(l, i).$

The ASP method of computing trajectories consists of finding the answer sets of the program $\Pi_S(\mathcal{D}) \cup \Gamma_n \cup \Omega_S$. Each of these answer sets will *define* a possible trajectory. Let us formally describe what "defining a sequence" means. (By $lit(P)$ we denote all the literals in the language of program P.)

Definition 6. *Let Γ_n be a history of $\mathcal{D}$ and A be a set of literals over $lit(\Pi_S(\mathcal{D}) \cup \Gamma_n \cup \Omega_S)$. We say that A* defines *the sequence*

$$\langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$$

if $\sigma_i = \{l \mid h(l, i) \in A\}$ for any $0 \leq i \leq n$, and $a_k = \{a \mid occurs(a, k)\}$ for any $0 \leq k < n$.

Example 4. [Trajectory Computation] For the system description in Example 1 and the history in Example 2, the answer set of $\Pi_S(\mathcal{D}) \cup \Gamma_1 \cup \Omega_S$ defines the trajectory

$$\langle \{\neg up(l_1), up(l_2), \neg open\}, toggle(l_1), \{up(l_1), up(l_2), open\} \rangle,$$

which, according to Example 3, is a model of Γ_1.

3.2 Computing Trajectories in $\mathcal{F}$LORA-2

We now present a method of computing trajectories using $\mathcal{F}$LORA-2. As before, we describe dynamic systems using $\mathcal{AL}$. We translate the $\mathcal{AL}$ system descriptions into $\mathcal{F}$LORA-2 logic programs. We cannot make use of the previous encoding, Π_S, because it includes rules with empty and disjunctive heads. Such rules have no well-founded semantics; as $\mathcal{F}$LORA-2 relies on the well-founded semantics, Π_S is not a $\mathcal{F}$LORA-2 logic program. We introduce a new program, Π_W, for the translation from $\mathcal{AL}$ into $\mathcal{F}$LORA-2 and use Π_W in the computation of trajectories.

Let $\mathcal{D}$ be a system description of $\mathcal{AL}$. The signature of $\Pi_W(\mathcal{D})$ will consist of the signature of $\Pi_S(\mathcal{D})$ together with the relations:

- $defeated(fluent, step)$ ($defeated(f, i)$ says that the non-static fluent f is defeated at time step i) and
- $defeated(n(fluent), step)$ ($defeated(n(f), i)$ says that $\neg f$ is defeated at time step i, where f is a non-static fluent).

By $\bar{l}$ we will denote $n(f)$ if $l = f$ or f if $l = \neg f$. The logic program $\Pi_W(\mathcal{D})$ is constructed as follows:[2]

1. For every static causal law "l **if** p" from $\mathcal{D}$, $\Pi_W(\mathcal{D})$ contains:

$$h(l, I) \leftarrow h(p, I). \tag{11}$$

   If l is a non-static literal, then $\Pi_W(\mathcal{D})$ also contains:

$$defeated(\bar{l}, I) \leftarrow h(p, I). \tag{12}$$

2. For every dynamic causal law "a **causes** l **if** p" from $\mathcal{D}$, $\Pi_W(\mathcal{D})$ contains:

$$\begin{aligned} h(l, I+1) \leftarrow\ & h(p, I), \\ & occurs(a, I). \end{aligned} \tag{13}$$

$$\begin{aligned} defeated(\bar{l}, I+1) \leftarrow\ & h(p, I), \\ & occurs(a, I). \end{aligned} \tag{14}$$

   (Note that based on the definition of inertial, defined and static fluents, the literal l appearing in the dynamic causal law above can only be an inertial literal).
3. For every executability condition "**impossible** $a_1, \ldots, a_k$ **if** p" from $\mathcal{D}$, and for every i such that $0 \leq i \leq k$, $\Pi_W(\mathcal{D})$ contains:

$$\begin{aligned} \neg occurs(a_i, I) \leftarrow\ & \text{not } \neg occurs(a_1, I), \\ & \ldots \\ & \text{not } \neg occurs(a_{i-1}, I), \\ & \text{not } \neg occurs(a_{i+1}, I), \\ & \ldots \\ & \text{not } \neg occurs(a_k, I), \\ & h(p, I). \end{aligned} \tag{15}$$

