FURTHERING THE CONTINUOUS-CHANGE EVENT CALCULUS: PROVIDING FOR EFFICIENT DESCRIPTION OF ADDITIVE EFFECTS AND AN AUTOMATED REASONER

By

Ankesh Khandelwal

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

_____
Peter Fox, Thesis Adviser

_____
James Hendler, Thesis Adviser

_____
Selmer Bringsjord, Member

_____
Leora Morgenstern, Member

Rensselaer Polytechnic Institute
Troy, New York

April 2013
(For Graduation May 2013)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

The contributions made here would not have been possible without constant support and learnings from others.

As research advisors, Peter Fox and Jim Hendler gave a lot of freedom with regards to research I could pursue, and helped me along. And, they assured my funding without me knowing how they managed it, which became important when I solely focused on my thesis work beyond third year. I also got unqualified consent for staying back at home with family, as they fully appreciated the situation.

Parents have played a very influential role in my life. They have influenced me more with their acts than words. But in past two years they did something even more special. Despite the situation at home, they told me not to worry and enjoy my life here. And, that is what I did! Last few years showed me the steadfast resolve and strength of my mother and the constant support from my father no matter what I did!

Leora Morgenstern and Selmer Bringsjord gave crucial feedback on improving my work.

Rob Miller was kind enough to spend time to go over some of my work. He helped me understand some of the mistakes I had made, and gave useful suggestions for improvements in my work.

I have usually got helpful feedback from reviewers at different publication venues, and I am particularly grateful to one anonymous reviewer of the Journal of Artificial Intelligence Research (JAIR) for a very constructive feedback. It offered some new insights into own work!

Articles and books by John McCarthy, Ray Reiter, Vladimir Lifschitz, Murray Shanahan et al. made research in the field of Knowledge Representation and nonmonotonic reasoning so much fun.

Implementation of the model builder for Event Calculus was only possible because of some preexisting fantastic works, namely `F2LP` tool, `DLVHEX` solver, and `Mathematica`. We are very grateful to Ravi Palla and Joohyung Lee for the `F2LP`

tool, Thomas Krennwallner, Thomas Eiter et al. for `DLVHEX` solver, and Wolfram Research for the very sophisticated `Mathematica` software and support for external program communication through their `MathLink` library.

Other than the advisors I shared some personal matters with Naveen Sundar Govindarajulu, and I got some great advise and support. More importantly, I learned a lot about logic, computer science, current affairs, and critical thinking (and about self!) from him.

I found great roommates in Gaurav Jain, Sai Praveen, and Vaibhav Raj Govil. We readily shared the chores at home. But crucially, I learned from each of them, under different circumstances, that personal problems should not affect the atmosphere at home or our interactions with others. They never let tragedies or difficulties in personal lives affect the fun we had together.

It has been very rewarding to share office space with Jesse Weaver. Not only did he teach me American ways, particularly in work environments, he was inspirational with his achievements from relatively humble background, work ethics, strong value system and moral standards.

ABSTRACT

Semantic Web refers to a Web of interconnected data enriched with semantics. It subscribes to logic-based representations of knowledge through W3C standards such as the Resource Description Framework and the Web Ontology Language for encoding clear semantics. To date, knowledge representation however has been confined to descriptions of artifacts or data. We began the research reported here in pursuit of the inclusion of knowledge about physical processes and natural laws, into the Semantic Web. Such knowledge could then be combined with experimental data, for example, in a largely automated fashion, for new inferences. In this pursuit we explored the extensive research in the field of reasoning about actions and changes, and deduced that the (circumscriptive) Event Calculus is the most expressive logic-based formalism available for logic-based description of continuous-changes.

In this thesis we extend the Event Calculus formalism with new predicates for descriptions of discrete and continuous additive effects whose semantics are given via aggregate formulas in first-order logic. To the best of our knowledge this is the first application of aggregate formulas in first-order logic, even though aggregates have been in use in other logics such as answer-set programming. The *frame problem* is one of representing the effects of actions without explicitly representing all their non-effects. Nonmonotonic reasoning via circumscription is used in the Event Calculus as a solution to the frame problem. We show, however, that circumscription which was defined for first-order logic without aggregates is inadequate for modeling the frame problem in the extended Event Calculus if used as it is for formulas with aggregates, as it selects anomalous models.

We extend the circumscription transformation to first-order logic with aggregates, named the $CIRC^A$ transformation. $CIRC^A$ transformation is a generic transformation that addresses a general problem of identifying and not selecting unintended models, classified formally as *weak* models, that circumscription normally selects in the presence of aggregates. We deploy $CIRC^A$ transformation for resolving the frame problem in the extended Event Calculus.

Finally, we devise a method for constructing models for given, numerical, and finite Event Calculus domain descriptions, given an initial state and narratives of external actions. An Event Calculus system evolves through alternating phases of continuous changes and instantaneous discontinuous changes. The devised method involves separation of logic and *equations* reasoning through syntactic derivations of new axioms from the given domain descriptions, such that discontinuous changes, equations for trajectories of continuous changes, and mathematical conditions for next discontinuous changes are determined from logic reasoning while trajectories of continuous changes and the time for next discontinuous change are determined from equations reasoning. With this separation, the off-the-shelf logic reasoners and equation solvers can be combined to implement an automated model builder for the Event Calculus. We have implemented a prototypical reasoner using the `DLVHEX` logic reasoner and `Mathematica` libraries.

The results of this thesis may encourage the use of logic formalisms/systems for descriptions of dynamical systems with quantitative descriptions of continuous-changes. Additive effects are very common in concurrent systems, and the extended Event Calculus allows for general, concise and elaboration tolerant descriptions of them, which among other things makes the descriptions amicable to sharing, reuse, and modular development. The prototypical model-builder for the continuous-change Event Calculus formalism broadens its scope beyond theory, positioning it for use in practice. Finally, we hope that these are some crucial steps towards realizing a process modeling language for the Semantic Web alluded to in the beginning.

CHAPTER 1
INTRODUCTION

Semantic Web refers to a Web of interconnected data enriched with semantics. It subscribes to logic-based representations of knowledge through W3C standards such as the Resource Description Framework and the Web Ontology Language for encoding clear semantics. To date, knowledge representation however has been confined to descriptions of artifacts or data. We began the research reported here in pursuit of the inclusion of knowledge about physical processes and natural laws, into the Semantic Web. Such knowledge could then be combined with experimental data, for example, in a largely automated fashion, for new inferences. In this pursuit we explored and have sought to extend the research in the field of reasoning about actions and changes.

Extensive research has happened in the field of modeling dynamical systems over the past five decades (and more). There are plenty of formalisms to choose from, including Petri nets (Murata, 1989), process algebras (Baeten, 2005), dynamic and temporal logics (Galton, 2008), hybrid automata (Henzinger, 1996), Markov decision processes (Bellman, 1957), differential equations, STRIPS operators (Fikes & Nilsson, 1971), etc. In the afore-stated pursuit we explored the formalisms in the field of declarative and mathematical logic based modeling, as opposed to procedural specifications and state-based approaches[1], and particularly reasoning about actions and changes. Research over the past five decades (and more) has led to many different logic-based formalisms for representations of discrete changes, such as Action Languages $\mathcal{E}$ (Gelfond & Lifschitz, 1998) and $\mathcal{C}+$ (Giunchiglia, Lee, Lifschitz, McCain, & Turner, 2004), Temporal Action Logics (Doherty & Kvarnstrm, 2008), Situation Calculus (Reiter, 1993; Lin & Reiter, 1994), Fluent Calculus (Thielscher,

[1]Quote by Reiter (2001)(Reiter, 2001) regarding declarative and logic-based modeling: "with a non-procedural specification of a domain, it is at-least clear what modeling assumptions are being made and one can hope to prove, entirely within the logic, various properties of the specification. The correctness properties of a system can be established via logical deduction. Instead of explicitly enumerating states and their transition function, unlike state-based approaches, sentences in mathematical logic are used to give the descriptions of what is true of the system and its environment and of the causal laws in effect for the domain".

2001), and Event Calculus (Kowalski & Sergot, 1986; Shanahan, 1999a).

Some of the above formalisms, Situation Calculus (Reiter, 1996) and Event Calculus (Miller & Shanahan, 1996) for example, were extended to reason about continuous changes described as functions of time. Differential calculus is one of the primary tools for mathematical modeling of processes in sciences and engineering, and in some of the above formalisms, Event Calculus for example, continuous changes can often also be described via (or decomposed into) systems of ordinary differential equations (ODEs). However, two aspects where the latter expressive formalisms were lacking were readily apparent: (A) Descriptions of additive effects were very inefficient because the only possible way of expressing the combined effects of concurrently active effects was to enumerate each different case. Additive effects are very common in concurrent systems, and some simple examples of additive effects include multiple forces applied on a block, water flowing out from multiple outlets of a tank, and multiple concurrent scoops from a water tank. Generic expressions such as the aggregate expressions which are now a common feature in query languages or even rule languages such as answer-set programming seemed directly relevant for descriptions of their combined effects. (B) There was no reasoner for any form of reasoning tasks. Automated theorem proving in first-order logic (Robinson & Voronkov, 2001) in general is hard, and real-valued time and temporal properties and differential constraints only make that worse. Besides, many past attempts at building reasoners for formalisms with continuous-change remain within pure logical reasoning (Chintabathina, Gelfond, & Watson, 2005; Lee & Palla, 2012b), unlike constraint logic programming (Gebser et al., 2011) for example, where external constraint solvers are used for solving constraint satisfaction problems. Many forms of reasoning for different and expressive logic formalisms, finding models for answer-set programs (Gelfond, 2008) for example, have advanced greatly over the past decade, and we could benefit from these gains in efficiencies together with solving differential and other constraints separately.

Miller and Shanahan's (1996) circumscriptive Event Calculus based formalism is one of the very advanced formalisms that support ODE-based descriptions of continuous-changes. Continuous-changes can be described using arbitrary *au-*

tonomous ordinary differential equations (Teschl, 2012), and the formalism has robust solutions for the frame and related problems and can handle domain constraints, triggered actions, concurrent actions, and actions with non-deterministic effects. ODE are in general of the form, $d(x)/dt = f(x,t)$, whereas autonomous ODE are of the form, $d(x)/dt = f(x)$ (Teschl, 2012). That is, f is independent of t in autonomous ODE. In the remainder, except in Chapter 6, we refer to the extended formalism as the (circumscriptive) Event Calculus. The Event Calculus and other background material is reviewed in Chapter 2.

For distinct and more efficient descriptions of discrete and continuous additive effects, we extend the circumscriptive Event Calculus formalism with new predicates whose semantics is given via aggregate formulas in first-order logic. The extensions are described in Chapter 4. To the best of our knowledge ours' is the first application of aggregate formulas in first-order logic, even though aggregates have been in use in other logics such as answer-set programming.

The *frame problem* is one of representing the effects of actions without explicitly representing all their non-effects. Nonmonotonic reasoning via circumscription is used in the Event Calculus as a solution to the frame problem. We show, however, that circumscription which was defined for first-order logic without aggregates is inadequate for modeling frame problem in the extended Event Calculus if used as it is for formulas with aggregates, as it selects models with anomalous additive effects which violate the physical laws of causality.

So, we extend the circumscription transformation to first-order logic with aggregates with a unique transformation for aggregates, referred to as the $CIRC^A$ transformation. $CIRC^A$ transformation is a generic circumscription-like transformation, and addresses a general problem of identifying and not selecting unintended models that circumscription normally selects in the presence of aggregate formulas. We deploy $CIRC^A$ transformation for resolving the frame problem in the extended Event Calculus. The $CIRC^A$ transformation is described in Chapter 3.

In the extended Event Calculus, discrete and continuous additive effects from different sources can be axiomatized separately, and summing of concurrent effects for their combined effects is handled implicitly by domain-independent semantics of

the extended Event Calculus. General principles of axiomatizing domains in first-order logic, (for example, Davis, 1999), can be applied to represent any discrete or ODE-based continuous additive effects systematically. Due to the implicit summation of concurrent effects, additive effects from new sources can often be included by simply appending axioms describing the new additive effects. Furthermore, indirect discrete additive effects as well as simultaneous continuous additive effects on quantities which have some interdependencies can also be conveniently represented in the extended Event Calculus. Unlike the previous proposals for distinct descriptions of additive effects, such as by Lee and Lifschitz (2003), which involve syntactic constructs, our method involves logical extensions.

The extended Event Calculus allows for general, concise and additive-elaboration tolerant descriptions of additive effects. We illustrate advantages from the extensions through description of a 2-D multiple blocks example for determining the effects of application of one or more vertical and/or horizontal forces on the velocities and accelerations of multiple blocks stacked over one another. The axiomatic description given in Section 4.5 is valid for a very general configurations of blocks.

Finally, an automated reasoner for the Event Calculus requires the capability to reason about differential constraints (specifically, solving ordinary differential equations), in addition to logic reasoning. While logic reasoners are not well-suited for solving differential equations, they can be combined with separate (differential) equations solvers. Furthermore, if a strict separation of logic reasoning and equations solving were possible, an automated reasoner for the Event Calculus could be built using off-the-shelf logic reasoners and equations solvers. Hence, we devise a method for separating logic reasoning and equations solving for reasoning about Event Calculus descriptions.

An Event Calculus system can be thought of as evolving through phases of continuous changes and instantaneous discontinuous changes caused by actions. We reformulate the axiomatization of the original Event Calculus semantics in terms of changes at the points of discontinuous changes, introduce and define new terminologies relevant in deductions, and derive syntactically new sets of axioms from given domain descriptions. With those changes and additions, logic reasoning can

be performed for inferring discontinuous changes and equations (and initial values of quantities) relevant for computing trajectories of continuous-change and determination of next point of discontinuity. And, equations can be solved for determining the next point of discontinuity and the values of quantities at that point. The state of temporal properties of the system between two consecutive points of discontinuities can often be determined from the states of temporal properties and equations active at the later of the two points.

Per the original semantics of the Event Calculus, quantities – temporal properties with real values – are continuous by default, and any discrete-change is caused by an action. That is, unless a given quantity is known to be subject to a discrete-change by an action, it is assumed to be continuous. This default assumption is crucial for solution to the frame problem because, for instance, in absence of a known action occurrence, a given quantity can be deterministically inferred to be continuous. But a continuous quantity can change along multiple trajectories, and so, in absence of an equation explicitly describing the change in its value, the quantity can change non-deterministically. And thereby, in the specific scenarios where the value of the quantity is constant, an equation stating so has to explicitly specified. This makes domain descriptions tedious, and solution to the frame problem with regards to change in the values of quantities very inefficient. An additional default assumption that by default, quantities are constant if continuous offers a more efficient solution to the frame problem in the above context. This default assumption cannot be expressed using the terminologies of the Event Calculus, but with help of the new terminologies mentioned earlier, we can axiomatize and introduce the afore-stated default assumption as well.

An Event Calculus domain description is *given*, *finite*, and *numerical* if all the temporal properties of the domain are explicitly given and finite in number, and all the real numbers are numerical (that is, they are not denoted by non-numerical constant symbols). We implement a prototypical model-builder for given, finite, and numerical domain descriptions, using the approach outlined above, with help of the `DLVHEX` reasoner (Eiter, Ianni, Schindlauer, & Tompits, 2006) and `Mathematica` libraries (Miyaji & Abbott, 2001). Given an initial state and narratives of exter-

nal actions, the model-builder constructs models that give the states of temporal properties in the future. It naturally supports simulation (or temporal projection or prediction) type of reasoning.

The separation of logic reasoning and equations solving for reasoning about Event Calculus domain descriptions, axiomatization of the new default assumption regarding quantities, and the prototypical implementations are described in Chapter 5. In theory we can construct models for any finite, given, and numerical domains using the prototypical implementation but in practice we are constrained by the available logic reasoners. We discuss some of these limitations in Appendix where we also give a few example domain descriptions in the Event Calculus.

CHAPTER 2
BACKGROUND

We review first-order logic, aggregate formulas, circumscription, and other concepts relevant for the definition of CIRC^A transformation and related results, in Section 2.1. We review the Event Calculus in Section 2.2. Finally, we review the first-order stable model semantics, HEX-programs, and other concepts relevant to reformulating Event Calculus as answer-set programs and reasoning with it in Section 2.3.

2.1 First-order Logic, Aggregates, and Circumscription Related

In the following sections, we review nonsorted and many-sorted first-order logic (without aggregate formulas), aggregate formulas, circumscription, Negation Normal Form (NNF), Clark Normal Form (ClNF), predicate completion, and UNA notation. We will also define the depth for aggregate expressions, NNF for formulas with aggregates, and implicative formulas. Note that NNF and depth of aggregate expressions are required for the definition of CIRC^A transformation, implicative formulas and predicate completion are required for some of the results related to CIRC^A transformation of formulas, and UNA notation is used for defining unique name constraints.

2.1.1 First-order Logic

We describe the syntax and semantics of nonsorted and many-sorted first-order logic. Our overview follows the definition given in Lifschitz, Morgenstern, and Plaisted (2008).

Portions of this chapter to appear in: Khandelwal, A., & Fox, P. (2013). General Descriptions of Additive Effects via Aggregates in the Circumscriptive Event Calculus. *Journal of Artificial Intelligence Research* (JAIR), and Khandelwal, A., & Fox, P. (2013). Constructing Models for Continuous-Time and Continuous-Change Event Calculus. Artificial Intelligence Journal (AIJ).

Syntax

In (nonsorted) first-order logic, a signature is a set of symbols of two kinds – function constants and predicate constants – with a nonnegative integer, called the arity, assigned to each symbol.

Variable symbols start in lower case, for instance, x, y, z, x_1, y_1, z_1, ... Terms of a signature σ are formed from variables and from function constants of σ. An atomic formula of σ is an expression of the form $P(t_1, \ldots, t_n)$ or $t_1 = t_2$, where P is a predicate constant of arity n, and each t_i is a term of σ. Formulas are formed from atomic formulas using unary connectives $\neg$, the binary connectives $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$, and the quantifiers $\forall$, $\exists$.

An occurrence of a variable v in a formula F is bound if it belongs to a subformula of F that has the form $\forall vG$ or $\exists vG$; otherwise it is free. If at least one occurrence of v in F is free then v is a free variable of F. A closed formula, or a sentence, is a formula without free variables. The universal closure of a formula F is the sentence $\forall v_1 \ldots v_n$F, where v_1,..., v_n are the free variables of F.

Semantics

An interpretation (or structure) of a signature σ consists of

- a nonempty set $|I|$, called the universe (or domain) of I,
- for every object constant c of σ, an element c^I of $|I|$,
- for every function constant f of σ of arity $n > 0$, a function f^I from $|I|^n$ to $|I|$,
- for every propositional constant P of σ, an element P^I of $\{FALSE, TRUE\}$,
- for every predicate constant R of σ of arity $n > 0$, a function R^I from $|I|^n$ to $\{FALSE, TRUE\}$.

The interpretation of a predicate constant R of σ of arity $n > 0$, can equivalently be defined as $R^I \subseteq |I|^n$. Then, R^I is referred to as the *extent* or the *extension* of R.

For any element ξ of its universe $|I|$, select a new symbol ξ^*, called the name of ξ. σ^I denotes the signature obtained from σ by adding all names ξ^* as object constants. The interpretation I is extended to the new signature I by defining $(\xi^*)^I = \xi$ for all $\xi \in |I|$.

The semantics of first-order logic defines, for any sentence F and any interpretation I of a signature σ^I, the truth value F^I that is assigned to F by I.

For any term t of σ^I that does not contain variables, the element t^I of the universe that is assigned to t by I is defined recursively as follows. If t is an object constant then t^I is part of the interpretation I . For other terms, t^I is defined by the equation, $f(t_1, \ldots, t_n)^I = f^I(t_1^I, \ldots, t_n^I)$, for all function constants f of arity $n > 0$. For any propositional constant P, P^I is part of the interpretation I. F^I for a sentence F of the extended signature σ^I is defined as follows:

- $R(t_1, \ldots, t_n)^I = R^I(t_1^I, \ldots, t_n^I)$,
- $(\neg F)^I = \neg(F^I)$,
- $(F \odot G)^I = \odot(F^I, G^I)$ for every binary connective $\odot$,
- $\forall w F(w)^I = TRUE$ if $F(\xi^*)^I = TRUE$ for all $\xi^* \in |I|$,
- $\exists w F(w)^I = TRUE$ if $F(\xi^*)^I = TRUE$ for some $\xi^* \in |I|$.

An interpretation I satisfies a sentence F, or is a model of F, written as $I \models F$, if $F^I = TRUE$. A sentence F is logically valid if every interpretation satisfies F. Two sentences, or sets of sentences, are equivalent to each other if they are satisfied by the same interpretations.

A set Γ of sentences is satisfiable if there exists an interpretation satisfying all sentences in Γ. A set Γ of sentences entails a formula F (symbolically, $\Gamma \models F$) if every interpretation satisfying Γ satisfies the universal closure of F.

Sorts

Besides function constants and predicate constants, a many-sorted signature includes symbols called sorts. In addition to an arity n, every function constant and

every predicate constant is assigned its argument sorts $s_1, \ldots, s_n$; every function constant is assigned its value sort s_{n+1}.

For every sort s, we assume a separate infinite sequence of variables of that sort. The recursive definition of a term assigns a sort to every term. Atomic formulas are expressions of the form $P(t_1, \ldots, t_n)$, where the sorts of the terms $t_1, \ldots, t_n$ are the argument sorts of P, and also expressions $t_1 = t_2$ where t_1 and t_2 are terms of the same sort.

An interpretation, in the many-sorted setting, includes a separate nonempty universe $|I|^s$ for each sort s. Otherwise, extending the definition of the semantics to many-sorted languages is straightforward.

A further extension of the syntax and semantics of first-order formulas allows one sort to be a subsort of another. Generally, a subsort relation is an order (reflexive, transitive, and anti-symmetric relation) on the set of sorts.

2.1.2 First-order Aggregate Formulas

The review follows the definitions given by Lee and Meng (2009) (and Ferraris and Lifschitz, 2010). An *aggregate function* is a partial function from the class of multisets to $\mathcal{R} \cup \{+\infty, -\infty\}$, where $\mathcal{R}$ is the set of real numbers. A *multiset* is defined as a set along with a function assigning a positive integer (and $+\infty$), called the *multiplicity*, to each of its elements. The aggregate function $\#sum$, for example, maps multiset α to the sum of all real numbers from α if there are finitely many of them (otherwise, the sum may be $+\infty$, $-\infty$, or undefined).

