Some Properties of System Descriptions of $\mathcal{AL}_d$

Michael Gelfond and Daniela Inclezan

Texas Tech University, USA

Abstract. The paper discusses some properties of system descriptions in the recent extension of action language $\mathcal{AL}$ (also known as $\mathcal{B}$) by defined fluents.

1 Introduction

In this paper we study some properties of system descriptions of action language $\mathcal{AL}_d$. The language, introduced in [3], expands the earlier action language $\mathcal{AL}$ [5, 1], (also known as $\mathcal{B}$) by fluents which are defined in terms of other fluents and are not directly influenced by actions. The new language allows a shorter and more convenient description of dependencies between fluents.

In what follows, we start with a brief review of $\mathcal{AL}_d$. We give a sufficient condition guarantying that a system description $\mathcal{D}$ of $\mathcal{AL}_d$ is well–founded, i.e., its states are fully determined by statics and inertial fluents. We show that, for system descriptions satisfying this condition, the introduction of defined fluents though convenient is not essential. To prove this, we introduce an algorithm that eliminates defined fluents from a description without substantially changing its meaning. Finally, we show how the sufficient condition for well–foundedness can be used to expand results from [6] to find a common core of $\mathcal{AL}_d$ and action language $\mathcal{C}^+$.

2 Action Language $\mathcal{AL}_d$

We start with a short review of $\mathcal{AL}_d$. The language is parametrized by an *action signature* which consists of three disjoint, non-empty sets of symbols: the set $\mathcal{S}$ of *statics*, the set $\mathcal{F}$ of *fluents* and the set $\mathcal{A}$ of *actions*. The set $\mathcal{F}$ is partitioned into the set $\mathcal{F}_i$ of *inertial fluents* and the set $\mathcal{F}_d$ of *defined fluents*. Elements of $\mathcal{S} \cup \mathcal{F}$ are called atoms. A literal is an atom p or its negation $\neg p$.

Definition 1. *[System Description of $\mathcal{AL}_d$]*
A *system description* of $\mathcal{AL}_d$ is a collection of the following statements:

1. Dynamic Causal Law

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

which says that *if action a were to be executed in a state of a dynamic system satisfying property*[1] *p then inertial literal l_i would be true in any resulting state.*

[1] By property we mean an arbitrary collection of literals.

2. State Constraint

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

which says that *every state satisfying property p must also satisfy literal l*. Here l cannot be a negative literal formed by a defined fluent.

3. Executability Condition

$$\textbf{impossible } a_0, \ldots, a_k \textbf{ if } p \tag{3}$$

which says that *actions $a_0, \ldots, a_k$ cannot be simultaneously executed in a state satisfying property p.*

A system description $\mathcal{D}$ serves as a specification of the transition diagram $\mathcal{T}(\mathcal{D})$ defining all possible trajectories of the dynamic system defined by $\mathcal{D}$. Therefore, to define the semantics of $\mathcal{AL}_d$, we have to precisely define the states and transitions of this diagram. In $\mathcal{AL}$ a state is simply defined as a complete and consistent collection of literals satisfying the state constraints of $\mathcal{D}$. In $\mathcal{AL}_d$ the situation is more subtle. To see the problem consider

Example 1. Let D_1 be a a system description with two inertial fluents, f and g and a defined fluent h whose meaning is given by the following state constraints:

$$h \textbf{ if } f$$
$$h \textbf{ if } \neg g$$

In general the collection of state constraints whose head is a defined fluent h is referred to as the *definition* of h. As expected, the truth of h follows from the truth of the body of at least one of its defining rules. Otherwise, h is false. The old definition of a state does not comply with this intuition. Clearly, $\{\neg f, g, h\}$ is a complete and consistent set of literals satisfying state constraints of $\mathcal{D}$ but intuitively it is not a state since the truth of h does not follow from any of its defining rules.

The definition of a state of $\mathcal{AL}_d$ given below exploits the close relationship between action languages and logic programs with two negations under the answer set semantics [4, 8]. We will need the following notation. By $P_c(\mathcal{D})$ (where c stands for constraints) we denote the logic program defined as follows:

1. For every state constraint (2) program $P_c(\mathcal{D})$ contains

$$l \leftarrow p.$$

2. For every defined fluent f program $P_c(\mathcal{D})$ contains

$$\neg f \leftarrow \textit{not } f.$$

For any set σ of literals σ_{nd} denotes the collection of all literals of σ formed by inertial fluents and statics. (The $_{nd}$ stands for non-defined.)

Definition 2. *[State]*
A complete and consistent set σ of literals is a *state* of $\mathcal{T}(\mathcal{D})$ if σ is the unique answer set of $P_c(\mathcal{D}) \cup \sigma_{nd}$.

It is not difficult to check that the transition diagram defined by system description D_1 from Example 1 has the following states: $\{f, g, h\}$, $\{f, \neg g, h\}$, $\{\neg f, \neg g, \neg h\}$, $\{\neg f, g, \neg h\}$. To check that $\sigma_0 = \{f, g, h\}$ is a state it is enough to check that σ_0 is the only answer set of $P_c(\mathcal{D}) \cup \{f, g\}$. Similarly for other states. To see that $\sigma = \{\neg f, g, h\}$ is not a state it suffices to see that σ is not the answer set of $P_c(\mathcal{D}) \cup \{\neg f, g\}$.

