Identifying Deterministic Action Descriptions: a Sufficient Condition

by
Marcello Balduccini

February 25, 2005
Revised on March 6, 2005

Talk Outline

⇒ **Introduction**

- Action Language AL
- Sufficient condition for determinism

Goal

To find a simple algorithmic condition that guarantees that an action description is deterministic.

Complex Task $\Rightarrow$ we will be satisfied with a <u>sufficient</u> condition.

Domain Models

- We model domains of interest by *transition diagrams* (nodes $\Rightarrow$ states, arcs $\Rightarrow$ actions).
- Transition diagrams describe the changes of state caused by execution of actions.
- Transition diagrams are concisely encoded by *action descriptions*.

Talk Outline

- Introduction

⇒ **Action Language AL**

- Sufficient condition for determinism

Action Language AL: Syntax

We focus on the Action Description Component of AL.

- *Fluent*: relevant property of the domain.
- **Action Signature** $\langle F, A \rangle$:
  - ◇ F: set of fluents.
  - ◇ A: set of *elementary* actions.
- *Fluent Literal*: fluent f and its negation, $\neg f$.
- *(Compound) Action*: a set, $\{a_1, \ldots, a_k\}$, of elementary actions.

Statements: Dynamic Laws

$$d : a \text{ causes } l_0 \text{ if } l_1, \ldots, l_n \tag{1}$$

"If a were to be executed in a state in which $l_1, \ldots, l_n$ hold, l_0 would be caused to hold in the resulting state."

where:

- d: name the dynamic law.
- a: (compound) action.
- l_i's: fluent literals.

Statements: Other Laws

State Constraints:

$$s : \text{caused } l_0 \text{ if } l_1, \ldots, l_n \tag{2}$$

"In every state, the truth of $l_1, \ldots, l_n$ is sufficient to cause the truth of l_0."

Impossibility/Executability Conditions:

$$b : a \text{ impossible\_if } l_1, \ldots, l_n \tag{3}$$

"a cannot be performed (is impossible, not executable) in any state in which $l_1, \ldots, l_n$ hold."

Action Description

Action Description: a tuple $\langle \Sigma, L \rangle$, where:

- Σ: action signature.
- L: set of laws from Σ.

We normally use L to implicitly define Σ.

Terminology

Given a dynamic law, w:

$$d : a \text{ causes } l_0 \text{ if } l_1, \ldots, l_n$$

- $name(w) = d$.
- $head(w) = l_0$.
- $trigger(w) = a$.
- $body(w) = \{l_1, \ldots, l_n\}$.

Similarly for other laws ($trigger(w) = \emptyset$ and $head(w) = \epsilon$ when not applicable).

Action Language AL: Semantics

Given by defining the successor state *for each transition* $\langle \sigma_0, a, \sigma_1 \rangle$ *in the transition diagram.*

- set of fluent literals S is *closed under state constraint* w if:

$$body(w) \subseteq S \rightarrow head(w) \in S.$$

- $Cn_Z(S)$ (**consequences of** S **under** Z): smallest set of fluent literals that *contains* S and *is closed under* Z.
- **State**: *complete* and *consistent* set of fluent literals closed under the state constraints of action description AD.

AL Semantics (cont'd)

- $E(a,\sigma)$ (*direct effects of a in state σ*):

  $E(a,\sigma) =$
  $\{head(w) \mid trigger(w) \subseteq a, body(w) \subseteq \sigma, w$ dynamic law of $AD\}$

- **Transition Diagram of** AD $(trans(AD))$: directed graph, $\langle N, R\rangle$, such that:
  - N: collection of all states of AD.
  - R: set of all transitions $\langle \sigma_0, a, \sigma_1\rangle$ such that a is executable in σ_0, and

    $$\sigma_1 = Cn_Z(E(a,\sigma_0) \cup (\sigma_1 \cap \sigma_0))$$

    (Z: set of state constraints from AD).

Deterministic Action Descriptions

Definition 1 *AD is* **deterministic** *if:*

$$\langle \sigma_0, a, \sigma_1 \rangle, \langle \sigma_0, a, \sigma_2 \rangle \in trans(AD) \iff \sigma_1 = \sigma_2.$$

Example.

$$\begin{cases} \text{caused } p \text{ if } \neg q, r \\ \text{caused } q \text{ if } \neg p, r \\ a \text{ causes } r \end{cases}$$

is non-deterministic. In fact, there are two successor states for action a in state $\{\neg p, \neg q, \neg r\}$:

$$\{\neg p, q, r\} \text{ and } \{p, \neg q, r\}.$$

Talk Outline

- Introduction
- Action Language AL

$\Rightarrow$ **Sufficient condition for determinism**

Dependency Graph

Definition 2 (Dependency graph ($dep(AD)$)) *A directed graph* $\langle FL, C \rangle$*:*