If OP is an aggregate function, for example $\#sum$, $\#count$, $\#max$, $\#min$, $\#times$, and F is a formula, then $OP\langle \mathbf{x}.F(\mathbf{x})\rangle$ is an *aggregate expression*, where the construct $\mathbf{x}.F(\mathbf{x})$ is a quantifier binding the members of $\mathbf{x}$ and $\mathbf{x}$ is a tuple of free variables in $F(\mathbf{x})$. Further, if t is any term and $\succeq$ is a comparison operator, then $OP\langle \mathbf{x}.F(\mathbf{x})\rangle \succeq t$ is an *aggregate formula*. The *complement* of $OP\langle \mathbf{x}.F(\mathbf{x})\rangle \succeq t$, in the sense of $\neg(OP\langle \mathbf{x}.F(\mathbf{x})\rangle \succeq t)$, is denoted by $OP\langle \mathbf{x}.F(\mathbf{x})\rangle \prec t$. The aggregate formulas are considered well-formed formulas (WFFs) of first-order logic, and the usual recursive definition of WFFs of first-order logic is applicable for construction of other formulas.

The *extent* or *extension* of a predicate refers to the tuples that belong to a predicate. Let P and Q be unary predicate constants. If the extents of P and Q respectively are $\{A, B, C\}$ and $\{B, C, D\}$ then $x.(P(x) \wedge Q(x))$ binds x to $\{B, C\}$.

Aggregate formulas may be nested. We say that an aggregate formula (and the corresponding aggregate expression) in a formula is at *depth zero* relative to F if it is not inside any other aggregate expression in F. Therefore, if F is $OP\langle \mathbf{x}.F' \rangle \succeq t$ then any aggregate expression in F' is at a depth greater than zero relative to F. (The exact depth is not important.)

For any set X of n-tuples ($n \geqslant 1$), $msp(X)$, the *multiset projection* of X, is the multiset consisting of all ξ_1 such that $(\xi_1, \xi_2, \ldots, \xi_n) \in X$ for at least one $(n-1)$-tuple $(\xi_2, \ldots, \xi_n)$, with the multiplicity equal to the number of such $(n-1)$-tuples, possibly $+\infty$. Assuming A, B, C are unique object constants, $msp(\{\langle -1, A\rangle, \langle 1, B\rangle, \langle 1, C\rangle\})$ equals the multiset $\{-1, 1, 1\}$ and $\#sum\langle msp(\{\langle -1, A\rangle, \langle 1, B\rangle, \langle 1, C\rangle\})\rangle = 1$, where $\{\langle -1, A\rangle, \langle 1, B\rangle, \langle 1, C\rangle\}$ is a set of 2-tuples.

An interpretation I of first-order aggregate formulas, for signature σ^I, is defined as follows:

- $(x_1 \ldots x_n F(x_1, \ldots, x_n))^I = msp(\{(\xi_1, \ldots, \xi_n) \in |I|^n : F(\xi_1^*, \ldots, \xi_n^*)^I = TRUE\})$,
- $(OP\langle \mathbf{x}.F(\mathbf{x})\rangle)^I = OP((\mathbf{x}.F(\mathbf{x}))^I)$,
- $(OP\langle \mathbf{x}.F(\mathbf{x})\rangle \succeq t)^I = (OP\langle \mathbf{x}.F(\mathbf{x})\rangle)^I \succeq t^I$.

Signature σ^I denotes the signature obtained from σ by adding all names ξ^* - new symbols for every element ξ of universe of I (denoted as $|I|$) - as object constants. $\mathcal{R} \cup \{+\infty, -\infty\}$ is assumed to be a subset of $|I|$. Each comparison operator ($\succeq$) is interpreted by its standard interpretation in the real numbers.

$(\#sum\langle \mathbf{x}.F(\mathbf{x})\rangle)^I$, for example, equals sum of elements in the multiset $(\mathbf{x}.F(\mathbf{x}))^I$.

2.1.3 Circumscription

Circumscription is a transformation of predicate formulas proposed by McCarthy (1980) for the purpose of formalizing nonmonotonic aspects of a theory. Circumscription is used to minimize the extensions of certain predicates. The review follows the definitions given by Lifschitz (1994).

If P and Q are n-ary predicates, then P $\leqslant$ Q iff the extension of P is a subset of the extension of Q, that is, $\forall \mathbf{x}(P(\mathbf{x}) \longrightarrow Q(\mathbf{x}))$ is true. This can be extended to tuples $\mathbf{P} = [\mathrm{P}_1, \ldots, \mathrm{P}_m]$ and $\mathbf{Q} = [\mathrm{Q}_1, \ldots, \mathrm{Q}_m]$ of predicates. $\mathbf{P} \leqslant \mathbf{Q}$ iff $P_1 \leqslant Q_1 \wedge \ldots \wedge P_m \leqslant Q_m$. $\mathbf{P} = \mathbf{Q}$ iff $\mathbf{P} \leqslant \mathbf{Q} \wedge \mathbf{Q} \leqslant \mathbf{P}$ and $\mathbf{P} < \mathbf{Q}$ iff $\mathbf{P} \leqslant \mathbf{Q} \wedge \neg(\mathbf{Q} \leqslant \mathbf{P})$. Let $\mathbf{P}$ and $\mathbf{Q}$ be partitioned into $\{\mathbf{P_1}, \ldots, \mathbf{P_k}\}$ and $\{\mathbf{Q_1}, \ldots, \mathbf{Q_k}\}$. Then $\mathbf{P} \preceq \mathbf{Q}$ stands for $\bigwedge_{i=1}^{k} (\bigwedge_{j=1}^{i-1} (\mathbf{P_j} = \mathbf{Q_j}) \longrightarrow (\mathbf{P_i} \leqslant \mathbf{Q_i}))$. $\mathbf{P} \prec \mathbf{Q}$ iff $(\mathbf{P} \preceq \mathbf{Q}) \wedge \neg(\mathbf{Q} \preceq \mathbf{P})$.

Circumscription of formulas are second-order formulas defined below. $CIRC[F; \mathbf{P}; \mathbf{Z}]$ denotes *parallel circumscription* of F. $CIRC[F; \mathbf{P_1} > \ldots > \mathbf{P_k}; \mathbf{Z}]$ denotes *prioritized circumscription* of F. $F(\mathbf{p}, \mathbf{z})$ is obtained by replacing all constants from $\mathbf{P}$, $\mathbf{Z}$ in $F(\mathbf{P}, \mathbf{Z})$ with the corresponding variables from $\mathbf{p}$, $\mathbf{z}$.

$$CIRC[F; \mathbf{P}; \mathbf{Z}] = F(\mathbf{P}, \mathbf{Z}) \wedge \neg\exists \mathbf{p}, \mathbf{z}(\mathbf{p} < \mathbf{P} \wedge F(\mathbf{p}, \mathbf{z}))$$
$$CIRC[F; \mathbf{P_1} > \ldots > \mathbf{P_k}; \mathbf{Z}] = F(\mathbf{P}, \mathbf{Z}) \wedge \neg\exists \mathbf{p}, \mathbf{z}(\mathbf{p} \prec \mathbf{P} \wedge F(\mathbf{p}, \mathbf{z}))$$

In the parallel circumscription, the extensions of constants in $\mathbf{P}$ are minimized in parallel, that is without priorities. Whereas, in the prioritized circumscription, minimization of the extensions of constants in $\mathbf{P}_i$ is given priority over that of constants in $\mathbf{P}_j$ whenever $i < j$. The extensions of constants in $\mathbf{Z}$ are allowed to vary when comparing the extensions of constants in $\mathbf{P}$.

$\mathbf{Z}$ and $\mathbf{z}$ may be omitted if empty, and $\mathbf{P}$ may be denoted by the predicates in the tuple as in CIRC[F; P$_1$, ..., P$_n$] and F([P$_1$, ..., P$_n$]), for example. Further, the formula F was denoted as $F(\mathbf{x})$ in the definition of aggregate expressions and as $F(\mathbf{P})$ and $F(\mathbf{p})$ in the circumscription formulas. Such denotations are used subsequently, and whether the argument is a tuple of object variables, predicate constants, or predicate variables becomes clear from the context.

Given interpretation I_1 and I_2 of $F(\mathbf{P}, \mathbf{Z})$, we say that I_1 is *smaller* than I_2 with respect to $\mathbf{P}$ allowing $\mathbf{Z}$ to vary, denoted by $I_1 \subset_{\mathbf{P};\mathbf{Z}} I_2$, if

$$\bigwedge_{C \notin \mathbf{P}, \mathbf{Z}} [I_1(C) = I_2(C)] \wedge \bigwedge_{P \in \mathbf{P}} [I_1(P) \subseteq I_2(P)] \wedge \exists_{P \in \mathbf{P}} [I_1(P) \subset I_2(P)]$$

is valid. (Notation $I_1 \subseteq_{\mathbf{P};\mathbf{Z}} I_2$ is similarly defined.)

We use the following circumscription related results.

Proposition 2 (Lifschitz, 1994). For any sentence B not containing P, Z, CIRC[F(P, Z) $\wedge$ B; P; Z] $\equiv$ CIRC[F(P, Z); P; Z] $\wedge$ B.

Proposition 14 (Lifschitz, 1994). If $F(\mathbf{P}, \mathbf{Z})$ is positive relative to each member P_i of $\mathbf{P}$, then the parallel circumscription CIRC[$F(\mathbf{P}, \mathbf{Z})$; $\mathbf{P}$; $\mathbf{Z}$] is equivalent to $\bigwedge_i$ CIRC[$F(\mathbf{P}, \mathbf{Z})$; P_i; $\mathbf{Z}$].

Proposition 15 (Lifschitz, 1994). CIRC[F; $\mathbf{P_0} > \mathbf{P_1} > \ldots > \mathbf{P_n}$; $\mathbf{Z}$] = $\bigwedge_{i=0}^{n}$ CIRC[F; $\mathbf{P}_i$; $\mathbf{P_{i+1}}, \ldots, \mathbf{P_n}$, $\mathbf{Z}$].

Theorem 9.4.6 (Shanahan, 1997). Let L1 and L2 are disjoint and compatible languages of first-order predicate calculus, Σ is a formula of L1, Δ is a formula of $L1 + L2$, ρ^* is a tuple of predicate symbols from L1, and σ is a tuple of predicate, function, and constant symbols from $L1+L2$ that includes every predicate, function and constant symbol in L2. If there exists a mapping F from interpretations of L1 to interpretations of L2 such that, for any interpretation M of L1 that satisfies Σ, M+F(M) is a model of Δ, then CIRC[$\Sigma \wedge \Delta; \rho^*; \sigma^*$] is equivalent to CIRC[$\Sigma; \rho^*; \sigma^*$] $\wedge\Delta$.

2.1.4 Negation Normal Form

A logical formula F is in *NNF* if (i) the only logical connectives connecting the subformulas are $\neg$, $\wedge$, $\vee$, and (ii) $\neg$ appears only in front of atomic statements (Barwise, 1977).

The following additional constraint must be satisfied by formulas containing aggregate formulas: (iii) $\neg$ does not appear in front of aggregate formulas; any $\neg(OP\langle \mathbf{x}.F' \rangle \succeq t)$ formula can be replaced by equivalent $(OP\langle \mathbf{x}.F' \rangle \prec t)$. By the first two conditions subformulas inside aggregate expressions are in NNF too, that is F' in $OP\langle \mathbf{x}.F' \rangle$ is in NNF.

2.1.5 Implicative Formula, Clark Normal Form, and Predicate Completion

A first-order formula is in *implicative* form if it is a conjunction of subformulas of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \longrightarrow P(\mathbf{x}))$, where P is a predicate symbol and $\phi(\mathbf{x})$ is a formula without implication. $\forall \mathbf{x}(\phi(\mathbf{x}) \longrightarrow P(\mathbf{x}))$ is referred to as an *axiom*. Individual $\phi(\mathbf{x})$ and $P(\mathbf{x})$ are referred to as *antecedent* and *consequent*, respectively, and collectively as *antecedents* and *consequents* of the formula.

A formula in implicative form is in *ClNF* if it is a conjunction of subformulas of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \longrightarrow P(\mathbf{x}))$, one for each unique predicate symbol P. *Completion* of a formula F in ClNF, denoted by *Comp*(F), is obtained from F by replacing each $\forall \mathbf{x}(\phi(\mathbf{x}) \longrightarrow P(\mathbf{x}))$ with $\forall \mathbf{x}(P(\mathbf{x}) \longleftrightarrow \phi(\mathbf{x}))$ (Ferraris & Lifschitz, 2010).

2.1.6 UNA Notation

$UNA[f_1, \ldots, f_n]$, where $f_1, \ldots, f_n$ are (possibly 0-ary) functions, expresses that $f_1, \ldots, f_n$ are injections with disjoint ranges, and stands for the axioms (i) $f_i(x_1, \ldots, x_k) \neq f_j(y_1, \ldots, y_l)$, for $i < j$ where f_i has arity k and f_j has arity l, and (ii) $f_i(x_1, \ldots, x_k) = f_i(y_1, \ldots, y_k) \longrightarrow (x_1 = y_1 \wedge \ldots \wedge x_k = y_k)$, for f_i of arity $k > 0$ (Baker, 1991).

2.2 Circumscriptive Event Calculus

Event Calculus is a subset of many-sorted first-order logic which provides predefined predicate and function constants, domain-independent axioms that define the semantics of those constants, and a default reasoning strategy. Dynamic domains are axiomatized through domain-specific axioms using the predefined predicate and function symbols (new symbols can also be used). The review follows the definitions given by Miller and Shanahan (1996).[2]

The sorts used in Event Calculus are listed in Table 2.1. $\mathcal{T}$ is interpreted as non-negative real numbers. The following function symbols, $Value$: $\mathcal{P} \times \mathcal{T} \mapsto \mathcal{R}$,

[2]Miller and Shanahan (2002) give an overview of many different and the latest versions of Event Calculus. The focus of the paper is descriptions of continuous and discrete changes in real-valued quantities, and Miller and Shanahan's (1996) version is the latest with respect to that. The extensions described in the paper are compatible with many of the later developments in other aspects of Event Calculus such as qualification of actions, etc. Discrete effects of actions with non-zero durations on quantities may require some changes however.

Table 2.1: Sorts used in the Event Calculus.

Name of Sort	Symbol	Variables
Actions	$\mathcal{A}$	a, a_1, a_2, ...
Fluents	$\mathcal{F}$	f, f_1, f_2, ...
Times	$\mathcal{T}$	t, t_1, t_2, ...
Quantities/Parameters	$\mathcal{P}$	p, p_1, p_2, ..., q, q_1, q_2, ...
Reals	$\mathcal{R}$	s, r, r_1, r_2, ...

δ: $\mathcal{P} \mapsto \mathcal{P}$, and $Next$: $\mathcal{T} \mapsto \mathcal{T}$, are used. $Value(\delta(P), T)$ represents the numerical value of first derivative of a quantity P at time T. $Next(T)$ is helpful in derivations and is used to refer to the least time point after T at which an action occurs if such a time point exists.

The following predicate symbols are used: , $InitializedTrue$, $InitializedFalse$, $Happens$, $HoldsAt$, $Initiates$, $Terminates$, $LeftContinuous$, $Continuous$, $Differentiable$, $RightLimit$, $BreaksTo$, and $Breaks$. Their sorting can be understood from their arguments in the axioms below. We have omitted $Releases$ predicate but its addition would not interfere with any of the extensions proposed in the paper.

$LeftContinuous(P, T)$, $Continuous(P, T)$, and $Differentiable(P, T)$ express that at time T, the function associated with quantity P (henceforth, just quantity P) is left-hand continuous, continuous, and differentiable, respectively. $RightLimit(P, T, R)$ expresses that at time T, the right limit value of quantity P is R. The mathematical definitions of $Continuous$, $Differentiable$, $LeftContinuous$, and $RightLimit$ are axiomatized below.

$$Continuous(p,t) \equiv \tag{M1}$$
$$\forall r \exists t_1 \forall t_2 [[|t - t_2| < t_1 \wedge 0 < r] \longrightarrow |Value(p,t) - Value(p,t_2)| < r]$$
$$Differentiable(p,t) \equiv \forall r \exists t_1 \forall t_2 [[0 < |t - t_2| < t_1 \wedge 0 < r] \longrightarrow \tag{M2}$$
$$\left| \frac{(Value(p,t) - Value(p,t_2))}{(t - t_2)} - Value(\delta(p), t) \right| < r]$$
$$LeftContinuous(p,t) \equiv \tag{M3}$$
$$\forall r \exists t_1 \forall t_2 [[t_2 < t \wedge (t - t_2) < t_1 \wedge 0 < r] \longrightarrow |Value(p,t) - Value(p,t_2)| < r]$$
$$RightLimit(p,t,r) \equiv \tag{M4}$$

$$\forall r_1 \exists t_1 \forall t_2 [[t < t_2 \wedge (t_2 - t) < t_1 \wedge 0 < r_1] \longrightarrow |Value(p, t_2) - r| < r_1]$$

Some fluents are directly affected by actions, while others are not. The former are called *frame fluent*s (or *primitive fluent*s). And, the latter are called *non-frame fluent*s (or *derived fluent*s), and they are also not assigned any initial values (Shanahan, 1999b). Since we omit *Releases* predicate here, for every domain there is a strict division of frame and non-frame fluents.

$Happens(A, T)$ expresses that action A occurs at time T. $HoldsAt(F, T)$ expresses that fluent F holds at time T. By default, the frame fluents are persistent and quantities are continuous. The break in the persistence for fluents is described using $Initiates$ or $Terminates$, whereas change in the continuity for quantities is described using $BreaksTo$ and $Breaks$. $Initiates(A, F, T)$ expresses that action A initiates fluent F at time T (in other words, if A occurs at T it would initiate F). $Termintes(A, F, T)$ on the other hand expresses that action A terminates fluent F at time T. $BreaksTo(A, P, T, R)$ expresses that at time T, an occurrence of action A will cause quantity P to instantaneously take on value R. $Breaks(A, P, T)$ expresses that at time T, action A potentially causes a discontinuity in quantity P.

Axioms (EC1)–(EC4) express the default persistence of fluents, and initiation and termination effects when corresponding actions happen. We modify Axioms (EC1)–(EC2) to avoid inconsistencies that otherwise result when a fluent which is initialized true is terminated at time 0 and vice-versa, similarly to modifications by Miller and Shanahan (2002).

$$HoldsAt(f, t) \longleftarrow [InitialisedTrue(f) \wedge \neg \exists a(Happens(a, 0) \wedge Terminates(a, f, 0)) \wedge \neg Clipped(0, f, t)] \quad \text{(EC1)}$$

$$\neg HoldsAt(f, t) \longleftarrow [InitialisedFalse(f) \wedge \neg \exists a(Happens(a, 0) \wedge Initiates(a, f, 0)) \wedge \neg Declipped(0, f, t)] \quad \text{(EC2)}$$

$$HoldsAt(f, t_2) \longleftarrow [Happens(a, t_1) \wedge Initiates(a, f, t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1, f, t_2)] \quad \text{(EC3)}$$

$$\neg HoldsAt(f, t_2) \longleftarrow \quad \text{(EC4)}$$

$$[Happens(a, t_1) \wedge Terminates(a, f, t_1) \wedge t_1 < t_2 \wedge \neg Declipped(t_1, f, t_2)]$$

$$Clipped(t_1, f, t_2) \equiv^{def} \exists a, t[Happens(a, t) \wedge t_1 < t < t_2 \wedge Terminates(a, f, t)] \quad \text{(EC5)}$$

$$Declipped(t_1, f, t_2) \equiv^{def} \exists a, t[Happens(a, t) \wedge t_1 < t < t_2 \wedge Initiates(a, f, t)] \quad \text{(EC6)}$$

Axiom (EC7) expresses that quantities are left-hand continuous at every time-point, including those at which actions occur. In other words, any discontinuity caused by an action takes effect immediately after the action occurs. Axioms (EC8)–(EC9) express that functions associated with quantities are continuous and differentiable by default.

$$LeftContinuous(p, t) \quad \text{(EC7)}$$

$$\neg[Happens(a, t) \wedge Breaks(a, p, t)] \longrightarrow Continuous(p, t) \quad \text{(EC8)}$$

$$\neg[Happens(a, t) \wedge Breaks(a, \delta(p), t)] \longrightarrow Differentiable(p, t) \quad \text{(EC9)}$$

Axiom (EC10) constrains *RightLimit* in terms of *BreaksTo* and *Happens*. Axiom (EC11) expresses the relationship between *BreaksTo* and *Breaks*, and Axiom (EC12) expresses that an action that potentially causes a discontinuity in a given quantity also causes potential discontinuity in its higher derivatives.

$$[BreaksTo(a, p, t, r) \wedge Happens(a, t)] \longrightarrow RightLimit(p, t, r) \quad \text{(EC10)}$$

$$BreaksTo(a, p, t, r) \longrightarrow Breaks(a, p, t) \quad \text{(EC11)}$$

$$Breaks(a, p, t) \longrightarrow Breaks(a, \delta(p), t) \quad \text{(EC12)}$$

Axioms (EC13)–(EC15) axiomatize the definition of $Next(T)$.[3]

$$t < Next(t) \quad \text{(EC13)}$$

$$[t < t_1 \wedge t_1 < Next(t)] \longrightarrow \neg Happens(a, t_1) \quad \text{(EC14)}$$

[3]Note that if T_1 is the point of the last occurrence of any action for a given domain, then $Next(T_1)$ is not tightly defined, and it could be any T_2 such that $T_1 < T_2$.

$$[Happens(a_1, t_1) \wedge t < t_1] \longrightarrow \exists a.Happens(a, Next(t)) \qquad \text{(EC15)}$$

Axioms (EC1)–(EC15) constitute the domain-independent axioms which are associated with every domain description.