The next example demonstrates the importance of the uniqueness requirement of the definition.

Example 2. Let us consider a system description D_2 with two defined fluents f and g which are defined by the following mutually recursive laws:

$$g \textbf{ if } \neg f$$
$$f \textbf{ if } \neg g$$

Let us check if $\{f, \neg g\}$ is a state of $\mathcal{T}(D_2)$. Program $P_c(D_2)$ from Definition 2 consists of rules

$$g \leftarrow \neg f.$$
$$f \leftarrow \neg g.$$
$$\neg g \leftarrow \mathit{not}\ g.$$
$$\neg f \leftarrow \mathit{not}\ f.$$

Since all the fluents of D_2 are defined, $\sigma_{nd} = \emptyset$ and program $P_c(D_2) \cup \sigma_{nd}$ has two answer sets, $\{f, \neg g\}$ and $\{g, \neg f\}$. This violates the condition of Definition 2 and hence $\{f, \neg g\}$ is not a state. In fact, $\mathcal{T}(D_2)$ has no states.

Our definition of the transition relation of $\mathcal{T}(\mathcal{D})$ is also based on the notion of answer set. To describe a transition $\langle \sigma_0, a, \sigma_1 \rangle$ we construct a logic program $P(\mathcal{D}, \sigma_0, a)$ encoding system description $\mathcal{D}$, initial state σ_0, and set of actions a, such that answer sets of this program determine the states the system can move into after the execution of a in σ_0. To define the encoding $P(\mathcal{D})$ of $\mathcal{D}$ we need two steps, 0 and 1, which stand for the beginning and the end of a transition, relations $holds(f, i)$ – fluent f is true at step i, and $occurs(a, i)$ – action a occurred at step i. If l is formed by a fluent then a shorthand $h(l, i)$ denotes $holds(f, i)$ if $l = f$ and $\neg holds(f, i)$ if $l = \neg f$. If l is formed by a static then $h(l, i)$ is simply l. If p is a set of literals, then $h(p, i) =_{def} \{h(l, i) : l \in p\}$. I is a variable ranging over steps.

– For every causal law (1) $P(\mathcal{D})$ contains

$$h(l_i, 1) \leftarrow h(p, 0), occurs(a, 0) \tag{4}$$

– For every state constraint (2) $P(\mathcal{D})$ contains

$$h(l, I) \leftarrow h(p, I) \tag{5}$$

– $P(\mathcal{D})$ contains the CWA for every defined fluent f:

$$\neg holds(f, I) \leftarrow \mathit{not}\ holds(f, I) \tag{6}$$

– For every executability condition (3) $P(\mathcal{D})$ contains

$$\leftarrow h(p, I), occurs(a_0, I), \ldots, occurs(a_k, I) \tag{7}$$

– $P(\mathcal{D})$ contains the Inertia Axiom for every inertial fluent f

$$holds(f,1) \leftarrow holds(f,0), \mathit{not}\ \neg holds(f,1) \tag{8}$$

$$\neg holds(f,1) \leftarrow \neg holds(f,0), \mathit{not}\ holds(f,1) \tag{9}$$

This completes the construction of encoding $P(\mathcal{D})$ of system description $\mathcal{D}$. The encoding $h(\sigma_0, 0)$ of initial state σ_0 and the encoding $occurs(a, 0)$ of action a are defined as follows:

$$h(\sigma_0, 0) =_{def} \{h(l, 0) : l \in \sigma_0\}$$

$$occurs(a, 0) =_{def} \{occurs(a_i, 0) : a_i \in a\}.$$

Finally,

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

Definition 3. *[Transition]*
Let σ_0 be a state and a be an action. A triple $\langle \sigma_0, a, \sigma_1 \rangle$ is a *transition* of $\mathcal{T}(\mathcal{D})$ iff $P(\mathcal{D}, \sigma_0, a)$ has an answer set A such that $\sigma_1 = \{l : h(l, 1) \in A\}$.

3 Well–founded System Descriptions of $\mathcal{AL}_d$

Definition 4. *[Well–founded System Descriptions]*
A system description $\mathcal{D}$ of $\mathcal{AL}_d$ is called *well–founded* if for any complete and consistent set of fluent literals σ the program

$$P_c(\mathcal{D}) \cup \sigma_{nd} \tag{10}$$

has at most one answer set.

Intuitively, $\mathcal{D}$ is well–founded if its states are fully determined by the statics and inertial fluents of $\mathcal{D}$. To give a sufficient condition guarantying well–foundedness of $\mathcal{D}$ we need the following notions:

Definition 5. *[Fluent Dependency Graph]*
The *fluent dependency graph* of a system description $\mathcal{D}$ of $\mathcal{AL}_d$ is the directed graph such that

– its vertices are arbitrary literals,
– it has an edge
  - from l to l' if l is formed by a static or an inertial fluent and $\mathcal{D}$ contains a state constraint with the head l and the body containing l',
  - from f to l' if f is a defined fluent and $\mathcal{D}$ contains a state constraint with the head f and the body containing l' and not containing f.
  - from $\neg f$ to f for every defined fluent f.