4. $\Pi_W(\mathcal{D})$ contains the Inertia Axioms:

$$\begin{aligned} holds(F, I+1) \ \leftarrow\ & fluent(\mathbf{inertial}, F), \\ & holds(F, I), \\ & \text{not } defeated(F, I+1). \\ \neg holds(F, I+1) \leftarrow\ & fluent(\mathbf{inertial}, F), \\ & \neg holds(F, I), \\ & \text{not } defeated(n(F), I+1). \end{aligned} \tag{16}$$

5. $\Pi_W(\mathcal{D})$ contains the CWA for defined fluents:

$$\begin{aligned} \neg holds(F, I) \leftarrow\ & fluent(\mathbf{defined}, F), \\ & \text{not } defeated(n(F), I). \end{aligned} \tag{17}$$

[2] The symbol for classical negation in $\mathcal{F}$LORA-2 is "neg". In this paper, we will use $\neg$ instead of "neg" for simplicity.

Next, we need to introduce the program connecting the observations in the history to the relations *holds* and *occurs*. We cannot use the program Ω_S as it contains a rule with empty head: the reality check, which is not allowed by the syntax of $\mathcal{F}$LORA-*2*. Hence, we construct a new program, Ω_W, by replacing the reality check

$$\leftarrow obs(l,i), \text{not } h(l,i).$$

by the rule

$$inconsistency \leftarrow obs(l,i), \text{not } h(l,i).$$

In our $\mathcal{F}$LORA-*2* method, we compute trajectories by finding the well-founded model of the program $\Pi_W(\mathcal{D}) \cup \Gamma_n \cup \Omega_W$. In the case of consistent histories with complete initial situations, the well-founded model will not contain the *inconsistency* predicate; it will be a subset of the literals entailed by the history.

4 Mathematical Analysis of the ASP and $\mathcal{F}$LORA-*2* Methods

In this section we discuss the soundness and completeness of the two methods with respect to the specification. We will apply these results when comparing the two methods in the next section. Let us assume that our dynamic domain is specified via the system description $\mathcal{D}$ of $\mathcal{AL}$ and consider a history Γ_n of $\mathcal{D}$. We limit ourselves to histories with complete initial situations.

First, we show that the ASP method is *sound* and *complete* with respect to the specification. This requires showing that there is a 1-to-1 correspondence between answer sets of the program $\Pi_S(\mathcal{D}) \cup \Gamma_n \cup \Omega_S$ and trajectories of Γ_n:

Theorem 1. *(Soundness and Completeness of the ASP Method) If Γ_n is a* consistent *history of $\mathcal{D}$ such that the initial situation of Γ_n is complete, then M is a model of Γ_n* **iff** *M is defined by some answer set of $\Pi_S(\mathcal{D}) \cup \Gamma_n \cup \Omega_S$.*

Proof. (sketch) We first prove the following: For every system description $\mathcal{D}$ there is a system description $\mathcal{D}'$ with the same signature as $\mathcal{D}$ minus the defined fluents, such that $\mathcal{D}'$ is a residue of $\mathcal{D}$ (i.e., restricting the states and actions of $\mathcal{T}(\mathcal{D})$ to the signature of $\mathcal{D}'$ establishes an isomorphism between $\mathcal{T}(\mathcal{D})$ and $\mathcal{T}(\mathcal{D}')$ [17]). Then, we apply Lemma 5 from [16] for $\mathcal{D}'$.

Next, we analyze the $\mathcal{F}$LORA-*2* method. We begin by showing that $\Pi_W(\mathcal{D}) \cup \Gamma_n \cup \Omega_W$ is computable:

Theorem 2. *(Computability in $\mathcal{F}$LORA-2) If Γ_n is a* consistent *history of $\mathcal{D}$ with a complete initial situation, then the program $\Pi_W(\mathcal{D}) \cup \Gamma_n \cup \Omega_W$ is computable by the $\mathcal{F}$LORA-2 inference engine.*

Proof. We need to show that the program is occur-check free, does not flounder and that the computation of the $\mathcal{F}$LORA-2 inference engine terminates. We assume that the $\mathcal{F}$LORA-2 computation is sound.[3] Let us use the notation $\Pi_W(\Gamma_n)$ for $\Pi_W(\mathcal{D}) \cup \Gamma_n \cup \Omega_W$.

(a) $\Pi_W(\Gamma_n)$ is *occur-check free*: $\Pi_W(\Gamma_n)$ is well-moded [18] for the following input-output specification:

$$\begin{array}{ll} holds(-,+) & fluent(+,-) \\ defeated(+,+) & occurs(+,+) \end{array}$$

As there is no rule in $\Pi_W(\Gamma_n)$ whose head contains more than one occurrence of the same variable in its output positions, then, based on the result of Apt and Pellegrini [19], $\Pi_W(\Gamma_n)$ is occur-check free.