2.2.1 Circumscription Policy CIRC$_{CEC}$

The frame problem is resolved in the circumscriptive Event Calculus by modeling the following default assumptions: by default a given action does not occur at a given time point, by default an action does not change the value of a given fluent or a given quantity, and by default a given action occurrence does not result in a discontinuity for a given quantity, through circumscription.

Let D denotes a collection of domain-specific axioms. *Nar(D)*, *Eff(D)*, *Inst(D)*, *Con(D)*, *Cnst(D)*, *Una(D)* stand for axioms describing, respectively, the *narrative* ($Happens$ facts and statements about the initial values of fluents), the effects of actions on fluents (using $Initiates$, $Terminates$), the instantaneous effects of actions on quantities (using $Breaks$ and $BreaksTo$), the mathematical constraints between quantities in different circumstances, the relationship between the values of fluents and optionally, quantities (using $HoldsAt$), the uniqueness-of-names. The axioms in D not covered in the above categories are included in *Rem(D)*.

The circumscription policy, CIRC$_{CEC}(D)$, for modeling the default assumptions (which are domain-independent) is given below. EC refers to the conjunction of following domain-independent axioms: $[(EC1) \wedge \ldots (EC10) \wedge (EC13) \wedge \ldots \wedge (EC15)]$ The extensions of predicates are minimized separately in different parts, and this strategy is referred to as the forced separation strategy (Shanahan, 1997), which is based on the idea of filter preferential entailment (Sandewall, 1989b). CIRC$_{CEC}(D)$ is referred to as Event Calculus theory for domain D.

$$\begin{aligned} CIRC_{CEC}(D) = & CIRC[Nar(D); Happens, InitialisedTrue, InitialisedFalse] \\ & \wedge CIRC[Eff(D); Initiates, Terminates] \\ & \wedge CIRC[Inst(D) \wedge (EC11) \wedge (EC12); Breaks; BreaksTo] \\ & \wedge Con(D) \wedge Una(D) \wedge Cnst(D) \wedge Rem(D) \wedge EC \end{aligned}$$

2.2.2 User-Defined Domain-Specific Axioms

We review the syntax for domain-specific axioms for describing changes to fluents from (Mueller, 2006; Lee & Palla, 2012b)[4], and enumerate those for quantities and additive effects.

Let $\succeq \in \{<, \leq, \geq, >, =, \neq\}$ and $\succ \in \succeq \backslash \{=\}$. Let me (and me_i) denote algebraic expressions over reals defined using $\odot$. Let $\eta \in \{Initiates, Terminates\}$, $\sigma \in \{Started, Stopped\}$, and $\pi \in \{Initiated, Terminated\}$.

A condition (γ) is defined recursively as follows[5]:

- $\tau_1 = \tau_2$ and $\tau_1 \neq \tau_2$, where τ_1 and τ_2 are terms, are conditions;
- $HoldsAt(f, t)$ and $\neg HoldsAt(f, t)$ are conditions;
- $me_1 \succeq me_2$ is a condition;
- If γ_1 and γ_2 are conditions, then $\gamma_1 \wedge \gamma_2$ and $\gamma_1 \vee \gamma_2$ are conditions;
- If v is a variable and γ is a condition only on the states of fluents, $\exists v \gamma$ is a condition.

$Ini(D)$ describes the initial values using $InitiallyTrue$ and $InitiallyFalse$ with axioms of the form:

- $\gamma \longrightarrow InitiallyTrue(f)$, and
- $\gamma \longrightarrow InitiallyFalse(f)$,

where γ is possibly an empty condition on terms only.

$Occ(D)$ describes action occurrences with axioms of the form:

- $\gamma \longrightarrow Happens(a, t)$ (occurrences of exogenous actions as well as triggered actions),
- $\sigma(f, t) \wedge \pi_1(f_1, t) \wedge \ldots \wedge \pi_n(f_n, t) \longrightarrow Happens(e, t)$ (causal constraints), and
- $Happens(e, t) \longrightarrow Happens(e_1, t) \vee \ldots \vee Happens(e_n, t)$ (disjunctive event axioms).

$Nar(D)$ refers to $Ini(D) \wedge Occ(D)$.

$Eff(D)$ describes the effects of actions on fluents with axioms of the form:

- $\gamma \longrightarrow \eta(e, f, t)$,
- $\gamma \wedge \eta_1(e, f_1, t) \longrightarrow \eta_2(e, f_2, t)$ (effect constraints), and

[4]We omit abnormality predicates, but they are not precluded by the methods we develop in this article.

[5]New predicate symbols can also be used, but their semantics is defined by the users.

- $\gamma \wedge [\neg]Happens(e_1, t) \wedge \ldots \wedge [\neg]Happens(e_n, t) \longrightarrow \eta(e, f, t)$ (effects of concurrent actions, etc.).

$Inst(D)$ describes discontinuous changes in the values of quantities and/or breaks in continuities of quantities caused by actions with axioms of the form:

- $\gamma \longrightarrow Breaks(a, p, t)$, and
- $\gamma \longrightarrow BreaksTo(a, p, t, r)$.

$Con(D)$ describes continuous changes in the values of quantities – via autonomous ODEs or algebraic equations – with axioms of the form:

- $\gamma \longrightarrow Value(p, t) = me$, and
- $\neg px, r.PartValue(p, t, px, r) \longrightarrow Value(p, t) = 0$.

$PCnst(D)$ describes mathematical constraints with axioms of the form:

- $\gamma \longrightarrow me_1 \succ me_2$.

$FCnst(D)$ describes constraints other than mathematical constraints with axioms of the form:

- γ or $\gamma \longrightarrow \gamma_f$, where γ_f is free of quantities (state constraints),
- $Happens(e_1, t) \wedge \gamma \longrightarrow [\neg]Happens(e_2, t)$ (event occurrence constraints), and
- $Happens(e, t) \longrightarrow \gamma$ (action precondition axioms).

Finally, $UNA(D)$ describe the unique name assumptions, and $Rem(D)$ contains axioms that do not fall into any of the above categories.

2.3 Stable Model Semantics

We review the first-order stable model semantics and the SM transformation (Section 2.3.1), predicate dependency graph (Section 2.3.2), Splitting Theorem (Section 2.3.3), and canonical formulas (Section 2.3.4). Since the Event Calculus descriptions are canonical, their circumscription can be interpreted in terms of their SM transformations. The SM transformation gives a circumscription-like definition for first-order stable model semantics. The SM transformations of parts of Event Calculus descriptions can be equivalently represented as the SM transformation of the whole set of axioms because of the nature of the predicate dependency graph of the axioms involved which allows for the application of the Splitting Theorem.

2.3.1 First-order Stable Model Semantics

Our review of first-order stable model semantics follows the definition introduced by Ferraris, Lee, and Lifschitz (2007), which was extended to logic with aggregate formulas by Lee and Meng (2009). Their definition assumes the following set of primitive propositional connectives and quantifiers: $\bot$ (falsity), $\wedge$, $\vee$, $\longrightarrow$, $\forall$, $\exists$. $\neg F$ is treated as an abbreviation of $F \longrightarrow \bot$and $F \longleftrightarrow G$ stands for $(F \longrightarrow G) \wedge (G \longrightarrow F)$. We will refer to this notation as the *sm-notation*.

A model of a formula F is stable relative to tuple of predicate constants $\mathbf{P} = \mathrm{P}_1, \ldots, \mathrm{P}_n$, if it satisfies $SM_{\mathbf{P}}[F]$. $SM_{\mathbf{P}}[F] = F \wedge \neg\exists\mathbf{p}((\mathbf{p} < \mathbf{P}) \wedge F^*(\mathbf{p}))$, where $\mathbf{p}$ is a list of distinct predicate variables $\mathrm{p}_1, \ldots, \mathrm{p}_n$, and $F^*(\mathbf{p})$ is defined recursively:

- $P_i(\mathbf{t})^* = p_i(\mathbf{t})$ for any tuple $\mathbf{t}$ of terms;
- $F^* = F$ for any atomic F (including $\bot$ and equality) that does not contain members of $\mathbf{P}$;
- $(F \wedge G)^* = F^* \wedge G^*$;
- $(F \vee G)^* = F^* \vee G^*$;
- $(F \longrightarrow G)^* = (F^* \longrightarrow G^*) \wedge (F \longrightarrow G)$;
- $(\forall\mathbf{x}F)^* = \forall\mathbf{x}F^*$;
- $(\exists\mathbf{x}F)^* = \exists\mathbf{x}F^*$;
- $(OP\langle\mathbf{x}.F\rangle \succeq t)^* = (OP\langle\mathbf{x}.F^*\rangle \succeq t) \wedge (OP\langle\mathbf{x}.F\rangle \succeq t)$

Proposition 1 from (Lee & Palla, 2012b) reviewed below states that a stable model relative to tuple of some predicate constants in a formula can be equivalently rewritten as a stable model relative to the tuple of all predicate constants in the formula. *pr(F)* denotes the list of all predicate constants occurring in F. *Choice(*$\mathbf{P}$*)* denotes the conjunction of *choice formulas* $\forall\mathbf{x}(P(\mathbf{x}) \vee \neg P(\mathbf{x}))$ for all predicate constants P in $\mathbf{P}$. *False*($\mathbf{P}$) denotes the conjunction of $\forall\mathbf{x}\neg P(\mathbf{x})$ for all predicate constants P in $\mathbf{P}$.

Proposition 1 (Lee & Palla, 2012b). $SM[F;\mathbf{P}] \longleftrightarrow SM[F \wedge Choice(pr(F) \setminus \mathbf{P}) \wedge False(\mathbf{P} \setminus pr(F))]$, is logically valid.

2.3.2 Positive Occurrence and Predicate Dependency Graph

This review follows the definitions by (Ferraris, Lee, Lifschitz, & Palla, 2009).

An occurrence of a predicate constant, or any other subexpression, in a formula is called *positive* if the number of implications containing that occurrence in the antecedent is even, and *strictly positive* if that number is 0. An occurrence of a predicate constant in a formula is *negated* if it belongs to a subformula of the form $\neg F$ (or, $F \longrightarrow \bot$), and *nonnegated* otherwise. For example, in the formula, $Fo_1 = Q(x) \wedge \neg R(x) \longrightarrow P(x)$, P and R are positive, P is strictly positive, and R is negated. A formula F is said to be *negative* on a tuple $\mathbf{P}$ of predicate constants if members of $\mathbf{P}$ have no strictly positive occurrences in F. F is negative on $\{Q, R\}$ for example.

A *rule* of a first-order formula F is a strictly positive occurrence of an implication in F. For any first-order formula F, the *predicate dependency graph* of F (relative to the list $\mathbf{P}$ of predicates) is the directed graph that has all predicates as its vertices, and has an edge from P to Q if, for some rule $G \longrightarrow H$ of F,

- P has a strictly positive occurrence in H, and
- Q has a positive nonnegated occurrence in G.

$DG_{\mathbf{P}}[F]$ denotes the predicate dependency graph of F relative to $\mathbf{P}$. For example, $DG_{\{P,Q,R\}}[Fo_1]$ has one edge from P to Q.

2.3.3 Splitting Theorem

This review follows the definitions by Ferraris et al. (2009). The statement for the Splitting Theorem is reviewed below.

Splitting Theorem (Ferraris et al., 2009). Let F, G be first-order sentences, and let $\mathbf{P}$, $\mathbf{Q}$ be disjoint tuples of distinct predicate constants. If

- each strongly connected component of $DG_{\mathbf{PQ}}[F \wedge G]$ is a subset of $\mathbf{P}$ or a subset of $\mathbf{Q}$,
- F is negative on $\mathbf{Q}$, and
- G is negative on $\mathbf{P}$

then, $SM[F \wedge G; \mathbf{P}, \mathbf{Q}] \equiv SM[F; \mathbf{P}] \wedge SM[G, \mathbf{Q}]$, holds true.

The splitting theorem can be used to represent a conjunction of stable models of formulas F and G with respect to disjoint tuples of predicates as stable models

of $F \wedge G$ with respect to the union of the tuples of predicates. It is proven with help of the Splitting Lemma given below.

Splitting Lemma, Version 1 (Ferraris et al., 2009). Let F be a first-order sentence, and let $\mathbf{P}$, $\mathbf{Q}$ be disjoint tuples of distinct predicate constants. If each strongly connected component of $DG_{\mathbf{PQ}}[F]$ is a subset of $\mathbf{P}$ or a subset of $\mathbf{Q}$ then, $SM[F; \mathbf{P}, \mathbf{Q}] \equiv SM[F; \mathbf{P}] \wedge SM[F; \mathbf{Q}]$, holds true.

The Splitting Lemma is proven with help of the Lemmas given below.

Lemma 1 (Ferraris et al., 2009). $(\mathbf{p} \leq \mathbf{P}) \wedge F^*(\mathbf{p}) \longrightarrow F$ is logically valid.

Lemma 2 (Ferraris et al., 2009). If F is negative on $\mathbf{P}$ then, $(\mathbf{p} \leq \mathbf{P}) \longrightarrow (F^*(\mathbf{p}) \longleftrightarrow F)$, is logically valid.

Lemma 3 (Ferraris et al., 2009). Let $\mathbf{P_1}$, $\mathbf{P_2}$ be disjoint lists of distinct predicate constants, and let $\mathbf{p_1}$, $\mathbf{p_2}$ be disjoint lists of distinct predicate variables of the same length as $\mathbf{P_1}$, $\mathbf{P_2}$ respectively.

(a) If every positive occurrence of every predicate constant from $\mathbf{P_2}$ in F is negated then, $((\mathbf{p_1}, \mathbf{p_2}) \leq (\mathbf{P_1}, \mathbf{P_2})) \wedge F^*(\mathbf{p_1}, \mathbf{P_2}) \longrightarrow F^*(\mathbf{p_1}, \mathbf{p_2})$, is logically valid.

(b) If every nonpositive occurrence of every predicate constant from $\mathbf{P_2}$ in F is negated then, $((\mathbf{p_1}, \mathbf{p_2}) \leq (\mathbf{P_1}, \mathbf{P_2})) \wedge F^*(\mathbf{p_1}, \mathbf{p_2}) \longrightarrow F^*(\mathbf{p_1}, \mathbf{P_2})$, is logically valid.

Lemma 4 (Ferraris et al., 2009). Let $\mathbf{P_1}$, $\mathbf{P_2}$ be disjoint lists of distinct predicate constants such that $DG_{\mathbf{P_1},\mathbf{P_2}}[F]$ has no edges from predicate constants in $\mathbf{P_1}$ to predicate constants in $\mathbf{P_2}$, and let $\mathbf{p_1}$, $\mathbf{p_2}$ be disjoint lists of distinct predicate variables of the same length as $\mathbf{P_1}$, $\mathbf{P_2}$ respectively. Formula, $((\mathbf{p_1}, \mathbf{p_2}) \leq (\mathbf{P_1}, \mathbf{P_2})) \wedge F^*(\mathbf{p_1}, \mathbf{p_2}) \longrightarrow F^*(\mathbf{p_1}, \mathbf{P_2})$, is logically valid.

2.3.4 Canonical Formulas

This review follows the definitions by Lee and Palla (2012), which also subscribes to the sm-notation. For example, in $P(A) \wedge Q(B) \wedge \forall x(P(x) \wedge \neg Q(x) \longrightarrow R(x))$, both occurrences of Q are positive, but only the first one is strictly positive.

A formula F is *canonical* relative to a list $\mathbf{P}$ of predicate constants if:

• no occurrence of a predicate constant from **P** is in the antecedents of more than one implications in F, and

• every occurrence of a predicate constant from **P** that is in the scope of a strictly positive occurrence of $\exists$ or $\vee$ in F is strictly positive in F.

For example, $\forall x(\neg P(x) \longrightarrow Q(x))$ is not canonical relative to {P, Q} (it does not satisfy the first clause of the definition), but is canonical relative to {Q}. The formula, $\forall x(P(x) \vee \neg P(x))$, is not canonical relative to {P} (it does not satisfy the second clause of the definition). The formula, $P(A) \wedge (\exists x P(x) \longrightarrow \exists x Q(x))$ is canonical relative to {P, Q}, while $P(A, A) \wedge \exists x(P(x, A) \longrightarrow P(B, x))$ is not canonical relative to {P, Q} (it does not satisfy the second clause of the definition).

Non-canonical formulas can often be equivalently rewritten as canonical formulas. For example, non-canonical formula, $\forall x(\neg P(x) \longrightarrow Q(x))$, is equivalent to $\forall x(P(x) \vee Q(x))$, which is canonical relative to {P, Q}. According to Theorem 1 from Lee and Palla (2012), for any canonical formula F relative to **P**, the circumscription and SM transformations are equivalent. Since any equivalent transformation preserves the models of circumscription, the result can be applied to non-canonical formulas, by first rewriting them as canonical formulas.

Theorem 1 (Lee & Palla, 2012b). For any canonical formula F relative to **P**, $CIRC[F; \mathbf{P}] \longleftrightarrow SM[F; \mathbf{P}]$ is logically valid.

Theorem 1 follows from Lemmas 3 and 4 from (Lee & Palla, 2012b).

Lemma 3 (Lee & Palla, 2012b). If every occurrence of every predicate constant from **P** is strictly positive in F, $(\mathbf{p} \leq \mathbf{P}) \longrightarrow (F^*(\mathbf{p}) \longleftrightarrow F(\mathbf{p}))$, is logically valid.

Lemma 4 (Lee & Palla, 2012b). If F is canonical relative to **P**, $(\mathbf{p} \leq \mathbf{P}) \wedge F \longrightarrow (F^*(\mathbf{p}) \longleftrightarrow F(\mathbf{p}))$, is logically valid.

Using Theorem 1 and Proposition 1 (Section 2.3.1) from Lee and Palla (2012) , and the Splitting Theorem from Ferraris et al. (2009), Lee and Palla (2012) showed that any Event Calculus description can be expressed in terms of the SM operator. The statement of Theorem 2 from Lee and Palla (2012) reviewed below has been slightly modified according to the notations and concepts used in this

article. The version of Event Calculus referred to by Lee and Palla (2012) does not make a distinction between fluents and quantities, and continuous changes can only be described as functions of time (not ODEs, that is).

Theorem 2 (Lee & Palla, 2012b). For any Event Calculus theory, the following theories are equivalent to each other:

1. $CIRC[Nar(D); Happens, InitializedTrue, InitializedFalse] \wedge CIRC[Eff(D); Initiates, Terminates] \wedge EC$;
2. $SM[Nar(D); Happens, InitializedTrue, InitializedFalse] \wedge SM[Eff(D); Initiates, Terminates] \wedge EC$;
3. $SM[Nar(D) \wedge Eff(D) \wedge EC; Happens, InitializedTrue, InitializedFalse, Initiates, Terminates]$;
4. $SM[Nar(D) \wedge Eff(D) \wedge EC \wedge Choice(pr(Nar(D) \wedge Eff(D) \wedge EC) \backslash \{Happens, InitializedTrue, InitializedFalse, Initiates, Terminates\})]$.

The above theorem shows that circumscription of the parts of the Event Calculus descriptions can be equivalently formulated as stable models of the parts. Furthermore, the stable models of the parts can be equivalently reformulated in terms of stable models of the whole Event Calculus descriptions.

2.3.5 HEX-Programs

A HEX-program (Eiter, Ianni, Schindlauer, & Tompits, 2005) consists of rules

$$a_1 \vee \ldots \vee a_k \ \text{:-} \ b_1, \ldots, b_l, not \ b_{l+1}, \ldots, not \ b_m,$$

where each a_i is a classical literal and b_j is either a classical literal or an external literal of the form $\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_r)$, where g is the name of an external predicate, the $Y_1 \ldots Y_n$ and $X_1 \ldots X_r$ are two lists of terms called *input* and *output* lists respectively. (In the standard answer-set programs each a_i and b_j is a classical literal.) g is evaluated externally, where the value of input terms is passed as input. Intuitively, an external atom provides a way for deciding the truth value of an

output tuple, for a given set of input literals, depending on the extensions of input predicates. The semantics of HEX-programs are defined in terms of the FLP-reduct semantics (Faber, Leone, & Pfeifer, 2004) (see Eiter et al., 2005; Redl, Eiter, and Krennwallner, 2011 for more details).

Bartholomew, Lee, and Meng (2011) show that while in general the first-order FLP semantics do not coincide with the first-order stable model semantics, they coincide for the formulas with (first-order logic representations of) rules of the type:

$$a_1 \vee \ldots \vee a_k \ \text{:-} \ b_1, \ldots, b_l, not \ b_{l+1}, \ldots, not \ b_m,$$

where each a_i is a classical literal, $b_1, \ldots, b_l$ are classical literals or aggregate expressions not containing negations or nested aggregates, and $b_{l+1}, \ldots, b_m$ are classical literals. For exact details please refer to Theorem 7 from Bartholomew et al. (2011).

CHAPTER 3
CLOSED-WORLD REASONING FOR FIRST-ORDER LOGIC FORMULAS WITH AGGREGATES

An aggregate expression, $\#sum\langle y, \mathbf{x}.F(y,\mathbf{x})\rangle$, where $\mathbf{x}$ may be empty, generates a new term, which equals the sum of (integer or real) terms bound to y. Change in the bindings from $(y, \mathbf{x}.F(y,\mathbf{x}))^I$, for some interpretation I, usually changes the new term or the summation. Under the standard first-order semantics, for a certain binding for variables y and $\mathbf{x}$, if $F(y,\mathbf{x})$ is not known true or false, then it can be assumed to be either (true or false). Thus, given an aggregate formula, $\#sum\langle y, \mathbf{x}.F(y,\mathbf{x})\rangle \succeq t$ and an interpretation I such that $(\#sum\langle y, \mathbf{x}.F(y,\mathbf{x})\rangle \succeq t)^I$ is true, another interpretation I', with a different interpretation for formulas not known true or false such that $(OP\langle y, \mathbf{x}.F(y,\mathbf{x})\rangle \succeq t)^{I'}$ is false, often exists.

Consider the following example.

$$(\#sum\langle x.P(x)\rangle < 1 \longrightarrow Q(1)). \qquad (Fo_1)$$

$\{Q(1)\}$ is a model of Fo_1. The extension of P is $\emptyset$ and $\#sum\langle x.P(x)\rangle < 1$ is true under $\{Q(1)\}$. $\{P(1)\}$ is also a model of Fo_1, and $\#sum\langle x.P(x)\rangle < 1$ is false under $\{P(1)\}$.