Note that for system descriptions not containing defined fluents this definition coincides with that given in [6].

Definition 6. *[Weak Acyclicity]*
A fluent dependency graph is *weakly acyclic* if it does not contain paths from defined fluents to their negations. By extension, a system description with a weakly acyclic fluent dependency graph is also called *weakly acyclic*.

Proposition 1 *[Sufficient Condition for Well–foundedness]*
If a system description $\mathcal{D}$ of $\mathcal{AL}_d$ is weakly acyclic then $\mathcal{D}$ is well–founded.

It is easy to check that system description D_1 from Example 1 is weakly acyclic and well–founded while D_2 from Example 2 is neither weakly acyclic nor well–founded. Here are some additional examples.

Example 3. System description D_3 with defined fluents f and g and causal laws:

$$f \ \textbf{if}\ f, \neg g$$
$$g \ \textbf{if}\ \neg f$$

is weakly acyclic and hence well–founded; $\mathcal{T}(D_3)$ has one state, $\{\neg f, g\}$.

Example 4. System description D_4 with defined fluents f and g and causal laws:

$$f \ \textbf{if}\ g$$
$$g \ \textbf{if}\ f$$

D_4 is weakly acyclic and hence well–founded; $\mathcal{T}(D_4)$ has one state, $\{\neg f, \neg g\}$.

The next example shows that weak acyclicity is not necessary for well–foundedness.

Example 5. System description D_5 with inertial fluents f and g, defined fluent d and causal laws:

$$f \ \textbf{if}\ g$$
$$d \ \textbf{if}\ \neg f, g, \neg d$$

is not weakly acyclic but well–founded.

Proof of Proposition 1 (Sketch).

To prove the proposition we will need the following well known fact: Let P be a logic program and Q be obtained from P by removing every rule whose head occurs in its body. Then P and Q are equivalent.
Let $\mathcal{D}$ be as in Proposition 1 and σ be a complete and consistent set of literals. We need to show that the program

$$P =_{def} \Pi_c(\mathcal{D}) \cup \sigma_{nd}$$

has at most one answer set.

Let Q be the program obtained from P by eliminating all rules of the type " $f \leftarrow f, p$ " where f is a defined fluent. As mentioned above P and Q are equivalent, which means that it is enough to prove that Q has at most one answer set. In what follows, we abuse the notation and treat a negative literal $\neg f$ of Q as a new atom.

Let G be the (weakly acyclic) fluent dependency graph of description $\mathcal{D}$ and let G' be the dependency graph of program Q.[2] Let us mark edges of G going to a defined fluent f from $\neg f$ as negative and other edges as positive. It is obvious that now G' is identical to G. It is easy to see that any negative cycle of G' (and hence of G) must contain a link from $\neg f$ to f for some defined fluent f, and hence a path from f to $\neg f$. Since G is weakly acyclic this is impossible. This implies that Q is locally stratified and hence has at most one answer set.

4 Eliminating Defined Fluents from $\mathcal{AL}_d$ System Descriptions

In this section we show that defined fluents, though a convenient tool allowing to substantially shorten system descriptions of $\mathcal{AD}_d$, can be eliminated – at least in the case of weakly acyclic system descriptions.

For a precise formulation of this result we need the following definition.

Definition 7. *[Residue (from [2])]*
Let $\mathcal{D}$ and $\mathcal{D}'$ be system descriptions such that the signature of $\mathcal{D}'$ is part of the signature of $\mathcal{D}$. $\mathcal{D}'$ is a *residue* of $\mathcal{D}$ if 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 $\mathcal{T}(\mathcal{D})$.

The following algorithm eliminates an arbitrary defined fluent from a weakly acyclic system description in such a way that the resulting system description is a residue of the original one.

Definition 8. *[Defined Fluent Elimination Algorithm]*
Input: Let $\mathcal{D}$ be a system description of $\mathcal{AL}_d$ and let f be a defined fluent from the signature of $\mathcal{D}$, defined by the following laws:

$$\begin{array}{l} f \ \textbf{if} \ f, p_1 \\ \quad \ldots \\ f \ \textbf{if} \ f, p_k \\ f \ \textbf{if} \ \beta_1 \\ \quad \ldots \\ f \ \textbf{if} \ \beta_m \end{array}$$

such that $p_1, \ldots, p_k$ and $\beta_1, \ldots, \beta_m$ do not contain f. (Note that they will not contain $\neg f$ either because of the condition that $\mathcal{D}$ is weakly acyclic.) We call the set of rules

$$\begin{array}{l} f \ \textbf{if} \ \beta_1 \\ \quad \ldots \\ f \ \textbf{if} \ \beta_m \end{array}$$

the *relevant part of the f's definition*.

[2] We consider the direction of edges in the dependency graph of a logic program to be from the head literal to body literals of a rule.