(b) $\Pi_W(\Gamma_n)$ *does not flounder*: $\Pi_W(\Gamma_n)$ is well-moded for the input-output specification in (a) and all predicate symbols occurring under negation as failure (i.e., $defeated$ and $occurs$) are moded completely by input. Hence, based on results by Apt and Pellegrini [19], and Stroetman [20], $\Pi_W(\Gamma_n)$ does not flounder.

(c) The $\mathcal{F}$LORA-2 computation *terminates*: $\Pi_W(\Gamma_n)$ is a function-free program. The SLG resolution of XSB terminates for function-free programs [21] and $\mathcal{F}$LORA-2 is build on top of XSB.

We can now state that, for any consistent history with a complete initial situation, the trajectory computed by the $\mathcal{F}$LORA-2 method is sound with respect to the specification:

Theorem 3. *(Soundness of the $\mathcal{F}$LORA-2 Method) Let Γ_n be a consistent history of $\mathcal{D}$ with a complete initial situation, l be a fluent literal, and $0 \leq i \leq n$. If $h(l,i)$ is in the well-founded model of $\Pi_W \cup \Gamma_n \cup \Omega_W$, then*

$$\forall \text{ model } M \text{ of } \Gamma_n, M \models h(l,i)$$

Proof. (sketch) We use the notation $\Pi_W(\Gamma_n)$ for $\Pi_W \cup \Gamma_n \cup \Omega_W$ and $\Pi_S(\Gamma_n)$ for $\Pi_S \cup \Gamma_n \cup \Omega_S$. We have to show that the well-founded model W of $\Pi_W(\Gamma_n)$ is sound with respect to the specification. We prove it by showing that:

(a) Under the given conditions (i.e., Γ_n is a consistent history of $\mathcal{D}$ with a complete initial situation), $\Pi_W(\Gamma_n)$ has the same *answer sets* as $\Pi_S(\Gamma_n)$.

(b) Based on Corollary 5.7 from [3], the well-founded model W of $\Pi_W(\Gamma_n)$ is compatible with every answer set of $\Pi_W(\Gamma_n)$.

(c) From (a) and (b), W is compatible with every answer set of $\Pi_S(\Gamma_n)$.

[3] Explicit negation is not implemented yet in $\mathcal{F}$LORA-2 for programs that do not use the argumentation theory. Hence, "no consistency check is done to ensure that p and neg p are not true at the same time" [2]. However, as we only consider consistent histories and due to the way we encode the inertia axiom, p and neg p (i.e., $\neg p$, in the notation we adopted in this paper) can never be obtained simultaneously.

(d) From Theorem 1, every answer set of $\Pi_S(\Gamma_n)$ defines a trajectory of Γ_n.
(e) From (c) and (d), W is compatible with all possible trajectories of Γ_n.

We now continue discussing the completeness of the $\mathcal{F}$LORA-*2* method. In the general case, the $\mathcal{F}$LORA-*2* method is not complete. We will show an example when the $\mathcal{F}$LORA-*2* method is incomplete.

Example 5. [Incompleteness] Let $\mathcal{D}$ be the following system description:

$$a \textbf{ causes } f$$

$$\begin{aligned} \neg g_1 &\textbf{ if } f, g_2 \\ \neg g_2 &\textbf{ if } f, g_1 \\ d &\textbf{ if } g_1 \\ d &\textbf{ if } g_2 \end{aligned}$$

with inertial fluents f, g_1, and g_2, and defined fluent d. Let us consider the history $\Gamma_n = \{\ observed(f, false, 0)$, $observed(g_1, true, 0)$, $observed(g_2, true, 0)$, $happened(a, 0)\ \}$.

This system has two possible trajectories: in one of them f, g_1, and d are true and g_2 is false at step 1; in the other one f, g_2, and d are true and g_1 is false at step 1. However, the only information about step 1 that the well-founded model of $\Pi_W(\mathcal{D}) \cup \Gamma_n \cup \Omega_W$ can provide is that f holds in all possible trajectories of the system. The values of the remaining fluents, g_1, g_2, and d, are unknown.