Many applications would like to incorporate closed-world-like assumption when the aggregate expression, $\#sum\langle y, \mathbf{x}.F(y,\mathbf{x})\rangle$, is evaluated, such that for a given binding for variables y and $\mathbf{x}$, if $F(y,\mathbf{x})$ is not known true, then it must be false. When an expression, aggregate expression such as $\#sum\langle y, \mathbf{x}.F(y,\mathbf{x})\rangle$ or existentiality check such as $\exists y, \mathbf{x}.F(y,\mathbf{x})$ or $\neg\exists y, \mathbf{x}.F(y,\mathbf{x})$, is evaluated under the afore-stated closed-world-like assumption, we say that the expression is used as a *constraint*. If a model contains a fact that is not known to be true (that is, a ground term that is not known to be true but is assumed to be true) then such facts are considered

Portions of this chapter to appear in: Khandelwal, A., & Fox, P. (2013). General Descriptions of Additive Effects via Aggregates in the Circumscriptive Event Calculus. *Journal of Artificial Intelligence Research* (JAIR).

to be *unjustified*. For example, when aggregate summation is used to compute the net effect of additive effects, aggregate summation is intended for use as constraints. The aggregate expression in Fo_1 is not evaluated as a constraint under the model {P(1)}, for example, and P(1) is an unjustified fact. In other words, {P(1)} is not a model of Fo_1 when its aggregate expression is used as a constraint.

Circumscription is a common nonmonotonic reasoning technique for formalizing closed-world-like assumptions. For example, it is used to formalize the default assumptions regarding occurrences of actions, effects of actions on fluents and quantities, etc., in Event Calculus. However, circumscription is not adequate to formalize the use of aggregate expressions as constraints. For example, {P(1)} is a model of CIRC[Fo_1; P, Q].

The models of CIRC[F; $\mathbf{P}$; $\mathbf{Z}$] are minimal models, minimal with respect to $\subset_{\mathbf{P};\mathbf{Z}}$, and we refer to the minimal models containing unjustified facts as *weak* models. We formally define weak models in Section 3.1 and an alternative circumscription-like transformation, CIRC^A, to eliminate the weak models in Section 3.2. In Section 3.3, we discuss how a value different from zero for an aggregate summation when the aggregate expression is not satisfied may be accommodated. In Section 3.4, we show equivalent first-order forms for CIRC^A transformations of formulas with stratified aggregates. Finally, in Section 3.5, we compare the $CIRC^A$ transformation and first-order stable model semantics.

3.1 Weak Models from Standard Circumscription

Consider the following example:

$$\#sum\langle x.P(x)\rangle < 1 \longrightarrow Q(1). \qquad (Fo_1)$$

$\{Q(1)\}$ is a model of CIRC[Fo_1; P, Q]. The extension of P is $\emptyset$ and $\#sum\langle x.P(x)\rangle < 1$ is true under $\{Q(1)\}$. $\{P(1)\}$ is also a model of CIRC[Fo_1; P, Q], and $\#sum\langle x.P(x)\rangle < 1$ is false under $\{P(1)\}$. Under an open-world assumption, P(1) that is not known to be true or false can be assumed to be either. If the aggregate formula in Fo_1 is used as a constraint, however, $P(1)$ is an unjustified fact and $\{P(1)\}$ is a weak model

of Fo_1.

Intuitively, a minimal model contains an unjustified fact when a smaller interpretation satisfies the formula after the interpretation of aggregate formulas interpreted as constraints has been fixed.

Definition 3.1.1. Let $\mathbf{P}$ be a tuple of predicate constants in F, $\mathbf{Z}$ be a tuple of constants in F, and $\mathbf{p}$, $\mathbf{z}$ are variables corresponding to $\mathbf{P}$, $\mathbf{Z}$. The *FA-transformation* of $F(\mathbf{P}, \mathbf{Z})$ into $F^{FA}(\mathbf{p}, \mathbf{z})$ is obtained by replacing all constants from $\mathbf{P}$ not inside any aggregate expression of F with corresponding variables from $\mathbf{p}$, and all constants from $\mathbf{Z}$ not inside any aggregate expression that contains constants from $\mathbf{P}$ by variables from $\mathbf{z}$.

The FA-transformation of Fo_1 is defined as:

$$Fo_1^{FA}([p,q]) = (\#sum\langle x.P(x)\rangle < 1 \longrightarrow q(1)).$$

$F^{FA}(\mathbf{p}, \mathbf{z})$ and $F(\mathbf{p}, \mathbf{z})$ are similar except that constants from $\mathbf{P}$, $\mathbf{Z}$ inside aggregate expressions are not substituted by variables $\mathbf{p}$, $\mathbf{z}$ in the former. A minimal model of F contains an unjustified fact if $\exists \mathbf{p}, \mathbf{z}(\mathbf{p} < \mathbf{P} \wedge F^{FA}(\mathbf{p}, \mathbf{z}))$ is valid. For example, model {P(1)} of Fo_1 contains an unjustified fact because the interpretation, {}, satisfies Fo_1^{FA}([p,q]).

$$(\#sum\langle x.P(x)\rangle \geq 1 \longrightarrow P(1)) \wedge (\#sum\langle x.P(x)\rangle < 1 \longrightarrow Q(1)) \qquad (Fo_2)$$

But, unjustified facts cannot be deduced using the FA-transformation in all cases. Consider a recursive formula, Fo_2. Its FA-transformation is defined as:

$$Fo_2^{FA}([p,q]) = (\#sum\langle x.P(x)\rangle \geq 1 \longrightarrow p(1)) \wedge (\#sum\langle x.P(x)\rangle < 1 \longrightarrow q(1))$$

{P(1)} is a minimal model of Fo_2, and P(1) is an unjustified fact. But, the smaller interpretation, {}, is not a model of Fo_2^{FA}.

Definition 3.1.2. Let $\mathbf{P}$ be a tuple of predicate constants in F, $\mathbf{Z}$ be a tuple of constants in F, and $\mathbf{p}$, $\mathbf{z}$ are variables corresponding to $\mathbf{P}$, $\mathbf{Z}$. The ***-transformation* of $F(\mathbf{P}, \mathbf{Z})$ into $F^{**}(\mathbf{p}, \mathbf{z})$ is obtained in the following steps:

a) Rewrite F in NNF.

b) Replace constants from $\mathbf{P}$ and $\mathbf{Z}$ not inside an aggregate expression by corresponding variables from $\mathbf{p}$ and $\mathbf{z}$. Replace constants from $\mathbf{Z}$ inside aggregate expressions of depth zero (relative to F) by corresponding variables from $\mathbf{z}$ if no constant from $\mathbf{P}$ is inside the aggregate expressions.

c) Replace aggregate formulas of depth zero (relative to F) containing constants from $\mathbf{P}$, $OP\langle \mathbf{x}.\ F'(\mathbf{P}, \mathbf{Z})\rangle \succeq t$, by $(OP\langle \mathbf{x}.F'(\mathbf{p}, \mathbf{z})\rangle \succeq t) \vee (OP\langle \mathbf{x}.F'(\mathbf{P}, \mathbf{Z})\rangle \succeq t)$. □

The **-transformation can be used to formally define models with unjustified facts. $F^{**}(\mathbf{p}, \mathbf{z})$ is the same as $F(\mathbf{p}, \mathbf{z})$ whenever F has no aggregate expressions or none of the aggregate expressions contain constants from $\mathbf{P}$. Generally, if F is in NNF, then $F^{**}(\mathbf{p}, \mathbf{z})$ and $F(\mathbf{p}, \mathbf{z})$ are similar, except that there is an additional disjunct $(OP\langle \mathbf{x}.F'(\mathbf{P}, \mathbf{Z})\rangle \succeq t)$ in $F^{**}(\mathbf{p}, \mathbf{z})$, for every $(OP\langle \mathbf{x}.F'(\mathbf{p}, \mathbf{z})\rangle \succeq t)$, of depth zero, when F' contains constants from $\mathbf{P}$. Thus, Lemma 3.1.1, that every model of $F(\mathbf{p}, \mathbf{z})$ is also a model of $F^{**}(\mathbf{p}, \mathbf{z})$, follows. Similarly, $F^{FA}(\mathbf{p}, \mathbf{z}) \longrightarrow F^{**}(\mathbf{p}, \mathbf{z})$ is also valid.

Lemma 3.1.1. *$F(\mathbf{p}, \mathbf{z}) \longrightarrow F^{**}(\mathbf{p}, \mathbf{z})$*

The first step of the **-transformation ensures that formulas with the same canonical form (NNF) have equivalent **-transformations. Aggregate formulas that are subject of the third step, that is the aggregate formulas containing predicate constants whose extensions are minimized, are interpreted as constraints. Lemma 3.1.2 gives an equivalent **-transformation for implicative formulas such that the transformed formula retains implicative form.

Lemma 3.1.2. *The **-transformation of an implicative formula $F(\mathbf{P}, \mathbf{Z})$, a conjunction of $\forall \mathbf{x}(\phi(\mathbf{x}) \longrightarrow \rho(\mathbf{x}))$ axioms, into $F^{**}(\mathbf{p}, \mathbf{z})$ can equivalently be described in the following steps:*

a) Rewrite ϕ in NNF.

b) Replace constants from $\mathbf{P}$ and $\mathbf{Z}$ not inside an aggregate expression, including ρ, by corresponding variables from $\mathbf{p}$ and $\mathbf{z}$. Replace constants from $\mathbf{Z}$ inside

aggregate expressions of depth zero (relative to F or ϕ) by corresponding variables from $\mathbf{z}$ *if no constant from* $\mathbf{P}$ *is inside the aggregate expressions.*

c) Replace aggregate formulas of depth zero (relative to F or ϕ) containing constants from $\mathbf{P}$, $OP\langle \mathbf{x}.F'(\mathbf{P},\mathbf{Z})\rangle \succeq t$, *by* $(OP\langle \mathbf{x}.F'(\mathbf{p},\mathbf{z})\rangle \succeq t) \wedge (OP\langle \mathbf{x}.F'(\mathbf{P},\mathbf{Z})\rangle \succeq t)$.

Henceforth, we will use the **-transformation defined in Lemma 3.1.2 for implicative formulas. The **-transformations of Fo_1, Fo_2 are defined as:

$$Fo_1^{**}([p,q]) = ([\#sum\langle x.p(x)\rangle < 1 \wedge \#sum\langle x.P(x)\rangle < 1] \longrightarrow q(1))$$
$$Fo_2^{**}([p,q]) = ([\#sum\langle x.P(x)\rangle \geq 1 \wedge \#sum\langle x.p(x)\rangle \geq 1] \longrightarrow p(1)) \wedge ([\#sum\langle x.P(x)\rangle < 1 \wedge \#sum\langle x.p(x)\rangle < 1] \longrightarrow q(1)).$$

A minimal model of $F(\mathbf{P}, \mathbf{Z})$ contains an unjustified fact iff $\exists \mathbf{p},\mathbf{z}(\mathbf{p} < \mathbf{P} \wedge F^{**}(\mathbf{p},\mathbf{z}))$ is valid.

Definition 3.1.3. A model of CIRC[F; $\mathbf{P}$; $\mathbf{Z}$] is *weak*, in relation to interpreting the aggregate formulas containing constants from $\mathbf{P}$ as constraints, if it also satisfies $\exists \mathbf{p},\mathbf{z}(\mathbf{p} < \mathbf{P} \wedge F^{**}(\mathbf{p},\mathbf{z}))$.

CIRC[Fo_1; P, Q] has infinitely many models, {Q(1)}, {P(2)}, {P(3)}, {P(0.5), P(0.6)}, {P(0.2), P(0.3), P(0.6)}, etc., and all but {Q(1)} are weak. CIRC[Fo_2; P, Q] has two models, {Q(1)} and {P(1)}, and {P(1)} is weak. We discuss a few more examples:

$$(\#sum\langle x.P(x)\rangle < 1 \longrightarrow Q(1)) \wedge (P(2)) \qquad (Fo_3)$$
$$(\#sum\langle x.P(x)\rangle = s \longrightarrow Q(s)) \wedge (P(2)) \qquad (Fo_4)$$
$$(\#sum\langle x.P(x)\rangle < 1 \longrightarrow Q(1)) \wedge (\#sum\langle x.Q(x)\rangle > 0.5 \longrightarrow P(1.5)). \qquad (Fo_5)$$

CIRC[Fo_3; P, Q] has one model, {P(2)}, which is not weak. CIRC[Fo_4; P, Q] has infinitely many models, {P(2), Q(2)}, {P(2), P(3), Q(5)}, {P(2), P(4), Q(6)}, {P(2), P(3), P(4), Q(9)}, etc., and all but {P(2), Q(2)} are weak. CIRC[Fo_5; P, Q] has infinitely many models, {P(1.5)}, {Q(1), Q(-0.6)}, {Q(1), Q(-0.7)},{Q(1),

Q(-0.3), Q(-0.4)}, etc., all of which are weak.[6]

In the following we extend circumscription to first-order logic with aggregates such that the transformation for aggregate formulas is different.

3.2 CIRC^A Transformation

Definition 3.2.1. Let F be a first-order formula, $\mathbf{P}$ is a tuple of predicate constants in F, and $\mathbf{Z}$ is a tuple of constants in F. Then, CIRC$^A[F;\ \mathbf{P};\ \mathbf{Z}]$ defines a transformation for F into a second-order formula as given below:

$$CIRC^A[F;\mathbf{P};\mathbf{Z}] = F(\mathbf{P},\mathbf{Z}) \wedge \neg\exists\mathbf{p},\mathbf{z}(\mathbf{p} < \mathbf{P} \wedge F^{**}(\mathbf{p},\mathbf{z})),$$

where $\mathbf{p}$ and $\mathbf{z}$ are tuples of variables corresponding to constants in tuples $\mathbf{P}$ and $\mathbf{Z}$. The models of CIRC$^A[F;\ \mathbf{P};\ \mathbf{Z}]$ are referred to as *ag-minimal* models. □

CIRC$^A[F;\ \mathbf{P};\ \mathbf{Z}]$ = CIRC$[F;\ \mathbf{P};\ \mathbf{Z}]$ when F contains no aggregates because $F^{**}(\mathbf{p},\ \mathbf{z})$ is the same as $F(\mathbf{p},\ \mathbf{z})$. Besides, only the interpretation for aggregate formulas containing constants from $\mathbf{P}$ is different from the standard circumscription, and that of the rest including negation is the same.

Lemma 3.1.1 holds even for $\mathbf{p} < \mathbf{P}$, leading to Proposition 3.2.1, that every ag-minimal model of F is a minimal model of F.

Proposition 3.2.1. $CIRC^A[F;\mathbf{P};\mathbf{Z}] \longrightarrow CIRC[F;\mathbf{P};\mathbf{Z}]$

Proposition 3.2.2, that all ag-minimal models are not weak and all non-weak models are ag-minimal models, follows trivially from Definition 3.1.3 (and Proposition 3.2.1).

The converse of Proposition 3.2.1, however, is not true. The **-transformation helps to get the minimal models. In addition, it accepts only those models where aggregate formulas containing predicates from $\mathbf{P}$ are interpreted as constraints. Assuming that F is in NNF, the difference between $F^{**}(\mathbf{p},\ \mathbf{z})$ and $F(\mathbf{p},\ \mathbf{z})$ is formulas, $(OP\langle\mathbf{x}.F'(\mathbf{p},\mathbf{z})\rangle \succeq t) \vee (OP\langle\mathbf{x}.F'(\mathbf{P},\mathbf{Z})\rangle \succeq t)$, for different F' and t, in the former,

[6] We have only described weak models for CIRC[F; P, Q], but all other forms of circumscription — CIRC[F; P], CIRC[F; Q], CIRC[F; P > Q], CIRC[F; Q > P] — accept weak models for one or more example formulas.

in the place of $(OP\langle \mathbf{x}.F'(\mathbf{p},\mathbf{z})\rangle \succeq t)$ in the latter. When smaller interpretations, according to $\subset_{\mathbf{P};\mathbf{Z}}$, of a model of F are considered, then F^{**} cannot be falsified if the only impact of the difference between the two interpretations is to make some aggregate formulas that were true in the model to be false in the smaller interpretations. This is covered by the disjunct $(OP\langle \mathbf{x}.F'(\mathbf{P},\mathbf{Z})\rangle \succeq t)$. Therefore, some smaller interpretations that are not models of $F(\mathbf{p}, \mathbf{z})$ may be models of $F^{**}(\mathbf{p}, \mathbf{z})$. As a result, some models of CIRC[F; $\mathbf{P}$; $\mathbf{Z}$] may not be models of CIRC^A[F; $\mathbf{P}$; $\mathbf{Z}$] when they have smaller interpretations that are models of $F^{**}(\mathbf{p}, \mathbf{z})$.

In fact, ag-minimal models may not always exist and CIRC^A transformation of a formula may be inconsistent even though the formula and its circumscription are consistent. For example, CIRC^A[Fo_5; P, Q] has no models, even though CIRC[Fo_5; P, Q] has infinitely many models: {P(1.5)}, {Q(1), Q(-0.6)}, {Q(1), Q(-0.7)}, {Q(1), Q(-0.3), Q(-0.4)}, etc.

Proposition 3.2.2. *A model of* $CIRC[F;\mathbf{P};\mathbf{Z}]$ *is not weak, per Definition 3.1.3, iff it is also a model of* $CIRC^A[F;\mathbf{P};\mathbf{Z}]$.

Notation 3.2.2. $\mathbf{Z}$ and $\mathbf{z}$ may be omitted if they are empty. That is, notations CIRC^A[F; $\mathbf{P}$] and $F^{**}(\mathbf{p})$, like CIRC[F; $\mathbf{P}$] and $F(\mathbf{p})$, imply that $\mathbf{Z}$ and $\mathbf{z}$ are empty. Predicate symbols which occur in some aggregate expression in F are referred to as the *aggregate predicate symbols* of F. $\sigma_{\mathbf{x}}$ denotes variable mappings from variables in $\mathbf{x}$ to constants, and $\sigma_{\mathbf{x}}(\phi)$, where ϕ is a formula, denotes ϕ after variables in $\mathbf{x}$ are substituted for the corresponding constants.

If $\mathbf{P}$ is a set of aggregate predicate symbols disjoint from constants $\mathbf{Z}$ in F and $\mathbf{P}$ is partitioned into $\mathbf{P}_1$ and $\mathbf{P}_2$, then CIRC[F; $\mathbf{P}$; $\mathbf{Z}$] $\wedge$ CIRC^A[F; $\mathbf{P}_1$; $\mathbf{Z}$] accepts only those models of F that are minimal with respect to the extensions of $\mathbf{P}$ and not weak (only) with respect to $\mathbf{P}_1$ (that is, with respect to interpreting aggregate formulas containing constants from $\mathbf{P}_1$ as constraints).

If F is in implicative form, all predicate symbols in the consequents are aggregate predicate symbols or they do not appear in the antecedents, and the aggregate predicate symbols occur only inside aggregate formulas in the antecedents, then in any model of CIRC^A transformation of F, any aggregate predicate fact is inferred to be true only if at-least one of the relevant antecedents is true.

Proposition 3.2.3. *Let F be in implicative form with predicate symbols $\mathbf{P}$, where $\mathbf{P}$ contains all aggregate predicate symbols in F and predicate symbols that do not occur in any antecedent. If all predicate symbols in the consequents of F are from $\mathbf{P}$ and predicate symbols in $\mathbf{P}$ occur only in aggregate formulas in the antecedents then $CIRC^A[F;\mathbf{P}] \longrightarrow Comp(F)$ is valid.*

Proof. We assume, without loss of generality, that F is in Clark Normal Form. Let M is a model of CIRC^A[F; $\mathbf{P}$], $G \longrightarrow P(\mathbf{x})$ is an axiom in F, and M contains a fact $\sigma_{\mathbf{x}}(P(\mathbf{x}))$ such that $\sigma_{\mathbf{x}}(G)$ is not true under M. Then, $\mathrm{M}' = M \setminus \sigma_{\mathbf{x}}(P(\mathbf{x}))$ is a model of F^{**}. But then M cannot be an ag-minimal model, which leads to a contradiction. Therefore, $P(\mathbf{x}) \longrightarrow G$ must also be valid. □

The implication does not hold in the other direction. For example, $Comp(Fo_2)$, shown below, has two models {Q(1)} and {P(1)}. However, {P(1)} is not a model of $CIRC^A[Fo_2; P, Q]$.

$$([x = 1 \wedge \#sum\langle y.P(y)\rangle \geq 1] \longleftrightarrow P(x)) \wedge ([x = 1 \wedge \#sum\langle y.P(y)\rangle < 1] \longleftrightarrow Q(x))$$

Intuitively, completion allows circular justifications, including self-justifications, which are not allowed under CIRC^A transformation (that is, facts must be independently derived). P(1) in {P(1)} cannot be independently derived. Let F be in Clark Normal Form, M is a model of F, Δ is a subset of facts of predicate constants in $\mathbf{P}$ that are true in M, and $I = M \setminus \Delta$ (the interpretation that is the same as M but with facts in Δ assumed to be false). Δ forms a circular justification if for all $\sigma_{\mathbf{x}}(P(\mathbf{x}))\in\Delta$, for any predicate P, M satisfies $\sigma_{\mathbf{x}}(G)$ but I does not satisfy $\sigma_{\mathbf{x}}(G)$ where $G \longrightarrow P(\mathbf{x})$ is an axiom in F.