Output: A system description $\mathcal{E}(\mathcal{D}, f)$ with the same signature as $\mathcal{D}$ with the exception of defined fluent f.

Algorithm: $\mathcal{E}(\mathcal{D}, f)$ is obtained from $\mathcal{D}$ by:

1. Removing f's definition from $\mathcal{D}$ and f from the signature
2. Replacing each law of the type α **if** f, Γ by the set of axioms:[3]

$$\begin{array}{l} \alpha \ \textbf{if} \ \beta_1, \Gamma \\ \ldots \\ \alpha \ \textbf{if} \ \beta_m, \Gamma \end{array}$$

3. Replacing each law of the type α **if** $\neg f, \Gamma$ by the set of axioms:

$$\begin{array}{l} \alpha \ \textbf{if} \ \chi_1, \Gamma \\ \ldots \\ \alpha \ \textbf{if} \ \chi_n, \Gamma \end{array}$$

where $\chi_1, \ldots, \chi_n$ are sets of literals such that the set of sets $\{\chi_1, \ldots, \chi_n\}$ is defined as $\{ \{\bar{l}_1, \ldots, \bar{l}_m\} \ : \ l_1 \in \beta_1 \wedge \ldots \wedge l_m \in \beta_m\}$ where $\bar{l}_i$ is the literal complementary to l_i (i.e., $\{\chi_1, \ldots, \chi_n\}$ is the set of all minimal sets of literals falsifying the *relevant part* of f's definition).

We illustrate the application of this algorithm by some examples.

Example 6. Let us compute $\mathcal{E}(D_3, f)$, where D_3 is the description from Example 3. The relevant part of f's definition is empty, which means that the only minimal set falsifying it is the empty set. Thus, $\mathcal{E}(D_3, f)$ is

$$g.$$

Consequently, $\mathcal{E}(\mathcal{E}(D_3, f), g)$ is an empty system description. Its transition diagram contains one state, which is empty.

Let us now compute $\mathcal{E}(D_3, g)$. The relevant part of g's definition is g **if** $\neg f$, which means that the only minimal set falsifying it is $\{f\}$. Thus, $\mathcal{E}(D_3, g)$ is

$$f \ \textbf{if} \ f, \neg g$$

and $\mathcal{E}(\mathcal{E}(D_3, g), f)$ is again the empty system description.

Example 7. Let us compute $\mathcal{E}(D_4, f)$, where D_4 is the description from Example 4. The relevant part of f's definition is f **if** g, which means that the occurrence of f in the body of g **if** f should be replaced by g. Thus, $\mathcal{E}(D_4, f)$ is

$$g \ \textbf{if} \ g$$

Also, $\mathcal{E}(\mathcal{E}(D_4, f), g)$ is an empty system description. Its transition diagram contains one state, which is empty.

[3] α can be an expression of the form (1) or (2) or (3):
(1) a **causes** l_i
(2) l
(3) $\mathbf{impossible}\ a_0, \ldots, a_k$.

Example 8. Let D_6 be a system description with one defined fluent f, five inertial fluents, a, b, c, d, e, and the laws:

$$\begin{array}{ll} f \textbf{ if } a, b & d \textbf{ if } f \\ f \textbf{ if } c & e \textbf{ if } \neg f \end{array}$$

Both laws in f's definition are relevant. There are two minimal sets falsifying f's definition: $\{\neg a, \neg c\}$ and $\{\neg b, \neg c\}$. Hence, $\mathcal{E}(D_6, f)$ is the system description consisting of the laws:

$$\begin{array}{ll} d \textbf{ if } a, b & e \textbf{ if } \neg a, \neg c \\ d \textbf{ if } c & e \textbf{ if } \neg b, \neg c \end{array}$$

The contribution of this algorithm to the elimination of defined fluents is described by the following lemma.

Lemma 1. *If the fluent dependency graph of $\mathcal{D}$ is weakly acyclic and f is a defined fluent then $\mathcal{E}(\mathcal{D}, f)$ is a residue of $\mathcal{D}$.*

Proof (Sketch). In order to prove that $\mathcal{E}(\mathcal{D}, f)$ is a residue of $\mathcal{D}$, we have to show that:

(1) There is a one-to-one correspondence between states of $\mathcal{E}(\mathcal{D}, f)$ and states of $\mathcal{D}$, i.e., for any complete and consistent set σ of literals, there is a one-to-one correspondence between answer sets of $P_c(\mathcal{D}) \cup \sigma_{nd}$ and answer sets of $P_c(\mathcal{E}(\mathcal{D}, f)) \cup \sigma_{nd}$.
(2) There is a one-to-one correspondence between transitions of $\mathcal{E}(\mathcal{D}, f)$ and transitions of $\mathcal{D}$, i.e., for any state σ_0 and action a, there is a one-to-one correspondence between answer sets of $P(\mathcal{D}, \sigma_0, a)$ and answer sets of $P(\mathcal{E}(\mathcal{D}, f), \sigma_0, a)$.