- FL*: fluent literals of* AD*.*
- C*: set of* 1*-arcs and* $+$*-arcs. For every state constraint* w*:*
  - ◇ *if* $body(w) = \{l\}$*, then* $\langle head(w), 1, l \rangle \in dep(AD)$*.*
  - ◇ *if* $|body(w)| > 1$*, then for every* $l_i \in body(w)$*,* $\langle head(w), +, l_i \rangle \in dep(AD)$*.*

Dependency Paths in $dep(AD)$

Definition 3 (Dependency path in $dep(AD)$) *A sequence*

$$\pi = \langle l_1, t_1, l_2, t_2, \ldots, t_{k-1}, l_k \rangle \qquad (k > 1)$$

such that, for every $1 \leq i < k$, $\langle l_i, t_i, l_{i+1} \rangle \in dep(AD)$.

- **Notation:** $\pi^s = l_1$; $\pi^e = l_k$; $|\pi| = k$ (nodes in π).
- *Arcs' labels omitted from arcs and paths when possible* (e.g. $\langle l_1, l_2, \ldots, l_k \rangle$).
- **Terminology:** π is *conditional* if it contains one or more $+$-arcs.

Sequences Through Negation

Definition 4 (Sequence through negation (neg-seq) in $dep(AD)$)
A non-empty sequence, $\nu = \langle \pi_1, \ldots, \pi_k \rangle$, such that:

- *every π_i is a dependency path.*
- *For every $1 \leq i < k$:*

  $\pi^s_{i+1} = \overline{\pi^e_i}$. *($\overline{\pi^e_i}$ denotes complement of π^e_i.)*

Terminology: $dep(AD)$ *contains* ν.

Loops Through Negation and Safety

Definition 5 (Dependency loop through negation (neg-loop)) *A neg-seq, $\langle \pi_1, \ldots, \pi_k \rangle$, such that*

$$\pi_1^s = \overline{\pi_k^e}.$$

Definition 6 (Conditional neg-seq or neg-loop) *A neg-seq (resp., neg-loop) $\langle \pi_1, \ldots, \pi_k \rangle$ such that every π_i is conditional.*

Definition 7 (Safe Dependency Graph) $dep(AD)$ *is* **safe** *if it does not contain any conditional neg-loop.*

Sufficient Condition for Determinism

Theorem 1 *For every action description, AD, if $dep(AD)$ is safe, then AD is deterministic.*

Lemmas

Lemma 1 *For every* $\langle \sigma_0, a, \sigma_1 \rangle$, $\langle \sigma_0, a, \sigma_2 \rangle \in trans(AD)$ *(*$\sigma_1 \neq \sigma_2$*) and every* $l \in \sigma_1 \setminus \sigma_2$ *such that* $l \notin \sigma_0$*, there exists an arc* $\langle l, l' \rangle$ *in* $dep(AD)$ *such that* $l' \in \sigma_1 \setminus \sigma_2$*.*

Proof. $l \notin E(a, \sigma_0)$. *In fact,* $E(a, \sigma_0) \subseteq \sigma_2$ *by def. of successor state, and* $l \notin \sigma_2$ *by hypothesis. Also,* $l \notin \sigma_0$ *implies* $l \notin \sigma_1 \cap \sigma_0$*.*

Hence, there exists some state constraint, w*, such that:*
$l = head(w)$, $body(w) \subseteq \sigma_1$*, and* $body(w) \not\subseteq \sigma_2$*.*

By definition of $dep(AD)$*, for every* $l' \in body(w)$*, there exists* $\langle l, l' \rangle$ *in* $dep(AD)$*. Since* $body(w) \subseteq \sigma_1$ *and* $body(w) \not\subseteq \sigma_2$*,* $\langle l, l' \rangle$ *for some* $l' \in body(w)$*.*

◇

S-Contained Paths

Definition 8 (S-contained path) *A dependency path $\langle l_1, l_2, \ldots, l_k \rangle$ such that, for every l_i, $l_i \in S$.*

Definition 9 (S-support of l, C_l^S) *The set of all fluent literals that occur in at least one S-contained path starting from l.*

Lemmas (cont'd)

Lemma 2 *For every* $\langle \sigma_0, a, \sigma_1 \rangle$, $\langle \sigma_0, a, \sigma_2 \rangle \in trans(AD)$ ($\sigma_1 \neq \sigma_2$) *and every* $l \in \sigma_1 \setminus \sigma_2$ *such that* $l \notin \sigma_0$, *there exists a* ($\sigma_1 \setminus \sigma_2$)-*contained path in* $dep(AD)$ *that starts from* l.