Our goal is to find a class of system descriptions for which the $\mathcal{F}$LORA-*2* method is complete. We first introduce the following definition:

Definition 7. *A* simple *system description is a system description of $\mathcal{AL}$ such that:*

1. *there are no circular dependencies between fluents and*
2. *all executability conditions are only for single actions.*

We can now express the sufficient condition for completeness:

Theorem 4. *(Sufficient Condition for Completeness of the $\mathcal{F}$LORA-2 Method) Let $\mathcal{D}$ be a* simple *system description, let Γ_n be a consistent history of $\mathcal{D}$ with a complete initial situation, l be a fluent literal, and $0 \leq i \leq n$. If M is a model of Γ_n and $h(l, i) \in M$ then $h(l, i) \in W$, where W is the well-founded model of $\Pi_W(\mathcal{D}) \cup \Gamma_n \cup \Omega_W$.*

Proof. We first introduce some notation: For any logic program P, let P^* be the *general* logic program obtained from P by replacing all occurrences of $\neg pred$ by n_pred, for every predicate $pred$ from the signature of P. As before, by $\Pi_W(\Gamma_n)$ we denote the program $\Pi_W(\mathcal{D}) \cup \Gamma_n \cup \Omega_W$, and by $\Pi_S(\Gamma_n)$ we denote the program $\Pi_S(\mathcal{D}) \cup \Gamma_n \cup \Omega_S$.

We prove Theorem 4 by showing that the general logic program $\Pi_W(\Gamma_n)^*$ is a locally stratified program when $\mathcal{D}$ is a *simple* system description and Γ_n is a consistent history of $\mathcal{D}$ with a complete initial situation. From here, we obtain that $\Pi_W(\Gamma_n)$ has a unique answer set, equivalent to its well-founded model. Then, we show that the unique answer set of $\Pi_W(\Gamma_n)$ is equivalent to the unique answer set of $\Pi_S(\Gamma_n)$ modulo the common signature. As the unique answer set of $\Pi_S(\Gamma_n)$ is complete with respect to the specification (based on Theorem 1), so must be the well-founded model of $\Pi_W(\Gamma_n)$.

Let us now discuss how we show that $\Pi_W(\Gamma_n)^*$ is locally stratified. We do so by first defining the *graph of non-static fluents* $\mathcal{G}_{ns}(\mathcal{D}) = (V, E)$ as a directed graph where:

1. V is the set of all non-static (i.e., inertial or defined) fluent names from the signature of $\mathcal{D}$
2. For every static causal law "l **if** $p_1, \ldots, p_k$" from $\mathcal{D}$, where l is non-static, and for all $1 \leq i \leq k$ such that p_i is non-static, $(p_i, l) \in E$.

Given that $\mathcal{D}$ is a *simple* system description, $\mathcal{G}_{ns}(\mathcal{D}) = (V, E)$ is acyclic. We define the mapping $\alpha : V \rightarrow \{1, 2, \ldots, n\}$ as follows:

1. If $N \in V$ and N is a source, then: $\alpha(N) = 1$
2. For every $N \in V$ such that $(N_1, N) \in E$, ..., $(N_m, N) \in E$: $\alpha(N) = max\{\alpha(N_1), \ldots, \alpha(N_m)\} + 1$
3. $n = max\{\alpha(N) \mid N \text{ is a sink}\}$

Let $k = n + 1$. In order to prove that $\Pi_W(\Gamma_n)^*$ is locally stratified we use the mapping α defined above and the ranking, ρ, defined as follows:

$$
\begin{array}{lll}
\rho(happened(a,i)) & = 0 & \\
\rho(observed(f,true,i)) = \rho(observed(f,false,i)) & = 0 & \text{if } f \text{ is non-static} \\
\rho(f) = \rho(n\_f) & = 0 & \text{if } f \text{ is static} \\
\rho(occurs(a,i)) = \rho(n\_occurs(a,i)) & = k * (i+1) & \\
\rho(holds(f,i)) = \rho(n\_holds(f,i)) & = k * i + \alpha(f) & \text{if } f \text{ is non-static} \\
\rho(defeated(\bar{l},i)) & = k * i + (\alpha(l) - 1) &
\end{array}
$$

It is obvious that ρ is a local stratification for $\Pi_W(\Gamma_n)^*$.

The results in the theorems above clearly show a difference between the ASP and $\mathcal{F}$LORA-2 method. In the next section we further analyze the relationship between the two of them.