In what follows, we expand the proof of (1); (2) is proven using similar techniques. Let σ be an arbitrary complete and consistent set of literals and let

$$\begin{array}{l} P_1 =_{def} P_c(\mathcal{D}) \cup \sigma_{nd} \\ P_2 =_{def} P_c(\mathcal{E}(\mathcal{D}, f)) \cup \sigma_{nd}. \end{array}$$

We want to show that:

(a) If A_1 is an answer set of P_1 and A_2 is defined as

$$A_2 =_{def} A_1 \setminus \{f, \neg f\}$$

then A_2 is an answer set of P_2.
(b) If A_2 is an answer set of P_2 and A_1 is defined as

$$A_1 =_{def} \begin{cases} A_2 \cup \{f\} & \text{if } \exists \beta_i, 1 \leq i \leq m \text{ such that } \beta_i \subset A_2 \\ A_2 \cup \{\neg f\} & \text{otherwise} \end{cases}$$

then A_1 is an answer set of P_1.

Based on $\mathcal{D}$'s weak-acyclicity, Proposition 1, and Definition 4, if A_1 is an answer set of P_1 then it is its *unique* answer set. Hence, the above conditions (a) and (b) are enough to

prove the one-to-one correspondence between answer sets of P_1 and P_2. We give here only the proof of (a); (b) is proven using the same standard approach.

To prove (a), we need to show that A_2 is a minimal set closed under the rules of $P_2^{A_2}$. First, we compute the reducts $P_1^{A_1}$ and $P_2^{A_2}$. By P we denote their common part, which consists of rules not containing f nor $\neg f$.

$P_1^{A_1}$ consists of P and

- rules encoding the definition of f

$$\begin{array}{l} f \leftarrow f, p_1. \\ \quad \ldots \\ f \leftarrow f, p_k. \\ f \leftarrow \beta_1. \\ \quad \ldots \\ f \leftarrow \beta_m. \end{array} \tag{11}$$

- $\neg f$ if $f \notin A_1$, and
- rules of the type

$$l \leftarrow f, p. \tag{12}$$

and

$$l \leftarrow \neg f, p. \tag{13}$$

where $l \neq f$ and $l \neq \neg f$. To simplify the presentation, we can assume that $f \notin p$ and $\neg f \notin p$.

$P_2^{A_2}$ consists of P and

- the rules

$$\begin{array}{l} l \leftarrow \beta_1, p. \\ \quad \ldots \\ l \leftarrow \beta_m, p. \end{array} \tag{14}$$

for every rule of the form (12) in $P_1^{A_1}$ and

- the rules

$$\begin{array}{l} l \leftarrow \chi_1, p. \\ \quad \ldots \\ l \leftarrow \chi_n, p. \end{array} \tag{15}$$

for every rule of the form (13) in $P_1^{A_1}$, with $\chi_1, \ldots, \chi_n$ defined as in Definition 8.

Part I. A_2 is closed under the rules of $P_2^{A_2}$.
Case 1: A_2 is closed under rules from P.
Such rules also appear in $P_1^{A_1}$. As A_1 is an answer set of P_1, A_1 must be closed under rules of this type. Based on how A_2 is constructed from A_1 and the fact that the rules of P do not contain f nor $\neg f$, A_2 is also closed under such rules.

Case 2: A_2 is closed under rules of the type $l \leftarrow \beta_i, p$ where $1 \leq i \leq m$.
We have to show that, if $\beta_i \subseteq A_2$ and $p \subseteq A_2$, then $l \in A_2$.
If $p \subseteq A_2$ then $p \subseteq A_1$. If $\beta_i \subseteq A_2$ then $\beta_i \subseteq A_1$ and, based on the definition of f in $P_1^{A_1}$ and the fact that A_1 is closed under the rules of $P_1^{A_1}$, $f \in A_1$.
$P_1^{A_1}$ must contain a rule

$$l \leftarrow f, p.$$

and A_1 is closed under this rule. As we have shown that $f \in A_1$ and $p \subseteq A_1$, this means that $l \in A_1$. Since $l \neq f, l \neq \neg f, l \in A_2$.
Case 3: A_2 is closed under rules of the type $l \leftarrow \chi_i, p.$ where $1 \leq i \leq n$.
We have to show that, if $\chi_i \subseteq A_2$ and $p \subseteq A_2$, then $l \in A_2$.
If $p \subseteq A_2$ then $p \subseteq A_1$. If $\chi_i \subseteq A_2$ then $\chi_i \subseteq A_1$. Hence, none of the rules in the definition of f in $P_1^{A_1}$ is satisfied, which means that $f \notin A_1$ and $\neg f \in A_1$.
$P_1^{A_1}$ must contain a rule

$$l \leftarrow \neg f, p.$$

and A_1 is closed under this rule. As we have shown that $\neg f \in A_1$ and $p \subseteq A_1$, this means that $l \in A_1$. Since $l \neq f, l \neq \neg f, l \in A_2$.

Part II. A_2 is minimal.
We prove this by assuming that there exists a set $S \subseteq A_2$ such that S is closed under the rules of $P_2^{A_2}$ and showing that $A_2 \subseteq S$, i.e., $\forall l \in A_2, l \in S$.