Proof. *Lemma 1 guarantees the existence of an arc* $\langle l, l' \rangle \in dep(AD)$ *such that* $l' \in \sigma_1 \setminus \sigma_2$.

By def. of ($\sigma_1 \setminus \sigma_2$)-*contained path,* $\langle l, l' \rangle$ *is a* ($\sigma_1 \setminus \sigma_2$)-*contained path.*

$\diamond$

Lemmas (cont'd)

Lemma 3 *For every* $\langle \sigma_0, a, \sigma_1 \rangle$, $\langle \sigma_0, a, \sigma_2 \rangle \in trans(AD)$ $(\sigma_1 \neq \sigma_2)$ *and every* $l \in \sigma_1 \setminus \sigma_2$, *the set* $\sigma_1 \setminus C_l^{\sigma_1 \setminus \sigma_2}$ *is closed under the state constraints of* AD.

Proof. *Let* $\delta = \sigma_1 \setminus C_l^{\sigma_1 \setminus \sigma_2}$. *Proving the claim by contradiction, let us assume that there exists a state constraint,* **caused** g **if** $g_1, \ldots, g_h$, *such that* $\{g_1, \ldots, g_h\} \subseteq \delta$ *but* $g \notin \delta$.

Obviously, $g \in \sigma_1$. *Since* $g \notin \delta$, $g \in C_l^{\sigma_1 \setminus \sigma_2}$. *By def. of* $C_l^{\sigma_1 \setminus \sigma_2}$, *there exists a* $(\sigma_1 \setminus \sigma_2)$*-contained path* $\langle l, \ldots, g \rangle$ *in* $dep(AD)$. *By def. of dependency path, for every* $1 \leq i \leq h$, $\langle l, \ldots, g, g_i \rangle$ *is a dependency path.*

Notice that there exists $g' \in \{g_1, \ldots, g_h\}$ *such that* $g' \notin \sigma_2$. *(Otherwise, it would follow that* $g \in \sigma_2$, *which contradicts* $g \in C_l^{\sigma_1 \setminus \sigma_2}$.*) Hence,* $g' \in \sigma_1 \setminus \sigma_2$. *By def. of* $(\sigma_1 \setminus \sigma_2)$*-contained path,* $\langle l, \ldots, g, g' \rangle$ *is* $(\sigma_1 \setminus \sigma_2)$*-contained. By def. of* $C_l^{\sigma_1 \setminus \sigma_2}$, $g' \in C_l^{\sigma_1 \setminus \sigma_2}$. *Hence,* $g' \notin \delta$, *which contradicts the assumption that* $\{g_1, \ldots, g_h\} \subseteq \delta$.

$\diamond$

Lemmas (cont'd)

Lemma 4 *For every* $\langle \sigma_0, a, \sigma_1 \rangle$, $\langle \sigma_0, a, \sigma_2 \rangle \in trans(AD)$ $(\sigma_1 \neq \sigma_2)$ *and every* $l \in \sigma_1 \setminus \sigma_2$ *such that* $l \notin \sigma_0$, *there exists a* $(\sigma_1 \setminus \sigma_2)$*-contained path,* $\langle l, l_1, \ldots, l_k \rangle$, *such that* $l_k \in \sigma_0$.

Proof. *Proving by contradiction, assume that, for every* $(\sigma_1 \setminus \sigma_2)$*-contained path* $\langle l, l_1, \ldots, l_k \rangle$, $l_i \notin \sigma_0$ *for every* l_i.

Let $\delta = \sigma_1 \setminus C_l^{\sigma_1 \setminus \sigma_2}$. *Since existence of a* $(\sigma_1 \setminus \sigma_2)$*-contained path starting from* l *is guaranteed by Lemma 2,* $C_l^{\sigma_1 \setminus \sigma_2}$ *is not empty. Hence,* $\sigma_1 \supset \delta$.

From $E(a, \sigma_0) \subseteq \sigma_1 \cap \sigma_2$ *and* $C_l^{\sigma_1 \setminus \sigma_2} \subseteq \sigma_1 \setminus \sigma_2$, *it follows that* δ *contains* $E(a, \sigma_0)$. *The assumption that* $l_i \notin \sigma_0$ *for every* l_i, *implies that* $C_l^{\sigma_1 \setminus \sigma_2} \cap \sigma_0 = \emptyset$. *Therefore,* δ *also contains* $\sigma_1 \cap \sigma_0$.