In the following, we look at a few examples. In these examples ag-minimal models refer to the models of CIRC^A[F; P, Q] for any formula F and $\mathbf{P}$ = [P, Q].

$$\forall x(\neg P(x) \longrightarrow Q(x)) \qquad (Fo_6)$$

$$\forall x(P(x) \vee Q(x)) \qquad (Fo_7)$$

$\text{CIRC}^A[Fo_6;\ \text{P},\ \text{Q}] \equiv \text{CIRC}^A[Fo_7;\ \text{P},\ \text{Q}] \equiv \forall x((\neg P(x) \wedge Q(x)) \vee (\neg Q(x) \wedge P(x)))$.

$$(\#sum\langle x.P(x)\rangle < 1 \longrightarrow Q(2)) \qquad (Fo_8)$$

$$([z = \#sum\langle x.P(x)\rangle \wedge z < 1] \longrightarrow Q(2)) \qquad (Fo_9)$$

Fo_8 and Fo_9 are equivalent but their **-transformations are different.

$$Fo_8^{**}([p,q]) = ([\#sum\langle x.p(x)\rangle < 1 \wedge \#sum\langle x.P(x)\rangle < 1] \longrightarrow q(2))$$
$$Fo_9^{**}([p,q]) = ([z = \#sum\langle x.p(x)\rangle \wedge z = \#sum\langle x.P(x)\rangle \wedge z < 1] \longrightarrow q(2))$$

{P(0.5), Q(2)} is a model of Fo_8 and Fo_9. While {P(0.5), Q(2)} does not satisfy Fo_8^{**}, it is a model of Fo_9^{**}. {q(2), P(0.5), Q(2)} is a model of Fo_8^{**}, however. Anyway, {Q(2)} is the only ag-minimal model of Fo_8 and Fo_9.

$$(Q(1) \vee Q(2)) \wedge (Q(1) \longrightarrow \#sum\langle x.P(x)\rangle < 1) \qquad (Fo_{10})$$

$$(Q(1) \vee Q(2)) \wedge (Q(1) \longrightarrow \#sum\langle x.P(x)\rangle \geq 1) \qquad (Fo_{11})$$

The **-transformations of Fo_{10} and Fo_{11} are given above.

$$Fo_{10}^{**}([p,q]) = (q(1) \vee q(2)) \wedge (q(1) \longrightarrow [\#sum\langle x.p(x)\rangle < 1 \vee \#sum\langle x.P(x)\rangle < 1)])$$
$$Fo_{11}^{**}([p,q]) = (q(1) \vee q(2)) \wedge (q(1) \longrightarrow [\#sum\langle x.p(x)\rangle \geq 1 \vee \#sum\langle x.P(x)\rangle \geq 1)])$$

Fo_{10} has two ag-minimal models: {Q(1)}, {Q(2)}. Fo_{11} has just one ag-minimal model: {Q(2)}.

$$(\#sum\langle x.P(x)\rangle \geq 1 \longrightarrow Q(2)) \wedge (\neg P(1) \longrightarrow Q(3)) \qquad (Fo_{12})$$

$$([\exists x.P(x) \wedge \#sum\langle x.P(x)\rangle < 1] \longrightarrow Q(2)) \wedge (\neg\exists x.P(x) \longrightarrow Q(3)) \qquad (Fo_{13})$$

Fo_{12} has two ag-minimal models: {Q(3)} and {P(1), Q(2)}. Fo_{13} has infinitely many ag-minimal models: {Q(3)}, {P(0.5), Q(2)}, {P(0), Q(2)}, {P(-1), Q(2)}, ..., {P(1)}, {P(2)}, etc. It has many other minimal models which are not ag-minimal models: {P(0.5), P(0.6)}, {P(-0.1), P(0.5), P(0.7)}, etc. $(Q(3) \vee \exists x.P(x))$ entails

that Q(3) is true or the extension of P is non-empty. When Q(3) is not true in a model, ag-minimal models contain exactly one (true) ground fact for P(x). If $\neg\exists x.P(x)$ is interpreted as a constraint, all the ag-minimal models of Fo_{13} except {Q(3)} contain unjustified facts. We elaborate on this in the following section.

Discussions in the remainder are restricted to implicative formulas.

3.3 When Sum of Empty Multiset is Not to be Zero

$\#sum\langle \mathbf{x}.F'(\mathbf{x})\rangle$ is zero when $F'(\mathbf{x})$ is not satisfied. At times, we want to use different values when $F'(\mathbf{x})$ is not satisfied. For example, a value may be defined either directly or as a sum of components such that the summation is used only when at-least one component is defined (otherwise, the value is given directly). We discuss how such a separation can be axiomatized.

$$(\exists \mathbf{x}.F'(\mathbf{x}) \wedge v = \#sum\langle \mathbf{x}.F'(\mathbf{x})\rangle) \vee (\neg\exists \mathbf{x}.F'(\mathbf{x}) \wedge v = Default(\mathbf{w})) \qquad \text{(C1)}$$

Let F be a formula, $\mathbf{v}$ is a tuple of fresh variables, and $F_{DD'}$ is obtained from F by replacing some aggregate expressions in some of the axioms with variables from $\mathbf{v}$ and appending expressions of the form (C1) to the antecedent of the axioms as a conjunct for the corresponding variables in $\mathbf{v}$. (C1)-like expressions give the value for the corresponding variables from $\mathbf{v}$. Variable v in (C1) equals $\#sum\langle \mathbf{x}.F'(\mathbf{x})\rangle$ when $F'(\mathbf{x})$ is satisfied and equals the value returned by the $Default$ function otherwise. $\mathbf{x}$, v, $\mathbf{w}$ are disjoint set of variables. Let $F^{**'}_{DD'}(\mathbf{p})$ be obtained similarly from $F^{**}(\mathbf{p})$ using (C2)-like expressions. Formula (C1) is transformed into (C2) by applying a similar transformation as in the third step of Lemma 3.1.2 to the formulas checking existence (and lack of it) in addition to the aggregate formulas.

$$\begin{aligned}&(\exists \mathbf{x}.F'(\mathbf{p}) \wedge \exists \mathbf{x}.F'(\mathbf{P}) \wedge v = \#sum\langle \mathbf{x}.F'(\mathbf{p})\rangle \wedge v = \#sum\langle \mathbf{x}.F'(\mathbf{P})\rangle) \qquad \text{(C2)}\\ &\vee (\neg\exists \mathbf{x}.F'(\mathbf{p}) \wedge \neg\exists \mathbf{x}.F'(\mathbf{P}) \wedge v = Default(\mathbf{w}))\end{aligned}$$

The ag-minimal models of Fo_{13} hint that (C1)-like expressions may not work as only the ag-minimal models of $F_{DD'}$ that also satisfy $\neg\exists \mathbf{p}(\mathbf{p} < \mathbf{P} \wedge F^{**'}_{DD'}(\mathbf{p}))$ are of interest. The latter condition check can be managed by CIRC^A transformation

if aggregate formulas are used to check satisfiability of the aggregate expressions, as shown in (C3). While $\#count\langle \mathbf{x}.F'(\mathbf{x})\rangle \neq 0$ and $\exists \mathbf{x}.F'(\mathbf{x})$ are equivalent, they are interpreted differently under CIRC^A transformation. The former is treated as a constraint whereas the latter is not.

$$(\#count\langle \mathbf{x}.F'(\mathbf{x})\rangle \neq 0 \wedge v = \#sum\langle \mathbf{x}.F'(\mathbf{x})\rangle) \\ \vee (\#count\langle \mathbf{x}.F'(\mathbf{x})\rangle = 0 \wedge v = Default(\mathbf{w})) \tag{C3}$$

Definition 3.3.1. Let F be a formula, $\mathbf{v}$ is a tuple of fresh variables, and F_{DD} is obtained from F by replacing some aggregate expressions with fresh variables from $\mathbf{v}$ and appending expressions of the form (C3) to the antecedent as a conjunct for the corresponding variables in $\mathbf{v}$. (C3)-like expressions give the value for the corresponding variables from $\mathbf{v}$. F_{DD} is referred to as a *DD-derivative* of F.[7]

A DD-derivative of Fo_3 is shown below, as example. Proposition 3.3.1 defines a relationship between F_{DD} and $F_{DD'}$ with regards to their CIRC^A transformations.

$$Fo_{3_{DD}} = ([((\#count\langle x.P(x)\rangle \neq 0 \wedge v = \#sum\langle x.P(x)\rangle) \\ \vee (\#count\langle x.P(x)\rangle = 0 \wedge v = Default())) \wedge v < 1] \longrightarrow Q(1)) \wedge (P(2))$$

Proposition 3.3.1. $CIRC^A[F_{DD}; \mathbf{P}] \equiv (CIRC^A[F_{DD'}; \mathbf{P}] \wedge \neg\exists \mathbf{p}(\mathbf{p} < \mathbf{P} \wedge F^{**'}_{DD'}(\mathbf{p})))$.

Proof. Follows from: $F^{**'}_{DD'}(\mathbf{p}) \longleftrightarrow F^{**}_{DD}(\mathbf{p})$, $F_{DD} \longleftrightarrow F_{DD'}$, and $F^{**}_{DD'}(\mathbf{p}) \longrightarrow F^{**'}_{DD'}(\mathbf{p})$. □

3.4 Stratified Aggregates

Along the lines of stratified programs (Przymusinski, 1989) and aggregate-stratified DLP$^{\mathcal{A}}$ programs (Faber, Pfeifer, Leone, Dell'armi, & Ielpa, 2008), we define *hierarchical* (implicative) formulas, which are also *positive hierarchical* if they obey a few more restrictions.

Definition 3.4.1. Let F be an implicative formula and $\mathbf{P}$ is a tuple of subset of predicate symbols in F. F is said to be *hierarchical in* $\mathbf{P}$ if there exists an assignment

[7] *DD* suggests **D**ifferent **D**efault value, different from zero.

of ordinal levels to symbols in $\mathbf{P}$ such that for each axiom, (i) if predicate constant appearing in the consequent is in $\mathbf{P}$, then it is of strictly higher level than any predicate constant from $\mathbf{P}$ in the antecedent (including in an aggregate expression), and (ii) if the predicate constant appearing in the consequent is not in $\mathbf{P}$, then the antecedent has no predicate from $\mathbf{P}$. Using the assignment, predicate constants in $\mathbf{P}$ can be partitioned into $\{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n\}$ such that predicate constants in $\mathbf{P}_i$ have ordinal levels i. Such a partitioning is referred to as a *stratification*. Any predicate constant in $\mathbf{P}$ that does not appear in consequent of any of the axioms in F is assumed to be in $\mathbf{P}_0$. $\mathbf{P}_0$ may be empty. Further, F can be partitioned into $F_0 \wedge F_1 \wedge \ldots \wedge F_n$ such that the axioms with consequent predicate symbols having ordinal levels i are in F_i for $i > 0$. Axioms with consequents not in $\mathbf{P}$ are in F_0. This partitioning is referred to as an *F-partitioning*.

Definition 3.4.2. A hierarchical formula F is said to be *positive hierarchical in* $\mathbf{P}$ if all aggregate predicate symbols from $\mathbf{P}$ in the antecedents are inside some aggregate expression or not inside a negated formula.

Fo_{12}, Fo_{13} are hierarchical in [P, Q], for example, but they are not positive hierarchical in [P, Q]. Fo_1, Fo_3, Fo_4 are positive hierarchical in [P, Q]. It is easy to verify that when F is (positive) hierarchical, F^{**} and its DD-derivatives are (positive) hierarchical too (Lemmas 3.4.1, 3.4.2).

Lemma 3.4.1. *$F^{**}(\mathbf{p})$ is (positive) hierarchical in $\mathbf{p}$ if $F(\mathbf{P})$ is (positive) hierarchical in $\mathbf{P}$. And if $F_0 \wedge F_1 \wedge \ldots \wedge F_n$ is an F-partition then $F_0^{**} \wedge F_1^{**} \wedge \ldots \wedge F_n^{**}$ is an F^{**}-partition.*

Lemma 3.4.2. *Let F_{DD} be a DD-derivative of F. $F_{DD}(\mathbf{p})$ is (positive) hierarchical in $\mathbf{p}$ if $F(\mathbf{P})$ is (positive) hierarchical in $\mathbf{P}$.*

Notation 3.4.3. We assume that the antecedents of implicative formulas are in NNF. Let formula F be hierarchical in $\mathbf{P}$ and $\mathbf{P}$ is a tuple of all aggregate predicate symbols in F. Let $\{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n\}$ be a stratification with corresponding F-partition $F_0 \wedge F_1 \wedge \ldots \wedge F_n$. F is assumed to be satisfiable. The models of CIRC[F; $\mathbf{P}_0 > \mathbf{P}_1 > \ldots > \mathbf{P}_n$; $\mathbf{Z}$] are referred to as *pr-minimal* models of F.

According to Lemma 3.4.3, a model for lower strata of F can be extended to the higher strata of F. That is, a model for higher strata can always be constructed from a given model for lower strata.

Notation 3.4.4. $M[C]$ denotes the extension of symbol C in the interpretation M. Let M_1 and M_2 be two interpretations for constants, then $M = M_1 + M_2$ represents interpretation for union of the constants such that $M[C]$ equals $M_1[C]$ if C belongs to M_1 and $M_2[C]$ if C belongs to M_2. $M_1[C] = M_2[C]$ if C belongs to both M_1 and M_2.

Lemma 3.4.3. *Let M_1 be a model of $F_0 \wedge \ldots \wedge F_i$, for some $0 \leq i < n$. M_1 also has interpretations for all constants not in $\mathbf{P}$. Then, an interpretation M_2 of constants in $\mathbf{P_{i+1}}, \ldots, \mathbf{P_n}$ can be constructed such that $M_1 + M_2$ is a model of F.*

Proof. For any k, $0 \leq k < n$, if extensions of constants in $\mathbf{P_0}, \ldots, \mathbf{P_k}$ are given then Comp(F_{k+1}) gives unique extensions for constants in $\mathbf{P_{k+1}}$. Therefore, a model for $\bigwedge_{j=i+1}^{n}$ Comp(F_j) can be created from M_1. Let that model be M_2. Then $M_1 + M_2$ is a model of F. □

The extension for predicates in $\mathbf{P}_0$ is empty in any pr-minimal model because none of the predicates in $\mathbf{P}_0$ appear in the consequents of F.

Lemma 3.4.4. *CIRC[F; $\mathbf{P_0} > \mathbf{P_1} > \ldots > \mathbf{P_n}$] $\longrightarrow \bigwedge_{P \in \mathbf{P}_0} \forall \mathbf{x}.\neg P(\mathbf{x})$*

Proof. CIRC[F; $\mathbf{P_0} > \mathbf{P_1} > \ldots > \mathbf{P_n}$] = $\bigwedge_{i=0}^{n}$ CIRC[F; $\mathbf{P}_i$; $\mathbf{P_{i+1}}, \ldots, \mathbf{P_n}$], by Proposition 15 of (Lifschitz, 1994). Since none of the predicates in $\mathbf{P}_0$ appear in the consequent side in F, extension of $\mathbf{P}_0$ is empty in any model of CIRC[F; $\mathbf{P}_0$; $\mathbf{P_1}, \ldots, \mathbf{P_n}$]. □

According to Lemma 3.4.5, given a model of F, M_1, if there exists another model of F, M_2, which has smaller or equal extensions for all predicates in lower strata, then a model for F^{**} can be constructed by taking the extensions for predicates in the lower strata from M_2 and that for predicates in the higher strata from M_1.

Notation 3.4.5. Let M satisfies $F(\mathbf{P}, \mathbf{Z})$, then $F^{**}(\mathbf{p}, \mathbf{z})$ *corresponding to* M refers to all $F^{**}(\mathbf{p}, \mathbf{z})$ such that the interpretation for constants from $\mathbf{P}$, $\mathbf{Z}$ (occurring in F^{**}) is given by M.

Lemma 3.4.5. *Let F be positive hierarchical in* $\mathbf{P}$*, and* M_1*,* M_2 *are models of F such that for some* $0 \leq i < n$, $\bigwedge_{j=0}^{i} \bigwedge_{C \in P_j} M_2[C] \subseteq M_1[C]$ *and* $\bigwedge_{C \notin \mathbf{P}} M_2[C] = M_1[C]$*. Consider* $F^{**}(\mathbf{p})$ *corresponding to* M_1*. Then* $M_3 = \sum_{j=0}^{i} \sum_{C \in \mathbf{P_j}} M_2[C] + \sum_{j=i+1}^{n} \sum_{C \in \mathbf{P_j}} M_1[C] + \sum_{C \notin \mathbf{P}} M_1[C]$ *is a model of* $F^{**}(\mathbf{p})$*.*

Proof. M_1 and M_2 are models of $F_0^{**} \wedge \ldots \wedge F_n^{**}$, by Lemma 3.1.1. Since M_2 and M_3 coincide on extensions of $\mathbf{P_0}, \ldots, \mathbf{P_i}$, M_3 is a model of $F_0^{**} \wedge \ldots \wedge F_i^{**}$. We will show that M_3 is a model of $F_{i+1}^{**} \wedge \ldots \wedge F_n^{**}$ as well.

Predicates from $\mathbf{P_0}, \ldots, \mathbf{P_i}$ appear only in the antecedents of $F_{i+1} \wedge \ldots \wedge F_n$. By construction, $\bigwedge_{j=0}^{i} \bigwedge_{C \in P_j} M_3[C] \subseteq M_1[C]$ is valid. $M_1 = \sum_{j=0}^{i} \sum_{C \in P_j} M_1[C] + \sum_{j=i+1}^{n} \sum_{C \in P_j} M_1[C] + \sum_{C \notin \mathbf{P}} M_1[C]$ is a model of $F_{i+1}^{**} \wedge \ldots \wedge F_n^{**}$. Since predicates from $\mathbf{P_0}, \ldots, \mathbf{P_i}$ appear either in positive literals or in some conjunction of aggregate expressions of the form $OP\langle \mathbf{x}.F'(\mathbf{p})\rangle \succeq t \wedge OP\langle \mathbf{x}.F'(\mathbf{P})\rangle \succeq t$, where $OP\langle \mathbf{x}.F'(\mathbf{P})\rangle \succeq t$ is determined by M_1 (and $OP\langle \mathbf{x}.F'(\mathbf{p})\rangle \succeq t$ is determined by M_3), smaller extensions for $\mathbf{P_0}, \ldots, \mathbf{P_i}$, can only cause certain antecedents of $F_{i+1}^{**} \wedge \ldots \wedge F_n^{**}$ that were true under M_1 to be false under M_3. That is, none of the antecedents of $F_{i+1}^{**} \wedge \ldots \wedge F_n^{**}$ which were false under M_1 can be true under M_3. Therefore, M_3 satisfies $F_{i+1}^{**} \wedge \ldots \wedge F_n^{**}$ as well. □

Proposition 3.4.6 states that all pr-minimal models of F are ag-minimal models. The converse does not necessarily hold. For example, CIRC[Fo_{12}; P > Q] has just one model, {Q(3)}, as opposed to two ag-minimal models of Fo_{12}, {Q(3)} and {P(1), Q(2)}. However, if F is positive hierarchical then the converse holds as well (Proposition 3.4.7). Proposition 3.4.6 follows from Lemma 3.4.3 and Proposition 3.4.7 follows from Lemma 3.4.5.

Proposition 3.4.6. $CIRC[F; \mathbf{P_0} > \mathbf{P_1} > \ldots > \mathbf{P_n}] \longrightarrow CIRC^A[F; \mathbf{P}]$.

Proof. Let M_1 be a pr-minimal model. Assume that M_1 is not an ag-minimal model. Consider $F^{**}(\mathbf{p})$ corresponding to M_1. Then there exists a model M_2 of $F^{**}(\mathbf{p})$ such that

$\bigwedge_{C\in\mathbf{P}} M_2[C] \subseteq M_1[C] \wedge \exists_{C\in\mathbf{P}} M_2[C] \subset M_1[C] \wedge \bigwedge_{C\notin\mathbf{P}} M_2[C] = M_1[C]$ is valid.

Let i be such that $\bigwedge_{j=0}^{i-1} \bigwedge_{C\in\mathbf{P_j}} M_2[C] = M_1[C] \wedge \exists_{C\in\mathbf{P_i}} M_2[C] \subset M_1[C]$ is valid. Since, by Lemma 3.4.4, $\bigwedge_{C\in\mathbf{P_0}} M_1[C] = \emptyset$ is valid, $i > 0$. Let $\mathrm{M}_3 = \sum_{j=0}^{i} \sum_{C\in\mathbf{P_i}} M_2[C] + \sum_{C\notin\mathbf{P}} M_2[C]$. M_3 is a model of $F_0^{**} \wedge \ldots \wedge F_i^{**}$ and $F_0 \wedge \ldots \wedge F_{i-1}$. We will show that M_3 is a model of $F_0 \wedge \ldots \wedge F_i$ as well.

Since antecedents of F_i contain predicate symbols from $\mathbf{P}_0$, ..., $\mathbf{P}_{i-1}$ only, and extensions of M_3 and M_1 coincide for these predicates, antecedents of F_i^{**} are true under M_3 iff antecedents of F_i are true under M_1, and M_3. Thus, since M_3 is a model of F_i^{**}, M_3 is a model of F_i.

By Lemma 3.4.3, we can construct an interpretation M_4 such that $M_3 + M_4$ is a model of F. But then M_1 cannot be a pr-minimal model, which leads to a contradiction. □

Proposition 3.4.7. *If F is positive hierarchical in $\mathbf{P}$, then*

$$CIRC^A[F;\mathbf{P}] \equiv CIRC[F;\mathbf{P_0} > \mathbf{P_1} > \ldots > \mathbf{P_n}].$$

Proof. We show that $CIRC^A[F;\mathbf{P}] \longrightarrow CIRC[F;\mathbf{P_0} > \mathbf{P_1} > \ldots > \mathbf{P_n}]$. Since positive hierarchical formula is hierarchical, the other direction follows from Proposition 3.4.6.

Let M_1 be an ag-minimal model. Assume that M_1 is not a pr-minimal model. Then there exists another model M_2 of F such that, for some $i \geq 0$, (a) $M_2[C] = M_1[C]$ for all constants in $\mathbf{P_0}, \ldots, \mathbf{P_{i-1}}$, (b) $M_2[C] \subseteq M_1[C]$ for all constants in $\mathbf{P}_i$, (c) $M_2[C] \subset M_1[C]$ for some constant in $\mathbf{P}_i$, and (d) $M_2[C] = M_1[C]$ for all the constants not in $\mathbf{P}$.