5 Comparative Analysis of the ASP and $\mathcal{F}$LORA-2 Methods

The two methods for computing trajectories rely on different non-monotonic formalisms that correspond to different intuitions. On the one hand, we have

the answer set semantics based on the principle of *belief*, which implies the possibility of multiple valid models. On the other hand, we have the well-founded semantics and hence the idea of a *unique* set of conclusions for *every* program. This distinction has direct implications for the properties of the two methods. In particular, it determines the completeness of the ASP method and the incompleteness, in the general case, of the $\mathcal{F}$LORA-*2* method. Depending on the reasoning task to be performed, the incompleteness of the $\mathcal{F}$LORA-*2* method can be seen as a limitation. This is especially the case when reasoning about the effects of non-deterministic actions, if we desire to know the consistent effects of these actions over all possible trajectories. Such effects are not detected by the $\mathcal{F}$LORA-*2* method. For instance, in Example 5 the fact that d holds at time step 1 is not inferable by $\mathcal{F}$LORA-*2*; however, d is an indirect effect of the non-deterministic action a, and d holds in every state following the execution of a in every trajectory that is a model of the history. The ASP method has an advantage here, as each possible trajectory of the system is defined by an answer set.

The $\mathcal{F}$LORA-*2* method might have an advantage over the ASP method in terms of efficiency. The SLG algorithm of XSB, the basis of the $\mathcal{F}$LORA-*2* inference engine, has polynomial time complexity for function-free programs [21]. The logic programs described in this paper are indeed function-free. However, with the efficiency of ASP solvers constantly improving, we cannot make strong claims here. We ran tests on 30 examples using the inference engine $\mathcal{F}$LORA-*2* and the answer set solver CLASP.[4] In most cases the two systems were equally efficient; only a minimal difference[5] was noted on three examples containing numerical constraints, where numbers ranged over a large set. New solvers integrating answer set reasoning with constraint solving techniques may annul this minimal advantage of $\mathcal{F}$LORA-*2*. This remains to be explored in the future.

In terms of applications of the two methods, we see an advantage for the ASP method. Substantial work has been done to investigate the suitability of ASP for solving reasoning problems related to dynamic domains and used in answering questions from natural language, for example planning [5], diagnosis [16], or the computation of preferred trajectories [22], [23]. Our ASP method can be easily extended with a planning or diagnosis module or a theory of intentions in order to accomplish those tasks. As far as we know, such work still remains to be done for $\mathcal{F}$LORA-*2*. Furthermore, although we were concerned in this paper only with consistent histories with a complete initial situation, the ASP approach can easily perform temporal projection for other types of histories. However, in the case of histories with an incomplete initial situation, the set of conclusions produced by $\mathcal{F}$LORA-*2* would be limited, due to the underlying formalism.

The use of $\mathcal{AL}$ as a specification language in both methods has two advantages. First of all, it determines a smaller class of logic programs for which it is easy to

[4] http://potassco.sourceforge.net

[5] 30 seconds versus 60 seconds on a machine with 1.73 GHz CPU and 1GB RAM running 32-bit Windows.

compare ASP and $\mathcal{F}$LORA-*2*. Focusing on this small class can, however, shed some light on the relationship between ASP and $\mathcal{F}$LORA-*2* in general, and can even contribute to the comparative study of their underlying formalisms, the answer set and the well-founded semantics. Secondly, the $\mathcal{F}$LORA-*2* programs presented here receive a specification independent from the computational technique that is used: in terms of a system description of $\mathcal{AL}$ and the transition diagram it describes. This gives trustworthiness to the system, as it ensures that whatever is computed is indeed correct.

6 Conclusions and Future Work

We presented two approaches for computing trajectories in dynamic domains: one based on ASP, and another one based on the language and inference engine $\mathcal{F}$LORA-*2*. We showed that both methods are sound, while only the ASP method is complete for the general case. We identified a class of what we called *simple* system descriptions for which the $\mathcal{F}$LORA-*2* method is also complete. Finally, we investigated the relationship between the two methods by showing some of their advantages and limitations. The $\mathcal{F}$LORA-*2* method can be used to perform temporal projection in the Digital Aristotle reasoning system.

Both methods start with the description of the domain in action language $\mathcal{AL}$ with defined fluents. The $\mathcal{AL}$ system description, together with a given history, is encoded as a logic program under the syntax of CLASP for the ASP method. The answer sets of this program define trajectories for the given history. For the $\mathcal{F}$LORA-*2* method, the input information is encoded as a logic program under the syntax of $\mathcal{F}$LORA-*2*; the conclusions that are inferred are compatible with all trajectories for that specific history.