We assume that, for every predicate pr in the signature of P_1, $\neg pr$ is a new symbol. Hence, $P_1^{A_1}$ is a definite program. Let N be the $T_{P_1^{A_1}}$ operator. For every $l \in lit(P_1^{A_1})$, let

$$\delta(l) = n \text{ if } l \notin N^{n-1}(\emptyset) \text{ and } l \in N^n(\emptyset).$$

Since $P_1^{A_1}$ is definite,

$$\exists \omega \text{ such that } \forall l \in lit(P_1^{A_1}), \delta(l) \leq \omega.$$

We want to prove that

$$\forall l \in A_2, \text{if } \delta(l) = n \leq \omega \text{ then } l \in S.$$

We prove this by induction of n.

Base Case: $n = 0$ (Trivially true)
Induction Step: We assume that $0 < n \leq \omega$ and that

$$\forall l \in A_2, \text{if } \delta(l) < n \text{ then } l \in S.$$

We want to show that

$$\forall l \in A_2, \text{if } \delta(l) = n \text{ then } l \in S.$$

Let $l \in A_2$ such that $\delta(l) = n$. We will show that $l \in S$.
Case 1: l was derived from a rule $l \leftarrow p.$ in P.

Since $P \subset P_1^{A_1}$ and $\delta(l) = n$, then $\delta(p) < n$ and, based on the inductive hypothesis, $p \subseteq S$. Since S is closed under the rules of $P_2^{A_2}$, which includes the rules in P, then $l \in S$.
Case 2: l was derived from a rule $l \leftarrow f, p.$ in $P_1^{A_1}$.
Since $\delta(l) = n$, then $\delta(p) < n$ and, based on the inductive hypothesis, $p \subseteq S$. Also, we obtain that $\delta(f) < n$, which means that $\exists \beta_i, 1 \leq i \leq m$, such that $\delta(\beta_i) < n - 1$ and hence $\beta_i \subseteq S$. $P_2^{A_2}$ must contain a rule $l \leftarrow \beta_i, p.$ and, since $p \subseteq S$, $\beta_i \subseteq S$, and S is closed under the rules of $P_2^{A_2}$, then $l \in S$.
Case 3: l was derived from a rule $l \leftarrow \neg f, p.$ in $P_1^{A_1}$.
First, let us notice that $\delta(x) = 0$ if x is an inertial literal such that $x \in \sigma$ or x is a negative defined literal such that $x \in A_1$.
Since $\delta(l) = n$, then $\delta(p) < n$ and, based on the inductive hypothesis, $p \subseteq S$. From $\delta(l) = n$, we also obtain that $\delta(\neg f) < n$. In fact, as noticed above $\delta(\neg f) = 0$. Since $\neg f$ is derived, then $\neg f \in A_1$, which means that $\exists \chi_i, 1 \leq i \leq n$ such that $\chi_i \subseteq A_1$.
$P_2^{A_2}$ must contain the rule $l \leftarrow \chi_i, p.$ From here there are two possibilities:

(i) $\forall x \in \chi_i$, $\delta(x) < n$ and hence $\chi_i \subseteq S$. Based on the fact that S is closed under the rules of $P_2^{A_2}$, $l \in S$.

(ii) $\exists x \in \chi_i$ such that $\delta(x) \geq n$. We will show that this leads to a contradiction. Based on our initial observation, $\delta(x) \geq n$ implies that x must be a positive defined literal. Also, as $\delta(l) = n \leq \delta(x)$, x must depend on l, i.e., there must be a path from x to l in the fluent dependency graph of $\mathcal{D}$. Since the fluent dependency graph must contain edges from l to $\neg f$, from $\neg f$ to f, from f to $\overline{x} = \neg x$, and from $\neg x$ to x, this would mean that there is a path from f to $\neg f$ in the fluent dependency graph, which contradicts the fact that $\mathcal{D}$ is weakly acyclic.

This concludes the brief presentation of the proof of Lemma 1.

We can now formulate the following result.

Proposition 2 *For every weakly acyclic system description $\mathcal{D}$ there is a system description $\mathcal{D}'$ whose signature does not contain defined fluents, such that $\mathcal{D}'$ is a residue of $\mathcal{D}$.*

Proof (Sketch). Let $\pi = (f_1, \ldots, f_n)$ be a possible enumeration of the set $\mathcal{F}_d$ of defined fluents of $\mathcal{D}$. Let $\mathcal{D}'$ be the system description resulting from the successive elimination of defined fluents, i.e.

$$\mathcal{D}' =_{def} \mathcal{E}(\ldots \mathcal{E}(\mathcal{E}(\mathcal{D}, f_1), f_2) \ldots, f_n)$$

By applying Lemma 1 successively, we obtain that $\mathcal{D}'$ is a residue of $\mathcal{D}$.

5 Common Core of $\mathcal{AL}_d$ and $\mathcal{C}^+$

In this section we use the results from the previous sections to investigate the relationship between $\mathcal{AL}_d$ and another popular action language $\mathcal{C}^+$ [9]. In particular we describe their *common core* – a subset of $\mathcal{AL}_d$ which allows a simple equivalent transformation into $\mathcal{C}^+$. This generalizes recent results identifying such a common core for $\mathcal{AL}$ and $\mathcal{C}$ [5] of which $\mathcal{AL}_d$ and $\mathcal{C}^+$ are extensions.