Consider $F^{**}(\mathbf{p})$ corresponding to M_1. By Lemma 3.4.5, there exists another model M_3 that satisfies $F^{**}(\mathbf{p})$ such that, (a) $M_3[C] = M_2[C]$ for all constants in $\mathbf{P_0}, \ldots, \mathbf{P_i}$, (b) $M_3[C] = M_1[C]$ for all constants in $\mathbf{P_{i+1}}, \ldots, \mathbf{P_n}$, and (c) $M_3[C] = M_1[C]$ for all the constants not in $\mathbf{P}$. But then M_1 cannot be an ag-minimal model,

which leads to a contradiction. □

Proposition 3.4.8 states that prioritized circumscription of F is equivalent to the completion, and the extensions of all constants in $\mathbf{P}_0$ are empty.

Proposition 3.4.8. $CIRC[F; \mathbf{P_0} > \ldots > \mathbf{P_n}] \equiv \bigwedge\limits_{P \in \mathbf{P_0}} \forall \mathbf{x}.\neg P(\mathbf{x}) \wedge F_0 \wedge \bigwedge\limits_{j=1}^{n} Comp(F_j)$.

Proof. CIRC[F; $\mathbf{P_0} > \mathbf{P_1} > \ldots > \mathbf{P_n}$] = $\bigwedge\limits_{j=0}^{n}$ CIRC[F; $\mathbf{P}_j$; $\mathbf{P_{j+1}}, \ldots, \mathbf{P_n}$], by proposition 15 from (Lifschitz, 1994). By Lemma 3.4.4 $\bigwedge\limits_{P \in \mathbf{P_0}} \forall\ \mathbf{x}.\neg\ \mathrm{P}(\mathbf{x})$ is valid. By Lemma 3.4.3, a model for $F_{j+1} \wedge \ldots \wedge F_n$ can be created from the model for CIRC[$F_0 \wedge \ldots \wedge F_j$; $\mathbf{P}_j$; $\mathbf{P_{j+1}}, \ldots, \mathbf{P_n}$]. Thus, CIRC[$F_0 \wedge \ldots \wedge F_n$; $\mathbf{P}_j$; $\mathbf{P_{j+1}}, \ldots, \mathbf{P_n}$] = CIRC[$F_0 \wedge \ldots \wedge F_j$; $\mathbf{P}_j$; $\mathbf{P_{j+1}}, \ldots, \mathbf{P_n}$] $\wedge F_{j+1} \wedge \ldots \wedge F_n$, using theorem 9.4.6 from (Shanahan, 1997). CIRC[$F_0 \wedge \ldots \wedge F_j$; $\mathbf{P}_j$; $\mathbf{P_{j+1}}, \ldots, \mathbf{P_n}$] = CIRC[$F_0 \wedge \ldots \wedge F_j$; $\mathbf{P}_j$] because $F_1 \wedge \ldots \wedge F_j$ do not contain any predicate symbols from $\mathbf{P_{j+1}}, \ldots, \mathbf{P_n}$. CIRC[$F_0 \wedge \ldots \wedge F_j$; $\mathbf{P}_j$] = $F_0 \wedge \ldots \wedge F_{j-1} \wedge$ CIRC[F_j; $\mathbf{P}_j$], because $F_0 \wedge \ldots \wedge F_{j-1}$ do not contain predicate from $\mathbf{P_j}$. CIRC[F_j; $\mathbf{P}_j$] = Comp(F_j), by propositions 2 and 14 from (Lifschitz, 1994). □

Propositions 3.4.7 and 3.4.8 show that CIRC^A transformation of implicative formulas positive hierarchical in aggregate predicates, to minimize the extensions of the aggregate predicates, is equivalent to the predicate completion.

3.4.1 Hierarchical by Arguments of Predicates

Like predicate-based stratification, we can define term-wise stratification. Next we define term-wise stratification of formulas with a single aggregate predicate through function symbols. Recall that the extended Event Calculus has single aggregate predicate, $PartValue$ (Section 4.2.1.3).

Definition 3.4.6. Let F be an implicative formula, P is an n-ary predicate symbol, $n > 0$, $\mathbf{G}$ is a tuple of function symbols (of same or different arities), and k-th term, $k < n$, of all predicates with symbol P in F are of the form G($\mathbf{x}$) for some G $\in \mathbf{G}$ (arguments may have variables). F is said to be *k-th term hierarchical in* P with respect to $\mathbf{G}$ if there exists an assignment of ordinal levels to symbols in $\mathbf{G}$ such that for each axiom, (i) if P appears in the consequent, then the function symbol

in the k-th term is of strictly higher level than any function symbol occurring in the k-th term of a predicate with symbol P in the antecedent, and (ii) if the consequent is not a predicate with symbol P, then the antecedent contains no predicate with symbol P. Any two function symbols which are assigned different ordinal levels must have disjoint ranges. (Recall that functions G_1 and G_2 have disjoint ranges if $\forall \mathbf{xy}(G_1(\mathbf{x}) \neq G_2(\mathbf{y}))$ is valid.) Function symbols that are assigned same ordinal levels can have common ranges. *k-th term positive hierarchical in P* formula, stratification of function symbols, and *F*-partition follow from the definition of *k-th term hierarchical in P* formula analogously to predicate-based stratification.

Notation 3.4.7. Let P be the only aggregate predicate in F, F be k-th term positive hierarchical in P with respect to function symbols $\mathbf{G}$, and $\{\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_n\}$ is a stratification of $\mathbf{G}$ with corresponding F-partition $F_0 \wedge F_1 \wedge \ldots \wedge F_n$. F is assumed to be satisfiable.

Proposition 3.4.9 states that CIRC^A transformation of implicative formulas which are k-th term positive hierarchical in the aggregate predicate, to minimize the extension of the aggregate predicate, are equivalent to the predicate completion.

Proposition 3.4.9. $CIRC^A[F;P] \equiv \bigwedge_{G \in \mathbf{G_0}} \forall \mathbf{wxy}.\neg P(\mathbf{w}, G(\mathbf{x}), \mathbf{y}) \wedge F_0 \wedge \bigwedge_{j=1}^{n} Comp(F_j)$, *where* $\mathbf{w}$, $\mathbf{x}$, $\mathbf{y}$ *are distinct tuples of variables of sizes k−1, arity of G, n−k respectively.*

Proof sketch. A proof similar to the combined proofs of Propositions 3.4.7 and 3.4.8 for stratification by function symbols instead of stratification by predicate symbols can be constructed. □

Proposition 3.4.9 can also be derived using intra-predicate prioritized circumscription, in contrast to the standard inter-predicate prioritized circumscription, from Propositions 3.4.10 and 3.4.11.

Definition 3.4.8. If P and Q are n-ary predicates, G is a function symbol of arity m, $m \geq 0$, and $0 < k \leq n$, then $P_{k=G} \leqslant_k Q_{k=G}$ iff $\forall \mathbf{wxy}.(P(\mathbf{w}, G(\mathbf{x}), \mathbf{y}) \longrightarrow Q(\mathbf{w}, G(\mathbf{x}), \mathbf{y}))$ is true, where $\mathbf{w}$, $\mathbf{x}$, $\mathbf{y}$ are disjoint tuples of variables of sizes k−1, m, n−k, respectively. ($P_{k=G}$ is a relational algebraic notation to denote that the k-th

term is produced from function G.) This can be extended to tuple $\mathbf{G} = G_1, \ldots, G_l$ of functions. $P_{k=\mathbf{G}} \leqslant_k Q_{k=\mathbf{G}}$ iff $\forall \mathbf{w}\mathbf{y}(\forall \mathbf{x_1}(P(\mathbf{w}, G_1(\mathbf{x_1}), \mathbf{y}) \longrightarrow Q(\mathbf{w}, G_1(\mathbf{x_1}), \mathbf{y})) \wedge \ldots \wedge \forall \mathbf{x_l}(P(\mathbf{w}, G_l(\mathbf{x_l}), \mathbf{y}) \longrightarrow Q(\mathbf{w}, G_l(\mathbf{x_l}), \mathbf{y})))$ is valid. $P_{k=\mathbf{G}} =_k Q_{k=\mathbf{G}}$ iff $P_{k=\mathbf{G}} \leqslant_k Q_{k=\mathbf{G}} \wedge Q_{k=\mathbf{G}} \leqslant_k P_{k=\mathbf{G}}$, and $P_{k=\mathbf{G}} <_k Q_{k=\mathbf{G}}$ iff $P_{k=\mathbf{G}} \leqslant_k Q_{k=\mathbf{G}} \wedge \neg(Q_{k=\mathbf{G}} \leqslant_k P_{k=\mathbf{G}})$. Let $\mathbf{G}$ be partitioned into $\{\mathbf{G_1}, \ldots, \mathbf{G_l}\}$. Then, $P_{k=\mathbf{G}} \preceq_k Q_{k=\mathbf{G}}$ stands for $\bigwedge_{i=1}^{l}(\bigwedge_{j=1}^{i-1}(P_{k=\mathbf{G_j}} =_k Q_{k=\mathbf{G_j}}) \longrightarrow (P_{k=\mathbf{G_i}} \leqslant_k Q_{k=\mathbf{G_i}}))$. $P_{k=\mathbf{G}} \prec_k Q_{k=\mathbf{G}}$ iff $P_{k=\mathbf{G}} \preceq_k Q_{k=\mathbf{G}} \wedge \neg(Q_{k=\mathbf{G}} \preceq_k P_{k=\mathbf{G}})$. *Intra-predicate prioritized circumscription* of a formula *Fo* with respect to predicate P is defined as

$$ICIRC[Fo; P; k; \mathbf{G_1} > \ldots > \mathbf{G_k}; \mathbf{Z}] = Fo(P, \mathbf{Z}) \wedge \neg\exists p\mathbf{z}(Fo(p, \mathbf{z}) \wedge p_{k=\mathbf{G}} \prec_k P_{k=\mathbf{G}}).$$

Inter-predicate prioritized circumscription, denoted by CIRC[*Fo*; $\mathbf{P}_1 > \ldots > \mathbf{P}_k$; $\mathbf{Z}$], is used to assign priorities to minimization of the extensions of predicates in *Fo*. Minimization of the extensions of predicates in $\mathbf{P}_i$ is given priority over that of the extensions of predicates in $\mathbf{P}_j$ whenever $i < j$. The extensions of predicates can be partitioned based on some constraints on the terms. In intra-predicate prioritized circumscription, denoted by ICIRC[*Fo*; P; k; $\mathbf{G}_1 > \ldots > \mathbf{G}_k$; $\mathbf{Z}$], the extension of P is separated based on the k-th term, specifically based on the functions that produce the k-th term. Intra-predicate prioritized circumscription is used to assign priorities to minimization of (disjoint) subsets of the extensions of P. Minimization of the subset of extensions of P where k-th term is generated by functions in $\mathbf{G}_i$ is given priority over the subset where k-th term is generated by functions in $\mathbf{G}_j$ whenever $i < j$.

Propositions 3.4.10 and 3.4.11 restate Propositions 3.4.7 and 3.4.8, respectively, for stratification by function symbols instead of predicate symbols. The former results follow analogously by the reasoning for the latter, after substituting inter-predicate analysis with intra-predicate analysis.

Proposition 3.4.10. $CIRC^A[F; P] \equiv ICIRC[F; P; k; \mathbf{G_0} > \ldots > \mathbf{G_k}]$.

Proposition 3.4.11. $ICIRC[F; P; k; \mathbf{G_0} > \ldots > \mathbf{G_k}] \equiv$

$$\bigwedge_{G \in \mathbf{G_0}} \forall \mathbf{w}\mathbf{x}\mathbf{y}.\neg P(\mathbf{w}, G(\mathbf{x}), \mathbf{y}) \wedge F_0 \wedge \bigwedge_{j=1}^{n} Comp(F_j).$$

So, CIRC^A transformation of positive hierarchical formulas, with predicate-based or function-based stratification, to minimize the extensions of aggregate predicates, can be interpreted as some prioritized circumscription, which in turn is equivalent to the predicate completion.

3.5 $CIRC^A$ Transformation and Stable Model Semantics

The CIRC^A transformation and first-order stable model semantics do not generally coincide. Consider Fo_2 (Section 3.1) and an equivalent reformulation, Fo_{14}.

$$(\neg\#sum\langle x.P(x)\rangle < 1 \longrightarrow P(1)) \wedge (\neg\#sum\langle x.P(x)\rangle >= 1 \longrightarrow Q(1)) \qquad (\text{Fo}_{14})$$

$SM_{P,Q}[Fo_2] = CIRC^A[Fo_2; P, Q] = CIRC^A[Fo_{14}; P, Q]$. Whereas $SM_{P,Q}[Fo_{14}] = Fo_{14} \wedge \neg\exists p, q[[p, q] < [P, Q] \wedge Fo_2^{FA}([p, q])]$. Recall that $Fo_2^{FA}([p, q])$ denotes the FA-transformation (Definition 3.1.1) of $Fo_2([p, q])$, which is also equivalent to $Fo_{14}^{FA}([p, q])$. So, {P(1)} is a stable model of Fo_{14}, but not a model of $CIRC^A[Fo_{14}; P, Q]$ or $SM_{P,Q}[Fo_2]$.

However, we can extend the canonical class of formulas (Section 2.3.4) to allow for aggregate expressions in the formulas such that the CIRC^A transformation and first-order stable model semantics coincide. The logical primitives in the sm-notation are different from the notation used in the main article (cf. Section 2). We will refer to the latter notation as the *circ-notation*.[8]

We will refer to $F^*(\mathbf{p})$ as the **-transformation* of F. Lemma 2 from (Lee & Palla, 2012b) (or Lemma 5 from (Ferraris, Lee, & Lifschitz, 2011)), given for logic without aggregates, is valid even for *-transformation of formulas with aggregate expressions.

Lemma 3.5.1. *Formula,* $\mathbf{p} \leq \mathbf{P} \longrightarrow (F^*(\mathbf{p}) \longrightarrow F)$*, is logically valid.*

Proof. By induction on F. □

Lemma 3.5.2 defines the **-transformation (Definition 3.1.2) under the sm-notation, without resorting to NNF.[9]

[8] We have been, and will be, using the variable naming convention of circ-notation even within the sm-notation.

[9] The **-transformation in the circ-notation can similarly be defined without resorting to NNF

Lemma 3.5.2. *Let* $\mathbf{P}$ *be a tuple of predicate constants in* F*,* $\mathbf{Z}$ *be a tuple of constants in* F*, and* $\mathbf{p}$*,* $\mathbf{z}$ *are variables corresponding to constants* $\mathbf{P}$*,* $\mathbf{Z}$*. Then* $F^{**}(\mathbf{p},\mathbf{z}) = G^{**}(\mathbf{p},\mathbf{z})$*, where* $G^{**}(\mathbf{p},\mathbf{z})$ *is defined recursively:*

- *If* $G = (OP\langle \mathbf{x}.G'\rangle \succeq t)$ *has depth zero and contains a constant from* $\mathbf{P}$*, then* G^{**} *equals*
  - $(OP\langle \mathbf{x}.G'(\mathbf{p},\mathbf{z})\rangle \succeq t) \vee (OP\langle \mathbf{x}.G'\rangle \succeq t)$ *if* G *is in the antecedent of even number of implications in* F*;*
  - $(OP\langle \mathbf{x}.G'(\mathbf{p},\mathbf{z})\rangle \succeq t) \wedge (OP\langle \mathbf{x}.G'\rangle \succeq t)$ *otherwise,*
- *If* G *is not contained in any aggregate expression that contains a constant from* $\mathbf{P}$ *and* G *contains no aggregate expression that contains a constant from* $\mathbf{P}$ *then* $G^{**} = G(\mathbf{p},\mathbf{z})$*,*
- *If* G *is not contained in any aggregate expression that contains a constant from* $\mathbf{P}$ *but* G *contains an aggregate expression that contains a constant from* $\mathbf{P}$
  - $(G \wedge H)^{**} = G^{**} \wedge H^{**}$*;*
  - $(G \vee H)^{**} = G^{**} \vee H^{**}$*;*
  - $(G \longrightarrow H)^{**} = (G^{**} \longrightarrow H^{**})$*;*
  - $(\forall \mathbf{x} G)^{**} = \forall \mathbf{x} G^{**}$*;*
  - $(\exists \mathbf{x} G)^{**} = \exists \mathbf{x} G^{**}$*.*

Proof sketch. Let $F^{**r}(\mathbf{p},\mathbf{z})$ denote the **-transformation of F constructed according to the recursive definition given above. Let F^{**} denote the **-transformation of F by the original definition (Definition 3.1.2). For any aggregate formula of depth zero, F', replace its **-transformation in $F^{**r}(\mathbf{p},\mathbf{z})$ by unique G', where G' acts as a place holder for the **-transformation of F'. Consider NNF of $F^{**r}(\mathbf{p},\mathbf{z})$ post substitutions. The $F^{**r}(\mathbf{p},\mathbf{z})$ in NNF and F^{**} are equivalent. □

In the remainder, we will use the **-transformation in the sm-notation. Next we extend the syntactic class of canonical formulas to formulas that may contain

by equivalently rewriting the original formula to eliminate all $\longrightarrow$ and $\longleftarrow$. The transformation for a given aggregate formula can then be decided based on the number of negations containing the aggregate formula.

aggregate formulas, called the *ag-canonical of type I restrictions* formulas.[10] It is based on *simple antecedent occurrences* of aggregate expressions.

Definition 3.5.1. An aggregate expression in a formula F is said to have a *simple antecedent occurrence* if:

- it is in the antecedent of exactly one implication in F, and
- it does not occur inside another aggregate expression (that is, its depth relative to F is zero).

Definition 3.5.2. A formula F is *ag-canonical of type I restrictions* relative to a list $\mathbf{P}$ of predicate constants if:

- no occurrence of a predicate constant from $\mathbf{P}$ is in the antecedents of more than one implications in F,
- every occurrence of a predicate constant from $\mathbf{P}$ that is in the scope of a strictly positive occurrence of $\exists$ or $\vee$ in F is strictly positive in F, and
- every occurrence of aggregate formulas containing a predicate constant from $\mathbf{P}$ is a simple antecedent occurrence.

For example, if $Con_P(D)$ contains no aggregate formulas and $Con_P(D)$ is canonical with respect to $PartValue$ then $T_{ESI}[Con_P(D)]$ is ag-canonical with respect to $PartValue$.

By Lemma 3.5.2, if an aggregate formula, $OP\langle \mathbf{x}.\ F'(\mathbf{P},\mathbf{Z})\rangle \succeq t$, has a simple antecedent occurrence, then $(OP\langle \mathbf{x}.\ F'(\mathbf{P},\mathbf{Z})\rangle \succeq t)^{**}$ equals $(OP\langle \mathbf{x}.F'(\mathbf{p},\mathbf{z})\rangle \succeq t) \wedge (OP\langle \mathbf{x}.F'\rangle \succeq t)$.

Proposition 3.5.5 states that the semantics of the CIRC^A and SM transformations coincide for ag-canonical formulas of type I restrictions (henceforth, just ag-canonical), because the **-transformations and *-transformations of ag-canonical formulas, constructed with respect to the models of the formulas, are equivalent (Lemma 3.5.4). The **-transformation, with the **-transformation for aggregate formulas fixed according to their appearance in ag-canonical formulas, and *-transformation of formulas with only strictly positive occurrences of constants from $\mathbf{P}$ are equivalent (Lemma 3.5.3).

[10] We demonstrate here a simple extension which is probably not the largest syntactic class of such type.

Lemma 3.5.3. *If every occurrence of every predicate constant from* $\mathbf{P}$ *is strictly positive in* $\mathbf{F}$ *and for every aggregate formula* $G = OP\langle \mathbf{x}.\ F'\rangle \succeq t$ *in* $\mathbf{F}$*, the depth of* G *is zero, occurrence of* G *is strictly positive in* F*, and* $G^{**}(\mathbf{p})$ *is assumed to be* $(OP\langle \mathbf{x}.F'(\mathbf{p})\rangle \succeq t) \wedge (OP\langle \mathbf{x}.F'\rangle \succeq t)$*, then* $(\mathbf{p} \leq \mathbf{P}) \longrightarrow (F^*(\mathbf{p}) \longleftrightarrow F^{**}(\mathbf{p}))$ *is logically valid.*

Proof. Proof by induction, analogous to the proof for Lemma 3 from (Lee & Palla, 2012b).

- Let $F = OP\langle \mathbf{x}.\ F'(\mathbf{P}, \mathbf{Z})\rangle \succeq t$. Then,
  $F^{**}(\mathbf{p}) = (OP\langle \mathbf{x}.F'(\mathbf{p})\rangle \succeq t) \wedge (OP\langle \mathbf{x}.F'\rangle \succeq t)$, and
  $F^{*}(\mathbf{p}) = (OP\langle \mathbf{x}.F'^{*}(\mathbf{p})\rangle \succeq t) \wedge (OP\langle \mathbf{x}.F'\rangle \succeq t)$.

Since every occurrence of every predicate from $\mathbf{P}$ is strictly positive in F' and F' cannot contain any aggregate formula, by Lemma 3 from (Lee & Palla, 2012b), $F'^{*}(\mathbf{p}) \longleftrightarrow F'(\mathbf{p})$.

- Among the formulas not contained inside any aggregate expression, we will show only the case when F is $G \longrightarrow H$. The other cases are straightforward.
  $F^{**}(\mathbf{p}) = G^{**}(\mathbf{p}) \longrightarrow H^{**}(\mathbf{p})$, and
  $F^{*}(\mathbf{p}) = (G^{*}(\mathbf{p}) \longrightarrow H^{*}(\mathbf{p})) \wedge (G \longrightarrow H)$.

Since every occurrence of predicate constants from $\mathbf{P}$ is strictly positive in F, G contains no predicate constants from $\mathbf{P}$, so that $G^{**}(\mathbf{p})$ and $G^{*}(\mathbf{p})$ are equivalent to G. Also, $H^{**}(\mathbf{p}) \longleftrightarrow H^{*}(\mathbf{p})$ by Induction Hypothesis (I.H.). Therefore, it is sufficient to prove that under the assumption $\mathbf{p} \leq \mathbf{P}$, $(G \longrightarrow H^{*}(\mathbf{p})) \wedge (G \longrightarrow H) \longleftrightarrow (G \longrightarrow H^{*}(\mathbf{p}))$, is valid. From left to right is clear. Assume, $\mathbf{p} \leq \mathbf{P}$, $G \longrightarrow H^{*}(\mathbf{p})$, and G. We get $H^{*}(\mathbf{p})$. By Lemma 3.5.1, we conclude H. □

Lemma 3.5.4. *If* F *is ag-canonical relative to* $\mathbf{P}$*, then formula,* $(\mathbf{p} \leq \mathbf{P}) \wedge F \longrightarrow (F^{*}(\mathbf{p}) \longleftrightarrow F^{**}(\mathbf{p}))$*, is logically valid.*

Proof. Proof by induction, analogous to the proof for Lemma 4 from (Lee & Palla, 2012b).