Summing up, $\delta \supseteq E(a, \sigma_0) \cup (\sigma_1 \cap \sigma_0)$, *and, by Lemma 3,* δ *is closed under the state constraints of* AD. *Therefore,* $\delta \supseteq Cn_Z(E(a, \sigma_0) \cup (\sigma_1 \cap \sigma_0))$. *Since* $\sigma_1 \supset \delta$, $\sigma_1 \neq Cn_Z(E(a, \sigma_0) \cup (\sigma_1 \cap \sigma_0))$. *Contradiction.*

◇

Lemmas (cont'd)

Lemma 5 *For every* $\langle\sigma_0, a, \sigma_1\rangle$, $\langle\sigma_0, a, \sigma_2\rangle \in trans(AD)$ *(*$\sigma_1 \neq \sigma_2$*) and for every* $l \in \sigma_1 \setminus \sigma_2$ *such that* $l \notin \sigma_0$, *there exists a conditional path* π *in* $dep(AD)$ *such that*

$$\pi^s = l \ \wedge \ \pi^e \in \sigma_1 \setminus \sigma_2 \ \wedge \ \pi^e \in \sigma_0. \quad (4)$$

Proof. *Existence of* π *satisfying (4): follows directly from Lemma 4.*

π *conditional: by contradiction. Assume* π *contains only 1-arcs. (*l_i *denotes i*th *node of* π*.) Then, for every* σ, $l_{i+1} \in \sigma \rightarrow l_i \in \sigma$. *Because* $\pi^e \in \sigma_0$, $l_{|\pi|-1} \in \sigma_0$. *By induction,* $l_1 \in \sigma_0$. *Since* $l_1 = l$, $l \in \sigma_0$. *But* $l \notin \sigma_0$ *by hypothesis. Contradiction.*

$\diamond$

Lemmas (cont'd)

Lemma 6 *For every $\langle \sigma_0, a, \sigma_1 \rangle$, $\langle \sigma_0, a, \sigma_2 \rangle \in trans(AD)$ ($\sigma_1 \neq \sigma_2$), every $l \in \sigma_1 \setminus \sigma_2$ such that $l \notin \sigma_0$, and every $k > 0$, there exists a conditional neg-seq, $\langle \pi_1, \ldots, \pi_k \rangle$, such that $\pi_1^s = l$.*

Proof. *By induction on k.*

Base: $k = 1$. The conclusion follows directly from Lemma 5.

Inductive Step: assume theorem holds for k, and prove it for $k + 1$.

By Lemma 5, there exists a conditional path, π_1, such that $\pi_1^s = l$, $\pi_1^e \in \sigma_1 \setminus \sigma_2$, and $\pi_1^e \in \sigma_0$.

Because $\pi_1^e \in \sigma_1 \setminus \sigma_2$, $\overline{\pi_1^e} \in \sigma_2 \setminus \sigma_1$; also, from $\pi_1^e \in \sigma_0$, it follows that $\overline{\pi_1^e} \notin \sigma_0$.

By inductive hypothesis, there exists a conditional neg-seq, $\langle \pi_2, \ldots, \pi_{k+1} \rangle$, of length k, such that $\pi_2^s = \overline{\pi_1^e}$.

By definition, $\langle \pi_1, \pi_2, \ldots, \pi_{k+1} \rangle$ is a conditional neg-seq. Since its length is $k + 1$, and $\pi_1^s = l$, the proof is complete.

$\diamond$

Lemmas (cont'd)

Lemma 7 *For every* σ_0 *and* a *such that* a *is executable in* σ_0*, if* $E(a, \sigma_0) \subseteq \sigma_0$*, then* σ_0 *is the only successor state of* σ_0 *under* a*.*

Proof. *Consider an arbitrary* $\langle \sigma_0, a, \sigma_1 \rangle \in trans(AD)$*, and let us prove that, under the hypotheses,* $\sigma_1 = \sigma_0$*.*

Recall that $\sigma_1 = Cn_Z(E(a, \sigma_0) \cup (\sigma_1 \cap \sigma_0))$*. Obviously,* $\sigma_1 \cap \sigma_0 \subseteq \sigma_0$*. As* $E(a, \sigma_0) \subseteq \sigma_0$ *by hypothesis,* $E(a, \sigma_0) \cup (\sigma_1 \cap \sigma_0) \subseteq \sigma_0$*.*