We start with briefly describing a subset of $\mathcal{C}^+$ sufficient for our purpose. It has symbols for actions, statics (called rigid constants), inertial and (statically) defined fluents, and causal laws of the form

$$\textbf{caused } l_i \textbf{ if } p' \textbf{ after } a, p \tag{16}$$

$$\textbf{caused } l \textbf{ if } p \tag{17}$$

where l_i is an inertial fluent literal, l is an arbitrary literal, a is an action, and p', p are collections of literals. A system (or action) description of our fragment of $\mathcal{C}^+$ is a collection of statements of the form 16 and 17 above, including statements

$$\begin{array}{llllll} \textbf{caused} & f & \textbf{if} & f & \textbf{after} & f \\ \textbf{caused} & \neg f & \textbf{if} & \neg f & \textbf{after} & \neg f \end{array}$$

for every inertial fluent f and

$$\textbf{caused } \neg f \textbf{ if } \neg f$$

for every statically defined fluent f. The semantics of $\mathcal{C}^+$ is based on the distinction between what is true and what has a cause. The law (16) says that if action a were to be executed in a state satisfying property p then there would be a cause for l_i in any resulting state. The law (17) says that there is a cause for l to hold in a state which satisfies p. The semantics of this fragment will be given directly in terms of logic programming. Readers interested in the original semantics and the proof of equivalence of both approaches are referred to [9]. To define states of a system description $\mathcal{D}$ of $\mathcal{C}^+$ we need a program, $R_c(\mathcal{D})$, defined as follows:

– For every static law (17) with $p = l_1, \ldots, l_n$, the program $R_c(\mathcal{D})$ contains

$$l \leftarrow \mathit{not}\ \bar{l}_1, \ldots, \mathit{not}\ \bar{l}_n$$

where $\bar{l}_i$ is the literal complimentary to l_i.

Definition 9. *[State]*
A complete and consistent set σ of domain literals is a *state* of $\mathcal{T}(\mathcal{D})$ if σ is an answer set of $R_c(\mathcal{D}) \cup \sigma_{nd}$.

A transition $\langle \sigma_0, a, \sigma_1 \rangle$ is defined by constructing a program $R(\mathcal{D})$:

– For every causal law (16) with $p' = l_1, \ldots, l_n$, $R(\mathcal{D})$ contains

$$h(l, 1) \leftarrow \mathit{not}\ h(\bar{l}_1, 1), \ldots, \mathit{not}\ h(\bar{l}_n, 1), occurs(a, 0), h(p, 0). \tag{18}$$

– For every static law (17) where $p = l_1, \ldots, l_n$, $R(\mathcal{D})$ contains

$$h(l, I) \leftarrow \mathit{not}\ h(\bar{l}_1, I), \ldots, \mathit{not}\ h(\bar{l}_n, I) \tag{19}$$

Definition 10. *[Transition]*
A transition $\langle \sigma_0, a, \sigma_1 \rangle$ of a system description $\mathcal{D}$ of $\mathcal{C}^+$ is in $\mathcal{T}(\mathcal{D})$ iff $R(\mathcal{D}, \sigma_0, a)$ has an answer set A such that $\sigma_1 = \{l : h(l, 1) \in A\}$.

To identify a common core of $\mathcal{AL}_d$ and $\mathcal{C}^+$ we define the $\mathcal{C}^+$–image of action description $\mathcal{D}$ of $\mathcal{AL}_d$, which is obtained by

– replacing statements (1) by

$$\textbf{caused } l_i \textbf{ if } true \textbf{ after } a, p \tag{20}$$

– replacing statements (2) by

$$\textbf{caused } l \textbf{ if } p \tag{21}$$

– replacing statements (3) by

$$\textbf{caused } \bot \textbf{ if } true \textbf{ after } a_0, \ldots, a_k, p \tag{22}$$

– adding inertia axioms

$$\begin{array}{l} \textbf{caused } \ f \ \textbf{ if } \ f \ \textbf{ after } \ f \\ \textbf{caused } \ \neg f \ \textbf{ if } \ \neg f \ \textbf{ after } \ \neg f \end{array} \tag{23}$$

for every inertial fluent f

– adding closed world assumption for defined fluents

$$\textbf{caused } \neg f \textbf{ if } \neg f \tag{24}$$

for every defined fluent f.

The $\mathcal{C}^+$–image of $\mathcal{D}$ is not always equivalent to $\mathcal{D}$, i.e., the two system descriptions do not always represent the same transition diagram. However, as shown in [6], the equivalence holds if $\mathcal{D}$ contains no defined fluents and has an acyclic fluent dependency graph. System description D_2 from Example 2 shows that this result does not hold for system descriptions containing defined fluents. The transition diagram of $\mathcal{C}^+$–image of D_2 has two states, $\{f, \neg g\}$ and $\{\neg f, g\}$, while the transition diagram of D_2 has none. The difference is caused by the absence of the uniqueness requirement in Definition 9.