The work presented here can continue in several directions. We plan to investigate the properties of the two methods for other types of histories (for example, inconsistent histories). We also intend to relax the sufficient condition for the completeness of the $\mathcal{F}$LORA-*2* method so that it would characterize a larger class of system descriptions. In particular, we plan to find a condition for the system description and history that would ensure that the resulting logic program is weakly stratified; we would then apply the result of Przymusinska and Przymusinski [24], which states that, for weakly stratified programs, the weakly perfect model is also well-founded and unique stable. Finally, the efficiency of the two methods can be investigated on examples illustrating a larger number of difficulties. Such results may be useful in comparing the two logic programming paradigms, ASP and $\mathcal{F}$LORA-*2*, and their underlying non-monotonic formalisms.

Acknowledgments. We would like to thank Michael Gelfond for helpful comments on the subject of this paper, and the anonymous reviewers for feedback.

References

1. The *Project Halo* web page, located at http://www.projecthalo.com (2010)

2. The $\mathcal{F}$LORA-*2* User's Manual, located at http://flora.sourceforge.net/docs/floraManual.pdf (2010)
3. Van Gelder, A., Ross, K. A., Schlipf, J.: The Well-founded Semantics for General Logic Programs. Journal of the ACM 38(3), 620–650 (1991)
4. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of the ACM 42, 741-843 (1995)
5. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press (2003)
6. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Proceedings of ICLP-88, pp. 1070–1080. (1988)
7. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9(3/4), 365–386 (1991)
8. Gelfond, M., Inclezan, D.: Yet Another Modular Action Language. In: Proceedings of SEA-09, pp. 64–78. University of Bath Opus: Online Publications Store (2009)
9. Turner, H.: Representing Actions in Logic Programs and Default Theories: A Situation Calculus Approach. Journal of Logic Programming 31(1-3), 245–298 (1997)
10. Baral, C., Gelfond, M.: Reasoning Agents in Dynamic Domains. In: Workshop on Logic-Based Artificial Intelligence, pp. 257–279. Kluwer Academic Publishers, Norwell (2000)
11. Hayes, P. J., McCarthy, J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 4, pp. 463–502. Edinburgh University Press, Edinburgh (1969)
12. Gelfond, M.: Representing Knowledge in A-Prolog. In: Kakas, A. C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II, pp. 413–451. Springer Verlag, Berlin (2002)
13. Lin, F.: Embracing Causality in Specifying the Indirect Effects of Actions. In: Proceedings of IJCAI-95, pp. 1985–1993. Morgan Kaufmann (1995)
14. Baral, C., Lobo, J.: Defeasible Specifications in Action Theories. In: Proceedings of IJCAI-97, pp. 1441–1446. Morgan Kaufmann Publishers (1997)
15. McCain, N., Turner, H.: A Causal Theory of Ramifications and Qualifications. Artificial Intelligence 32, 57–95 (1995)
16. Balduccini, M., Gelfond, M.: Diagnostic Reasoning with A-Prolog. Theory and Practice of Logic Programming 3(4), 425–461 (2003)
17. Erdoğan, S.T.: A Library of General-Purpose Action Descriptions. PhD thesis, The University of Texas at Austin (2008)
18. Dembinski, P., Maluszynski, J.: AND-Parallelism with Intelligent Backtracking for Annotated Logic Programs. In: Proceedings of SLP-85, pp. 29–38. Boston (1985)
19. Apt, K. R., Pellegrini, A.: On the Occur-Check-Free Prolog Programs. ACM Transactions on Programming Languages and Systems 16(3), 687–726 (1994)
20. Stroetman, K.: A completeness result for SLDNF resolution. Journal of Logic Programming 15, 337–357 (1993)
21. Chen, W., Warren, D. S.: Tabled Evaluation with Delaying for General Logic Programs. Journal of the ACM 43(1), 20–74 (1996)
22. Baral, C., Gelfond, M.: Reasoning about Intended Actions. In: Proceedings of AAAI-05, pp. 689–694. AAAI Press (2005)
23. Gelfond, M.: Going Places - Notes on a Modular Development of Knowledge about Travel. In: AAAI 2006 Spring Symposium Series, pp. 56–66 (2006)
24. Przymusinska, H., Przymusinski, T.: Weakly Stratified Logic Programs. Fundamenta Informaticae 13, 51–65 (1990)