- F is an atomic formula. Trivial.
- F is an aggregate formula, $(OP\langle \mathbf{x}.F'\rangle \succeq t)$. Then,

$F^*(\mathbf{p}) = (OP\langle \mathbf{x}.F'^*(\mathbf{p})\rangle \succeq t) \wedge (OP\langle \mathbf{x}.F'\rangle \succeq t)$, and
$F^{**}(\mathbf{p}) = (OP\langle \mathbf{x}.F'(\mathbf{p})\rangle \succeq t) \wedge (OP\langle \mathbf{x}.F'\rangle) \succeq t)$.

Since F' cannot contain any aggregate expression and every occurrence of every predicate constant from $\mathbf{P}$ is strictly positive in F', by Lemma 3 from (Lee & Palla, 2012b), $F'^*(\mathbf{p}) \longleftrightarrow F'(\mathbf{p})$.

- $F = G \wedge H$. Follows from I.H.
- $F = G \vee H$. Assume $(\mathbf{p} \leq \mathbf{P}) \wedge (G \vee H)$. Since $G \vee H$ is ag-canonical relative to $\mathbf{P}$, every occurrence of every predicate constant from $\mathbf{P}$ is strictly positive in G or in H so that, G and H cannot contain any aggregate formula (containing a constant from $\mathbf{P}$), and by Lemma 3 from (Lee & Palla, 2012b) (or Lemma 3.5.3), $G^*(\mathbf{p}) \longleftrightarrow G(\mathbf{p})$ and $H^*(\mathbf{p}) \longleftrightarrow H(\mathbf{p})$. Since G and H contain no aggregate formulas $G^{**}(\mathbf{p}) \longleftrightarrow G(\mathbf{p})$ and $H^{**}(\mathbf{p}) \longleftrightarrow H(\mathbf{p})$.
- $F = G \longrightarrow H$

  – Case 1. H has a strictly positive occurrence of a predicate constant from $\mathbf{P}$ inside an aggregate expression. Then, F must be in the antecedent of another implication. Thus, no predicate constant from $\mathbf{P}$ would occur in G. Therefore, $G^{**}(\mathbf{p})$ and $G^*(\mathbf{p})$ are equivalent to G.

  – Case 2. G has a positive occurrence of a predicate constant from $\mathbf{P}$ inside an aggregate expression. By Lemma 3.5.3, $G^*(\mathbf{p}) \longleftrightarrow G^{**}(\mathbf{p})$.

  – Case 3. F contains no aggregate expression. Then, $G^{**}(\mathbf{p}) \longleftrightarrow G(\mathbf{p})$. Since every occurrence of every predicate constant from $\mathbf{P}$ in G is strictly positive in G, so that, by Lemma 3 from (Lee & Palla, 2012b), $G^*(\mathbf{p}) \longleftrightarrow G(\mathbf{p})$.

Therefore, in all the three possible cases, $G^{**}(\mathbf{p}) \longleftrightarrow G^*(\mathbf{p})$. Now assume $(\mathbf{p} \leq \mathbf{P}) \wedge (G \longrightarrow H)$. It is sufficient to show, $(G^*(\mathbf{p}) \longrightarrow H^*(\mathbf{p})) \longleftrightarrow (G^{**}(\mathbf{p}) \longrightarrow H^{**}(\mathbf{p}))$.

  – Case 1: $\neg G$. By Lemma 3.5.1, $\neg G^*(\mathbf{p})$. The claim follows since $\neg G^*(\mathbf{p}) \longleftrightarrow \neg G^{**}(\mathbf{p})$.

  – Case 2: H. By I.H. $H^*(\mathbf{p}) \longleftrightarrow H^{**}(\mathbf{p})$. The claim follows since $G^*(\mathbf{p}) \longleftrightarrow G^{**}(\mathbf{p})$.

- $F = \forall x G$. Follows from I.H.

- $F = \exists x\, G$. Since every occurrence of every predicate constant form $\mathbf{P}$ in G is strictly positive in G, G contains no aggregate formulas (containing a constant from $\mathbf{P}$) and the claim follows from Lemma 3 from (Lee & Palla, 2012b) (or Lemma 3.5.3). □

Proposition 3.5.5. *For any ag-canonical formula F relative to $\mathbf{P}$, $CIRC^A[F;\mathbf{P}] \longleftrightarrow SM[F;\mathbf{P}]$, is logically valid.*

Proof. Follows from Lemma 3.5.4. □

3.6 Summary

In summary, we formally defined weak models with regards to interpretation of some aggregate formulas as constraints in Section 3.1 where the most intuitive idea was searching for a smaller interpretation that is a model of the fixed-aggregate-interpretation transformation of the formula, rather than model of the original formula as in the circumscription. We defined the alternative circumscription-like transformation, CIRC^A, to eliminate the weak models in Section 3.2. In Section 3.3, we discussed how a value different from zero for an aggregate summation when the aggregate expression is not satisfied may be accommodated. Since $\exists \mathbf{x}.F(\mathbf{x})$ cannot be interpreted as a constraint we use $\#count\langle \mathbf{x}.F(\mathbf{x}) \neq 0$ to verify if the formula F is satisfied. In Section 3.4, we showed equivalent first-order forms for CIRC^A transformations of formulas with stratified aggregates. Particularly, we showed that the CIRC^A transformation of positive hierarchical formulas, with predicate-based or function-based stratification, to minimize the extensions of aggregate predicates, can be interpreted as some prioritized circumscription, which in turn is equivalent to the predicate completion. Finally, in Section 3.5, we compared the $CIRC^A$ transformation and first-order stable model semantics where we defined a class of *ag-canonical* formulas for which the two semantics coincide.

CHAPTER 4
EXTENDING EVENT CALCULUS FOR EFFICIENT DESCRIPTIONS OF ADDITIVE EFFECTS

Some simple examples of additive effects, which we describe herein, include multiple forces applied on a block, water flowing out from multiple outlets of a tank, and multiple concurrent scoops from a water tank. Their descriptions in the current formalisms cannot be general, tend to be long, and are not additive-elaboration tolerant. For example, an axiomatic description of two pipes pouring water into a tank cannot be used when there are three pipes pouring water into a tank (hence, scenario-specific). Axiomatic descriptions of two pipes and three pipes pouring water into a tank have to be considered separately, and descriptions of conditions under which a given combined effect is valid tend to be verbose (hence, long). Besides, a new additive effect on the rate of change of the amount of water, water flowing out from an outlet for example, renders the earlier description of the rate of change of the amount of water (sum of the rates at which pipes poured water into the tank) incomplete and meaningless (hence, not allowing additive-elaboration). Recursive or iterative summation of additive effects is not a good choice either because it requires artificial ordering of additive effects, which is not natural from a knowledge representation perspective. These limitations are discussed in detail in Section 4.1.

Formally, those effects which affect the same real-valued quantity and must be summed together to determine the combined effect on the quantity when active simultaneously are called *additive effects*. Effects on the same real-valued quantity that are not additive are referred to as *non-additive effects*. Consider, for example, a tank with multiple outlets which are all covered initially (Figure 4.1). The effect of uncovering one outlet can be described as a function of time, $h = H_0 \times exp(-K \times (t - T_0))$, where H_0 is the height of water in the tank when cover is taken off at time T_0, h is the height at any time t, $t > T_0$, and K is a constant. Alternatively, the

Portions of this chapter to appear in: Khandelwal, A., & Fox, P. (2013). General Descriptions of Additive Effects via Aggregates in the Circumscriptive Event Calculus. *Journal of Artificial Intelligence Research* (JAIR).

effect can be described through the rate of change of height, $d(h)/dt = -K \times h$. If a second outlet is opened simultaneously, the net effect can be computed by summing the effects in the latter case, that is $d(h)/dt = -2 \times K \times h$, which is not true for the former description as the net effect is $h = H_0 \times exp(-2 \times K \times (t - T_0))$. In this example, the former effects are non-additive whereas the latter are additive. In a related example, the effects of two scoops from a tank of water are additive when the total amount scooped is less than the amount of water in the tank, otherwise they are non-additive.

As implied by the tank with multiple outlets example above, continuous concurrent effects must typically be described by way of rate of change in the value of some quantity to be additive. The rate of change may not be a constant, and the capability to reason about differential equations is required to reason about such additive effects.

In the Event Calculus, time-varying properties are differentiated into fluents and quantities (or parameters). *Fluents* are boolean-valued whereas *quantities* take a range of values at any given time. Fluents and quantities correspond respectively with *relational fluents* and *functional fluents* in Situation Calculus and other formalisms for example. The value of a quantity P at a time T is accessed via the $Value$ function, as $Value(P, T)$. Quantitative description of a continuous-change is defined by giving the value of a quantity as a function of time, as $Value(P, T) = F(T)$, where P may be a derivative of other quantity and $F(T)$ may be a function of values of other quantities. In order to specify additive continuous-changes we introduce a new relation $PartValue$. $PartValue(P, T, PX, V)$ denotes that quantity P has a partial, additive (henceforth just partial or additive) value of V (uniquely identified by PX) at time T. The quantities of a domain with additive values described using $PartValue$ relation are referred to as *additive quantities* (or *additive parameters*), in the spirit of additive fluents (Lee & Lifschitz, 2003).

The Event Calculus is designed such that users define axioms to specify necessary conditions for action occurrences, their effects on fluents or quantities, etc., while sufficiency conditions are inferred via circumscription (McCarthy, 1980; Lifschitz, 1994) of the axioms. That is, the solution to the frame problem is given

through nonmonotonic reasoning via circumscription. Circumscription was defined for first-order logic without aggregates, and to our knowledge, has never been applied to first-order logic with aggregates. We observe that the circumscription transformation extended as is to first-order logic with aggregate formulas is insufficient to model the sufficient conditions for active continuous additive changes. We discuss two other approaches for describing continuous additive changes, using $PartContributor$ and $PartContributorWFluent$ relations, but circumscription is insufficient to model the required nonmonotonic reasoning. $PartContributor(P_1, T, P_2)$ denotes that quantity P_1 has an additive value of $Value(P_2, T)$ at time T. $PartContributorWFluent(P_1, T, F, P_2)$ denotes that when fluent F holds at time T then quantity P_1 has an additive value of $Value(P_2, T)$ at time T. The latter approach is an attempt towards benefitting from nonmonotonic reasoning for change in the states of fluents. $PartValue$ and $PartContributor$ are similar, but the former has a more general form.

The $CIRC^A$ transformation selects those models of circumscribed formulas that do not have unexpected facts that result from aggregations in the formulas. The default reasoning for active continuous additive changes is defined via the $CIRC^A$ transformation.

The extension for descriptions of discrete additive changes to the values of quantities follows along the same lines. Discrete changes are described using $BreaksTo$ relations in the Event Calculus. $BreaksTo(A, P, T, V)$ denotes that when action A occurs at time T, it changes the value of quantity P to V. We introduce a relation $BreaksPartBy$. $BreaksPartBy(A, P, T, V)$ denotes that when action A occurs at time T, it additively changes the value of quantity P by V. Circumscription is sufficient for nonmonotonic reasoning about discrete additive effects without ramifications, $CIRC^A$ transformation must be used otherwise.

The previous proposals for distinct descriptions of additive changes, such as by Lee and Lifschitz (2003), address only discrete effects of actions. Besides, they resort to syntactic constructs, whereas our method involves logical extensions.

In Section 4.1, using an example of a tank filled by multiple moving pipes, we discuss the limitations of explicit enumeration of combined effects of additive

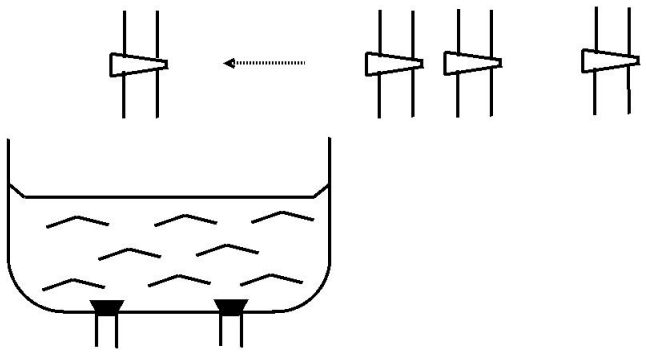

Figure 4.1: A water tank with two outlets at the bottom, filled by four pipes with taps rotating above. The pipes are at random distances from one another, and the arrow points in the direction of their rotation.

effects and of using recursion or iteration for computing the combined effects. We introduce the extensions to the Event Calculus in Section 4, and illustrate them using the example from the previous section. We discuss the limitations of using circumscription for nonmonotonic reasoning over continuous additive effects using another example involving a single block subject to multiple horizontal forces in Section 4.3. In Section 4.4 we discuss descriptions of ramifications of discrete additive effects in the extended Event Calculus. We complete and generalize the previous single block example to multiple blocks subject to multiple horizontal and vertical forces, to demonstrate the facility of the extended Event Calculus, in Section 4.5. In Section 4.6, we define a default reasoning strategy for the extended Event Calculus, discuss its correctness, and define the conditions under which the relevant CIRC^A transformations are equivalent to the completion.

4.1 Example I: The Water Tank and Pipes Problem

We demonstrate inefficiencies due to explicit enumeration of combined effects of continuous additive effects with help of the water tank and pipes example. In Section 4.1.1, we use recursion/iteration to compute the combined effects by assuming an ordering of the additive effects, to overcome some of those inefficiencies. In Section 4.1.2, we demonstrate similar issues with descriptions of additive discrete effects. The water tanks and pipes problem described above is a more general form

of the Many Tanks Problem (Shanahan, 1990) and the Water Tank Example (Miller & Shanahan, 1996).

Table 4.1: Predicates, quantities, fluents, and actions used in the water tank and pipes problem.

Term	Type	Description
$Outlet(o, v)$	Predicate	o is an outlet of tank v
$Pipe(n)$	Predicate	n is a pipe
$Conn(n, v)$	Fluent	Pipe n is above and connected to tank v
$Open(o)$	Fluent	Tap of outlet o is open
$TapOn(n)$	Fluent	Tap of pipe n is on
$Contr(n, v)$	Quantity	the rate at which tank n contributes water to tank v
$Contr(o)$	Quantity	the rate at which outlet o releases water
$Level(v)$	Quantity	the level of water in tank v
$Rate(n)$	Quantity	the rate at which water is released from pipe n if open
$Rate(o)$	Quantity	the rate at which water is released from outlet o if open
$Scoop(s, v)$	Action	s-th act of scooping water from tank v

A tank with multiple outlets at the bottom is filled by multiple pipes that are arbitrarily spaced. Figure 4.1 depicts one such scenario with four pipes and two outlets. The pipes are not fixed (the pipes could be rotating for example), and they move in and out of the space above the tank. The goal is to determine the rate at which the tank is filling at a given time, i.e. $Value(\delta(Level(v), t)$ for tank v at time t. The quantities, fluents, and predicate constants used for the descriptions of the domain are listed in Table 4.1. Here, we only consider axioms directly relevant to the computation of the rate of filling, not to breaks in the continuities of quantities ($Contr(n, v)$, $Contr(o)$, $\delta(Level(v))$) or changes in the states of the fluents ($Conn(n, v)$, $Open(o)$, $TapOn(n)$) for example. We assume the following unique name assumptions:

$$UNA[Conn, Open, TapOn] \wedge UNA[Rate, Contr, Level, \delta] \wedge UNA[Scoop].$$

Axioms (A1) and (A2) specify the amounts of water contributed by a pipe to a tank when the tap is on and off respectively.

$$[HoldsAt(Conn(n, v), t) \wedge HoldsAt(TapOn(n), t)] \longrightarrow \qquad \text{(A1)}$$

$$[HoldsAt(Conn(n,v),t) \wedge \neg HoldsAt(TapOn(n),t)] \longrightarrow \begin{array}{c} Value(Contr(n,v),t) = Value(Rate(n),t) \\ \\ Value(Contr(n,v),t) = 0 \end{array} \quad \text{(A2)}$$

Initially let us assume that all the outlets are covered permanently. Now if we assume that at any time-point exactly one pipe is connected to the tank, then the rate of filling of a tank could be described by Axiom (A3).

$$HoldsAt(Conn(n,v),t) \longrightarrow Value(\delta(Level(v)),t) = Value(Contr(n,v),t) \quad \text{(A3)}$$

Instead, if at any time-point there could be either one or two pipes connected to the tank, then the rate of filling of a tank is described by Axioms (A4) and (A5).

$$[HoldsAt(Conn(n_1,v),t) \wedge \neg\exists n_2(HoldsAt(Conn(n_2,v),t) \wedge n_2 \neq n_1)] \longrightarrow \quad \text{(A4)}$$
$$Value(\delta(Level(v)),t) = Value(Contr(n_1,v),t)$$
$$[HoldsAt(Conn(n_1,v),t) \wedge HoldsAt(Conn(n_2,v),t) \wedge n_2 \neq n_1] \longrightarrow \quad \text{(A5)}$$
$$Value(\delta(Level(v)),t) = Value(Contr(n_1,v),t) + Value(Contr(n_2,v),t)$$

But, if there could be between 1 and $N+1$ pipes, $N > 0$, then N more axioms of the form (A4) or (A5) would be required.

Further, if the outlets are not covered permanently, the axioms describing the rate of filling have to be changed to accommodate the new contributions. Assuming that the tank has only one outlet and at any time-point there is exactly one pipe that is connected to the tank, Axiom (A6) replaces Axiom (A3).

$$[HoldsAt(Conn(n,v),t) \wedge Outlet(o,v)] \longrightarrow \quad \text{(A6)}$$
$$Value(\delta(Level(v)),t) = Value(Contr(n,v),t) - Value(Contr(o),t)$$

The contribution from an outlet is zero or non-zero depending on whether it is

covered or not.

$$[Outlet(o, v) \wedge HoldsAt(Open(o), t)] \longrightarrow Value(Contr(o), t) = Value(Rate(o), t) \tag{A7}$$

$$[Outlet(o, v) \wedge \neg HoldsAt(Open(o), t)] \longrightarrow Value(Contr(o), t) = 0 \tag{A8}$$

In general, explicit enumeration of combined effects of concurrently active additive effects has the following limitations. (i) Descriptions become very scenario-specific. For example, different axioms were required in scenarios when a single pipe fills a tank and when two pipes fill a tank. The description for two-pipes scenario cannot be reused for a very similar three-pipes scenario. (ii) Descriptions are long. For example, when the number of pipes filling a tank at any given time can be one or two, each case has to be considered separately. Also, the conditions under which a combined effect is applicable, for example the antecedents of Axioms (A4) and (A5), are verbose. (iii) Elaborations are not additive. For example, inclusion of a new contributor to the rate of filling, outflow from a water outlet, rendered the previous combined effect, the total inflow from the pipes, incomplete. And, Axiom (A3) had to be replaced by Axiom (A6). (Additive elaboration can be achieved via abnormality predicates (Lin, 2008; Mueller, 2008), but less gracefully.)

4.1.1 Alternative Formulation using Recursion

We can use recursion/iteration instead to sum the additive effects. But for that, we have to establish some total ordering between the effects that are added. For example, the combined effects axiomatized in Axioms (A3)–(A6) can be formulated as shown below, where $<$ in Axioms (A9)–(A11) denotes an ordering relation (more like a successor relation). $InterValue(P, X, T)$ denotes an intermediate (or partial) value of parameter P at time T, where X is one of the ordered terms.

$$\begin{aligned}[HoldsAt(Conn(n_2, v), t) \wedge HoldsAt(Conn(n_1, v), t) \wedge n_1 < n_2] \\ \longrightarrow InterValue(\delta(Level(v)), n_1, t) = InterValue(\delta(Level(v)), n_2, t) \\ + Value(Contr(n_1, v), t)\end{aligned} \tag{A9}$$

$$[HoldsAt(Conn(n_2, v), t) \wedge \neg\exists n_1(HoldsAt(Conn(n_1, v), t) \wedge n_1 < n_2) \quad \text{(A10)}$$
$$\wedge \neg\exists o.(Outlet(o, v))] \longrightarrow Value(\delta(Level(v)), t) = InterValue(\delta(Level(v)), n_2, t)$$
$$[HoldsAt(Conn(n_2, v), t) \wedge \neg\exists n_1(HoldsAt(Conn(n_1, v), t) \wedge n_1 < n_2)$$
$$\wedge Outlet(o_1, v)] \longrightarrow Value(\delta(Level(v)), t) \quad \text{(A11)}$$
$$= InterValue(\delta(Level(v)), n_2, t) - Value(Contr(o_1), t)$$

The ordering relation $<$ for this example can be specified without qualifications if the pipes are all known and they move in and out of the space above the tank in a fixed order. However, in general, ordering relations may have to be qualified and inferred, and the extension may have to be minimized as well. The descriptions of additive effects can be expressed in generality and in an elaboration tolerant manner using recursion, provided the required ordering relations can be specified. However, as argued by Dell'Armi et al. (2003) (in the context of knowledge representation in Answer Set Programming), this approach is bad from a knowledge representation perspective as the encoding is not natural (because additive effects have to be ordered artificially).