Since σ_0 *is a state, for every* $X \subseteq \sigma_0$*,* $Cn_Z(X) \subseteq \sigma_0$*. Hence,* $Cn_Z(E(a, \sigma_0) \cup (\sigma_1 \cap \sigma_0)) \subseteq \sigma_0$*, which implies that* $\sigma_1 \subseteq \sigma_0$*. Since* σ_0*,* σ_1 *are states,* $\sigma_1 = \sigma_0$*.*

$\diamond$

Corollaries

Corollary 1 *For every* σ_0 *and* a *such that* a *is executable in* σ_0*, if* $\langle \sigma_0, a, \sigma_0 \rangle \in trans(AD)$*, then* σ_0 *is the only successor state of* σ_0 *under* a*.*

Proof. *By def. of successor state,* $E(a, \sigma_0) \subseteq \sigma_0$*. The application of Lemma 7 concludes the proof.*

◇

Corollary 2 *For every* $\langle \sigma_0, a, \sigma_1 \rangle$*,* $\langle \sigma_0, a, \sigma_2 \rangle \in trans(AD)$ *such that* $\sigma_1 \neq \sigma_2$*,*

$$\sigma_1 \neq \sigma_0 \text{ and } \sigma_2 \neq \sigma_0.$$

Proof. *By contradiction. If* $\sigma_1 = \sigma_0$*, then* $\sigma_2 = \sigma_0$ *by Corollary 1. Therefore,* $\sigma_1 = \sigma_2$*. Contradiction.*

◇

Proof of the Main Theorem

We prove the contrapositive of the theorem, i.e.:

If AD is non-deterministic, then $dep(AD)$ is not safe.

Proof. Since AD is non-deterministic, there exist $\langle \sigma_0, a, \sigma_1 \rangle, \langle \sigma_0, a, \sigma_2 \rangle \in trans(AD)$ such that $\sigma_1 \neq \sigma_2$. By Corollary 2, there exists $l \in \sigma_1 \setminus \sigma_2$ such that $l \notin \sigma_0$.

Let:

- n: number of all fluent literals from signature of AD.
- k': some positive integer such that $k' > n$.

Lemma 6 guarantees existence of a neg-seq, $\langle \pi_1, \ldots, \pi_{k'} \rangle$, such that $\pi_1^s = l$.

Since $k' > n$, there exist $1 \leq i < j \leq k'$ such that $\pi_i^s = \pi_j^s$. By def. of neg-seq, $\pi_j^s = \overline{\pi_{j-1}^e}$. By def. of neg-loop, $\langle \pi_i, \pi_{i+1}, \ldots, \pi_{j-1} \rangle$ is a conditional neg-loop.

Hence $dep(AD)$ contains a conditional neg-loop. By definition of safe dependency graph, $dep(AD)$ is not safe.

◇

Examples

Consider the non-deterministic action description:

$$\begin{cases} \text{caused } p \text{ if } \neg q, r \\ \text{caused } q \text{ if } \neg p, r \\ a \text{ causes } r \end{cases}$$

Its dependency graph is *not safe*, as it contains the conditional neg-loop:

$$\langle \langle p, \neg q \rangle, \langle q, \neg p \rangle \rangle.$$

Examples (cont'd)

The action description:

$$\begin{cases} \text{caused } p \text{ if } \neg p, q \\ a \text{ causes } q \end{cases}$$

is deterministic, and its dependency graph is safe (no arcs out of nodes $\neg p$ and q).

Examples (cont'd)

The action description:

$$\begin{cases} \text{caused } p \text{ if } q, r \\ \text{caused } p \text{ if } \neg q, r \\ a \text{ causes } r \end{cases}$$

is deterministic, and its dependency graph is safe (no arcs out of nodes q and $\neg q$).

Counter-Examples

The action description:

$$\left\{ \begin{array}{l} \text{caused } p \text{ if } \neg q, \neg r \\ \text{caused } q \text{ if } \neg p, r \\ a \text{ causes } r \end{array} \right.$$

is deterministic. However, its dependency graph is *not safe*, as it contains the conditional neg-loop:

$$\langle \langle p, \neg q \rangle, \langle q, \neg p \rangle \rangle.$$

Possible solution: parametrize $dep(AD)$ on a set of fluent literals, and re-define "safety" considering only consistent sets of fluent literals.

Counter-Examples (cont'd)

The action description:

$$\begin{cases} \text{caused } p \text{ if } \neg q, r \\ \text{caused } q \text{ if } \neg p, r \\ a \text{ causes } \neg r \end{cases}$$

is deterministic: executing a only makes r false. However, the dependency graph is *not safe*, as it contains the conditional neg-loop:

$$\langle \langle p, \neg q \rangle, \langle q, \neg p \rangle \rangle.$$

Possible solution: difficult, requires considering laws other than state constraints.