To guarantee that states of $\mathcal{D}$ and its $\mathcal{C}^+$–image coincide we introduce the following definition.

Definition 11. *[Common Core]*
A system description $\mathcal{D}$ of $\mathcal{AL}_d$ *belongs to the common core of* $\mathcal{AL}_d$ *and* $\mathcal{C}^+$ if its fluent dependency graph G satisfies the following conditions:

– G is weakly acyclic.
– The extension of G by edges from f to l' for every defined fluent f and literal l' such that f **if** $\Gamma \in \mathcal{D}$, $f \in \Gamma$, and $l' \in \Gamma$ is an acyclic graph.

Proposition 3 *A system description of* $\mathcal{AL}_d$ *that belongs to the common core of* $\mathcal{AL}_d$ *and* $\mathcal{C}^+$ *is equivalent to its* $\mathcal{C}^+$*–image.*

Proof (Sketch). Let $\mathcal{D}$ be an $\mathcal{AL}_d$ system description that belongs to the common core of $\mathcal{AL}_d$ and $\mathcal{C}^+$. By the first bullet of Definition 11, $\mathcal{D}$ is weakly acyclic. By Proposition 1, $\mathcal{D}$ is well–founded. As $\mathcal{D}$ is well–founded, it is clear that the transition diagrams described by $\mathcal{D}$ and its $\mathcal{C}^+$–image have the same states. We only need to show that they also have the same transitions. This can be shown by applying techniques used in [6].

Clearly the system descriptions D_2, D_3, and D_4 from Examples 2, 3 and 4 do not belong to the common core. One can easily check that the above transformation does not maintain the systems' equivalence; D_1 belongs to the core and hence its $\mathcal{C}^+$–image is equivalent to D_1. Equivalence is also preserved for D_5 but this does not follow from our proposition.

6 Conclusions and Future Work

In this paper, we have given a sufficient condition guarantying that states of an $\mathcal{AL}_d$ system description are fully determined by statics and inertial fluents. In system descriptions satisfying this condition, defined fluents are not essential and can be eliminated; they simply facilitate the description of dynamic domains. We have shown how our sufficient condition can be used to identify a common-core of action languages $\mathcal{AL}_d$ and $\mathcal{C}^+$. This is an expansion of the work in [6] where a common-core of languages $\mathcal{B}$ and $\mathcal{C}$ was identified. We hope that our result will contribute to a future comparison between modular action languages $\mathcal{ALM}$ [3] and MAD [7], which extend $\mathcal{AL}_d$ and $\mathcal{C}^+$, respectively.

Our sufficient condition for well–foundedness can be further relaxed to include, for instance, system descriptions like the one in Example 5. The system description D_5 in this example is well–founded, but it is not weakly acyclic because of the state constraint

$$d \textbf{ if } \neg f, g, \neg d$$

defining d. Let us note, however, that the definition of d is in fact a constraint saying that $\neg f$ and g cannot both be true in a state. Similarly for the system description D_7 consisting of the laws

$$\begin{array}{l} f \textbf{ if } g. \\ h \textbf{ if } \neg d. \\ d \textbf{ if } \neg f, g, h. \end{array}$$

where d is a defined fluent and f, g, h are inertial.

6.1 Acknowledgements

We would like to acknowledge the support of NASA grant #NNX1OAI86G and NSF grant IIS-1018031 and thank the anonimous referee for useful comments. But most of all we would like to thank David Pearce whom this Festrshrift is honoring. Both authors learned a lot from David's wonderful work and both hope to continue doing that for many years. The older author also like to thank David for the privilige of his friendship and for wonderful time he spent with David and his family. Happy Birthday David!

References

1. Chitta Baral and Michael Gelfond. Reasoning Agents In Dynamic Domains. In *Workshop on Logic-Based Artificial Intelligence*. Kluwer Academic Publishers, Jun 2000.
2. Selim Turhan Erdoğan. *A Library of General-Purpose Action Descriptions*. PhD thesis, University of Texas at Austin, Austin, TX, USA, 2008.
3. Michael Gelfond and Daniela Inclezan. Yet another modular action language. In *In Proceedings of the Second International Workshop on Software Engineering for Answer Set Programming10*, pages 64–78, 2009.
4. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.
5. Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on AI*, 3(16):193–210, 1998.
6. Michael Gelfond and Vladimir Lifschitz. The common core of the action languages B and C. In *Proceedings of the 14th International Workshop on Non-Monotonic Reasoning (NMR'2012)*, 2012.
7. Vladimir Lifschitz and Wanwan Ren. A modular action description language. Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI), pages 853–859, 2006.
8. David Pearce and Gerd Wagner. Logic programs with strong negation. In *Extensions of Logic Programming*, volume 475 of *Lecture Notes in Computer Science*, pages 311–326, 1991.
9. Hudson Turner, Vladimir Lifschitz, Norman McCain, Joohyung Lee, and E. Giunchiglia. Nonmonotonic causal theories. *Artificial Intelligence*, 2003.