4.1.2 Discrete Additive Effects

Axiom (A12) states that the level of water in a tank is always greater than or equal to zero. Axioms (A13)–(A14) state that an isolated scooping from the tank instantaneously decreases the level of water by an amount R unless it becomes less than zero.

$$Value(Level(v), t) >= 0 \quad \text{(A12)}$$
$$BreaksTo(Scoop(s, v), Level(v), t, Value(Level(v), t) - R) \longleftarrow \quad \text{(A13)}$$
$$Value(Level(v), t) \geqslant R$$
$$BreaksTo(Scoop(s, v), Level(v), t, 0) \longleftarrow Value(Level(v), t) < R \quad \text{(A14)}$$

When up to two scoops can occur concurrently, their effects can be described using *Happens* preconditions (Miller & Shanahan, 2002) as shown in Axioms (A15)–

(A18).

$$[Happens(Scoop(s2, v), t) \wedge s1 \neq s2 \wedge Value(Level(v), t) \geqslant 2 \times R] \tag{A15}$$
$$\longrightarrow BreaksTo(Scoop(s1, v), Level(v), t, Value(Level(v), t) - 2 \times R)$$
$$[Happens(Scoop(s2, v), t) \wedge s1 \neq s2 \wedge Value(Level(v), t) < 2 \times R] \tag{A16}$$
$$\longrightarrow BreaksTo(Scoop(s1, v), Level(v), t, 0)$$
$$[\neg\exists s2(\neg Happens(Scoop(s2, v), t) \wedge s1 \neq s2) \wedge Value(Level(v), t) \geqslant R] \tag{A17}$$
$$\longrightarrow BreaksTo(Scoop(s1, v), Level(v), t, Value(Level(v), t) - R)$$
$$[\neg\exists s2(\neg Happens(Scoop(s2, v), t) \wedge s1 \neq s2) \wedge Value(Level(v), t) < R] \tag{A18}$$
$$\longrightarrow BreaksTo(Scoop(s1, v), Level(v), t, 0)$$

4.2 New Relations for the Descriptions of Additive Effects

Intuitively, the limitations associated with explicit enumeration of combined effects, as detailed in Section 4.1, can be overcome with help of aggregate formulas. Here, we formally introduce new predicates for describing additive changes and define their semantics using aggregate formulas. We will see later in Section 4.3 that nonmonotonic reasoning using circumscription for determining the additive changes active at any given time is inadequate in the case of continuous additive changes. We discuss three different extensions in the continuous case in Section 4.2.1, and circumscription is inadequate for all three of them. We discuss the extensions for descriptions of discrete additive effects in Section 4.2.2. Finally, we discuss required changes and extensions to the CIRC$_{CEC}$ circumscription policy to accommodate axioms about additive effects in Section 4.2.3.

4.2.1 Descriptions of Continuous Additive Changes

In the first approach (Section 4.2.1.1), additive changes are denoted via parameters that contribute additively to the change. In the second approach (Section 4.2.1.2) additive changes are still denoted via parameters that contribute additively to the change, but in addition coupled with a fluent so that the effect is active only if the fluent is active too. The last approach was an attempt to leverage non-

monotonic reasoning for change in the states of fluents for nonmonotonic reasoning over additive changes. The third approach (Section 4.2.1.3) is a more general form of the first approach where the values of additive changes are directly provided.

4.2.1.1 Approach I: Additive Contributors to the Change

$PartContributor(P_1, T, P_2)$ denotes that quantity P_1 has an additive value of $Value(P_2, T)$ at time T. Its semantics is given by Axiom (EC16p1).

$$[\exists p.PartContributor(p_1, t, p) \wedge s = \#sum\langle r, p_2.PartContributor(p_1, t, p_2) \wedge Value(p_2, t) = r\rangle] \longrightarrow Value(p_1, t) = s \quad \text{(EC16p1)}$$

An additive parameter p may not always be subject to additive effects and then, non-additive effects describe the $Value(p, t)$ directly. Thus, while sum of an empty multiset is defined, and equals zero, $Value(p, t)$ is not required to be zero when the parameter p is not subject to any additive effects.

We can reformulate Axioms (A1)–(A8) as shown below.

$$PartContributor(\delta(Level(v)), t, Contr(n, v)) \longleftarrow [HoldsAt(Conn(n, v), t) \wedge HoldsAt(TapOn(n), t)] \quad \text{(B1)}$$

$$Value(Contr(o), t) = -1 \times Value(Rate(o), t) \longleftarrow [Outlet(o, v) \wedge HoldsAt(Open(o), t)] \quad \text{(B2)}$$

$$PartContributor(\delta(Level(v)), t, Contr(o)) \longleftarrow [Outlet(o, v) \wedge HoldsAt(Open(o), t)] \quad \text{(B3)}$$

$$Value(\delta(Level(v)), t) = 0 \longleftarrow \neg\exists p.PartContributor(\delta(Level(v)), t, p) \quad \text{(B4)}$$

4.2.1.2 Approach II: Additive Contributors Coupled with Fluents

$PartContributorWFluent(P_1, T, F, P_2)$ denotes that when fluent F holds at time T then quantity P_1 has an additive value of $Value(P_2, T)$ at time T. Its semantics is given by Axiom (EC16p2). It bears a similarity to $Trajectory$ and $AntiTrajectory$ predicates (Miller & Shanahan, 2002) included in many versions of the Event Calculus formalism for specifying continuous changes mathematically

represented as functions of time.[11]

$$[\exists p, f'.[HoldsAt(f', t) \wedge PartContributorWFluent(p_1, t, f', p)] \wedge s = \#sum\langle r, p_2.HoldsAt(f, t) \wedge PartContributorWFluent(p_1, t, f, p_2) \wedge Value(p_2, t) = r\rangle] \longrightarrow Value(p_1, t) = s \quad \text{(EC16p2)}$$

We can reformulate Axioms (A1)–(A8) as shown below.

$$PartContributorWFluent(\delta(Level(v)), t, TapOn(n), Rate(n)) \longleftarrow [HoldsAt(Conn(n, v), t) \wedge HoldsAt(TapOn(n), t)] \quad \text{(B5)}$$

$$Value(Contr(o), t) = -1 \times Value(Rate(o), t) \longleftarrow [Outlet(o, v) \wedge HoldsAt(Open(o), t)] \quad \text{(B6)}$$

$$PartContributorWFluent(\delta(Level(v)), t, Open(o), Contr(o)) \quad \text{(B7)}$$

$$Value(\delta(Level(v)), t) = 0 \longleftarrow \neg\exists p, f.[HoldsAt(f, t) \wedge PartContributorWFluent(\delta(Level(v)), t, f, p)] \quad \text{(B8)}$$

4.2.1.3 Approach III: Additive Values of the Change

$PartValue(P, T, PX, R)$ denotes that quantity P has a partial, additive (henceforth just partial or additive) value of R uniquely identified by PX, at time T. This is a generalized form of $PartContributor$ predicate in that R can be value of a parameter as before, an argument of a parameterized fluent, a constant value, a function (for example, *cos*) of the above, or an algebraic combination of the above. An additional argument is required to differentiate between different additive contributions of the same value, as we discuss shortly. The additional argument PX is referred to as the *Argument for Disambiguating Contributions* or DCA. The semantics of $PartValue$ predicate is given by Axiom (EC16p).

$$[\exists r', px'.PartValue(p, t, px', r') \wedge s = \#sum\langle r, px.PartValue(p, t, px, r)\rangle] \longrightarrow Value(p, t) = s \quad \text{(EC16p)}$$

[11] Continuous changes represented via $(Anti)Trajectory$ can be represented via autonomous ODE in the version reviewed in Section 2.2.

We can reformulate Axioms (A1)–(A8) as shown below. DCAs have been underlined.

$$PartValue(\delta(Level(v)), t, \underline{n}, Value(Rate(n), t)) \longleftarrow \quad \text{(B9)}$$
$$[HoldsAt(Conn(n, v), t) \wedge HoldsAt(TapOn(n), t)]$$
$$PartValue(\delta(Level(v)), t, \underline{o}, -1 \times Value(Rate(o), t)) \longleftarrow \quad \text{(B10)}$$
$$[Outlet(o, v) \wedge HoldsAt(Open(o), t)]$$
$$Value(\delta(Level(v)), t) = 0 \longleftarrow \neg px, r.PartValue(\delta(Level(v)), t, px, r) \quad \text{(B11)}$$

Consider a variant of Axiom (B9), as in Axiom (B12), using a ternary form of $PartValue$ defined below (Axiom (B13)).

$$PartValue(\delta(Level(v)), t, Value(Rate(n), t)) \longleftarrow \quad \text{(B12)}$$
$$[HoldsAt(Conn(n, v), t) \wedge HoldsAt(TapOn(n), t)]$$
$$[\exists r'.PartValue(p, t, r') \wedge s = \#sum\langle r.PartValue(p, t, r)\rangle] \quad \text{(B13)}$$
$$\longrightarrow Value(p, t) = s$$

If at a given time T taps of exactly two pipes are on and their rates equal R, the $Value(\delta(Level(v)), T)$ would be deduced to be R instead of $2 \times R$. Such additive contributions can be uniquely identified through DCAs.

4.2.2 Descriptions of Discrete Additive Effects

$BreaksPartBy(A, P, T, R)$ expresses that if action A occurs at time T then it discontinuously changes the value of quantity P by R. Axiom (EC17p) constrains $RightLimit$ in terms of concurrent $BreaksPartBy$ and $Happens$.

$$[\exists a, r.(BreaksPartBy(a, p, t, r) \wedge Happens(a, t)) \wedge s =$$
$$Value(p, t) + \#sum\langle r, a.(BreaksPartBy(a, p, t, r) \wedge Happens(a, t))\rangle] \quad \text{(EC17p)}$$
$$\longrightarrow RightLimit(p, t, s)$$

Axiom (EC18p), similar to Axiom (EC11), relates $BreaksPartBy$ with $Breaks$.

$$BreaksPartBy(a, p, t, r) \longrightarrow Breaks(a, p, t) \tag{EC18p}$$

We can reformulate Axioms (A15)–(A18) as shown below. Axiom (B14) states that a unique scoop reduces the level by amount R.

$$BreaksPartBy(Scoop(s, v), Level(v), t, -1 \times R) \tag{B14}$$
$$\longleftarrow Value(Level(v), t) \geqslant R$$

However, Axiom (B14) does not model Axioms (A15)–(A18) precisely, as more than $\lceil Value(Level(v), t)/R \rceil$ concurrent scoops, when the concurrent scoops are non-additive, lead to an inconsistency (because then $Value(Level(v), t) < 0$).

It is relatively straightforward to introduce *range constraints* (Erdem & Gabaldon, 2005) to model non-additive discrete effects. For example, consider a variant of $BreaksPartBy$ relation. $BreaksPartMinBy(A, P, T, R, MINP)$ expresses that if action A occurs at time T then it discontinuously changes the value of quantity P by R unless due to other concurrent additive effects the value becomes less than $MINP$.

$$[\exists a, r.(BreaksPartMinBy(a, p, t, r, m) \wedge Happens(a, t)) \wedge s' =$$
$$Value(p, t) + \#sum\langle r, a.(BreaksPartMinBy(a, p, t, r, m) \wedge Happens(a, t))\rangle$$
$$\wedge ((s' >= m \wedge s = s') \vee (s' < m \wedge s = m))] \longrightarrow RightLimit(p, t, s)$$

$MINP$ is part of $BreaksPartMinBy$ relation here but other representations where $MINP$ is specified separately can be defined similarly.

Also, $BreaksPartBy$ relations define change relative to the current value. We can define a similar relation, $BreaksPartTo$, to express absolute changes (that is, changes relative to zero). $BreaksPartTo(A, P, T, R)$ expresses that if action A occurs at time T then it discontinuously changes the value of quantity P by R

relative to zero.

$$[\exists a, r.(BreaksPartTo(a, p, t, r) \wedge Happens(a, t)) \wedge s = \\ \#sum\langle r, a.(BreaksPartTo(a, p, t, r) \wedge Happens(a, t))\rangle] \longrightarrow RightLimit(p, t, s)$$

4.2.3 Extending CIRC$_{CEC}$ Policy

As before, let D denotes a collection of domain-specific axioms. And, let $Con_P(D)$ denotes the axioms about continuous additive changes (using either of $PartContributor$, $PartContributorWFluent$, or $PartValue$), and $Inst_P(D)$ denotes the axioms about discrete additive effects (using $BreaksPartBy$).

Changes to the $CIRC_{CEC}(D)$ policy (Section 2.2.1) with regards to $Inst_P(D)$ are relatively straightforward. Change described by $BreaksPartBy$ take effect in the immediate future, and so minimal models of $Inst_P(D)$, minimal with respect to the extent of $BreaksPartBy$, are the intended models. Assuming that antecedents of $Inst_P(D)$ refer to the present or past values of quantities, the models of $Inst_P(D)$ minimal with respect to the extent of $BreaksPartBy$ give the intended models. We can replace $CIRC[Inst(D) \wedge (EC11) \wedge (EC12); Breaks, BreaksTo]$ in $CIRC_{CEC}(D)$ by

$$CIRC[Inst(D) \wedge (EC11) \wedge (EC12) \wedge (EC18p); Breaks, BreaksTo] \\ \wedge CIRC[Inst_P(D); BreaksPartBy] \wedge (EC17p).$$

The above assumption does not apply to domains containing ramifications of discrete additive effects that depend on the amount of cumulative change by them. See Section 4.4 for a discussion.

Similar changes to $CIRC_{CEC}(D)$ with regards to $Con_P(D)$ would amount to replacing $Con(D)$ by,

$$CIRC[T_{ESI}[Con_P(D)]; PartValue] \wedge Con(D) \wedge (EC16p),$$

for Approach III, where T_{ESI} transformation defined below replaces $Value(p, t)$ in the antecedent by summation of additive values as given by Axiom (EC16p) for each

additive parameter p.

Definition 4.2.1. *Explicit Summation Inclusion Reformulation* (*ESI-reformulation*) of an axiom Ax in implicative form, denoted by $T_{ESI}[Ax]$, is defined as follows. Every unique $Value(p,t)$ reference for any additive parameter p is replaced by a fresh variable z, and the following condition is conjoined with the antecedent:

$$(\#count\langle r, px.PartValue(p,t,px,r)\rangle \neq 0 \wedge z = \#sum\langle r, px.PartValue(p,t,px,r)\rangle)$$
$$\vee\, (\#count\langle r, px.PartValue(p,t,px,r)\rangle = 0 \wedge z = Value(p,t)).$$

The value for variable z at any given time is given by one of the disjuncts, by the first disjunct when an additive parameter is subject to additive effects and the second disjunct when the parameter is subject to non-additive or no effects. The ESI-reformulation of Con_P(D) is given by, $T_{ESI}[Con_P(D)] = \bigwedge_{Ax \in Con_P(D)} T_{ESI}[Ax]$. $\#count\langle r, px.PartValue(p,t,px,r)\rangle \neq 0 \equiv \exists r, px.PartValue(p,t,px,r)$, but using the latter in the ESI-reformulation can yield unintended models, as we discuss in Section 4.6.1 (and Section 3.3). If in a domain description $Value(p,t)$ is uniformly zero for every additive parameter p whenever no known additive value exists then $\mathrm{T}_{ESI,0}$ transformation defined below can be used instead.

Notation 4.2.2. $T_{ESI,0}[Ax]$ denotes a special form of ESI-reformulation, of an axiom Ax, where every unique $Value(p,t)$ reference for any additive parameter p is replaced by a fresh variable z and the condition, $z = \#sum\langle r, px.PartValue(p,t,px, r)\rangle$, is conjoined with the antecedent.

The changes to CIRC_{CEC} policy for additive continuous changes described using the other two predicates are similar to that described above for Approach III. The circumscription policy, $[CIRC_{CEC}(D) \wedge CIRC[T_{ESI}[Con_P(D)]; PartValue] \wedge (EC16p)]$, where $T_{ESI,0}$ may be used in place of T_{ESI} if the required conditions are satisfied, is adequate for the tank and multiple pipes example because the additive values of quantities have no dependencies. As we see in the following section, the above circumscription policy yields anomalous/unexpected models when there is a (conditional) dependency between additive effects on quantities.

Figure 4.2: A block with multiple horizontal forces in either direction. The solid lines (with arrow-heads) denote external forces, and the dotted line denotes friction force with arrow drawn under the assumption that the block has a tendency to move to the right.

4.3 Example II: A Single Block Problem

As shown in Figure 4.2, there is a block lying on the ground which may be subject to multiple horizontal forces at any given time. The goal is to determine the net force on the block at any given time. Actions, quantities, and fluents used in the single block problem are described in Table 4.2.

Table 4.2: Actions, fluents, and quantities used in the single block problem.

Term	Type	Description
$ApplyHF(frc, b, r)$	Action	Apply horizontal force frc of value r on block b
$RemoveHF(frc, b, r)$	Action	Stop applied horizontal force frc of value r on block b
$F_{AHF}(frc, b, r)$	Fluent	Block b is subject to external horizontal force frc of value r
$F_{NAHF}(b)$	Fluent	Block b is subject to net external horizontal force
$AHF(frc, b)$	Quantity	Externally applied horizontal force frc on block b
$NAHF(b)$	Quantity	Net external horizontal force applied on block b
$Fr(b, Ground)$	Quantity	Friction force applied on block b by the $Ground$

We separately axiomatize the domain using the three approaches, and describe the models deduced by the circumscription policy described in Section 4.2.3. The models are produced for the following narrative:

$$Value(AHF(Frc, B), 0) = 0. \tag{C1}$$

$$Value(NAHF(B), 0) = 0. \tag{C2}$$

$$Value(NHF(B), 0) = 0. \tag{C3}$$

$$Value(Fr(B, Ground), 0) = 0. \tag{C4}$$

$$Happens(ApplyHF(Frc, B, 10), 1). \tag{C5}$$

$$InitializedFalse(F_{AHF}(Frc, B, 10)). \tag{C6}$$

$$InitializedFalse(F_{NAHF}(B)). \tag{C7}$$

We assume that all constants are unique:

$$UNA[ApplyHF, RemoveHF] \wedge UNA[AHF, NAHF, NHF, Fr, \delta] \tag{C8}$$
$$\wedge UNA[F_{AHF}, F_{NAHF}] \wedge UNA[Frc, B]$$

4.3.1 Using Approach I

The example domain can be described as shown below. The fluents, $F_{AHF}(frc, b, v)$ and $F_{NAHF}(b)$, are not required in this approach.

$$BreaksTo(ApplyHF(frc, b, v), AHF(frc, b), t, v) \tag{C9}$$

$$BreaksTo(RemoveHF(frc, b, v), AHF(frc, b), t, 0) \tag{C10}$$

$$[(a = ApplyHF(frc, b, v) \vee a = RemoveHF(frc, b, v)) \wedge \tag{C11}$$
$$(p = NAHF(b) \vee p = NHF(b))] \longrightarrow Breaks(a, p, t)$$

$$Value(AHF(frc, b), t) \neq 0 \longrightarrow PartContributor(NAHF(b), t, AHF(frc, b)) \tag{C12}$$

$$\exists frc.(Value(AHF(frc, b), t) \neq 0) \longrightarrow PartContributor(NHF(b), t, NAHF(b)) \tag{C13}$$

$$\exists frc.(Value(AHF(frc, b), t) \neq 0) \longrightarrow \tag{C14}$$
$$PartContributor(NHF(b), t, Fr(b, Ground))$$

$$[(p = NAHF(b) \vee p = NHF(b)) \wedge \neg\exists p'.PartContributor(p, t, p')] \longrightarrow \tag{C15}$$
$$Value(p, t) = 0$$

The circumscription policy for the above domain description is as follows:

$$CIRC_{CECP1}(D) = CIRC[(C1) \wedge \ldots \wedge (C5); Happens,$$

$$Initialized True, Initialized False]$$
$$\wedge CIRC[(C9) \wedge (C10) \wedge (C11) \wedge (EC11) \wedge (EC12); Breaks, BreaksTo]$$
$$\wedge CIRC[T_{ESI,0}[(C12) \wedge (C13) \wedge (C14)]; PartContributor]$$
$$\wedge (C15) \wedge (EC16p1) \wedge (C8) \wedge EC$$

$T_{ESI,0}[(C12) \wedge (C13) \wedge (C14)] = (C12) \wedge (C13) \wedge (C14)$, since the antecedents do not contain any additive parameter.

We enumerate only the facts about $Value$ and $PartContributor$ for models of $CIRC_{CECP1}(D)$. $CIRC_{CECP1}(D)$ has just one model with the following facts for time 1 (and times before 1):

$$Value(AHF(Frc, B), 1) = 0 \wedge Value(NAHF(B), 1) = 0$$
$$\wedge Value(NHF(B), 1) = 0 \wedge Value(Fr(B, Ground), 1) = 0,$$

and for time 2 (and times after 1):

$$Value(AHF(Frc, B), 2) = 10 \wedge PartContributor(NAHF(B), 2, AHF(Frc, B))$$
$$\wedge PartContributor(NHF(B), 2, NAHF(B))$$
$$\wedge PartContributor(NHF(B), 2, Fr(B, Ground)) \wedge Value(NAHF(B), 2) = 10$$
$$\wedge Value(NHF(B), 2) = 10 + Value(Fr(B, Ground), 2).$$

Let us reformulate (C14) as shown below. Axiom (C14) states that friction force comes into existence when a horizontal force of non-zero value is applied. Axiom (C14′) states that friction force comes into existence only if the net horizontal force applied on the block is different from zero. (Axiom (C14′) introduces a conditional dependency between additive effects.)

$$Value(NAHF(b), t) \neq 0 \longrightarrow \qquad \text{(C14′)}$$
$$PartContributor(NHF(b), t, Fr(b, Ground))$$

$$T_{ESI,0}[(C14')] = [z = \#sum\langle p'.PartContributor(NAHF(b), t, p')\rangle \wedge z \neq 0]$$

$$\longrightarrow PartContributor(NHF(b), t, Fr(b, Ground)).$$

$CIRC_{CECP1}(D)$, after (C14′) has been substituted for (C14), has more than one model. In addition to the model above, it has another model with facts that differ for time 2 (and times after 1):

$$\begin{aligned}
&Value(AHF(Frc, B), 2) = 10 \wedge PartContributor(NAHF(B), 2, AHF(Frc, B)) \\
&\wedge \exists p.(PartContributor(NAHF(B), 2, p) \wedge Value(p, 2) = -10) \\
&\wedge PartContributor(NHF(B), 2, NAHF(B)) \wedge Value(NAHF(B), 2) = 0 \\
&\wedge Value(NHF(B), 2) = 0.
\end{aligned}$$

The difference in the two models with respect to the facts about $PartContributor$ is that the former contains $\{PartContributor(NHF(B), 2, Fr(B, Ground)\}$ whereas the latter contains $\{\exists p.PartContributor(NAHF(B), 2, p)\}$, where p is necessarily different from $Fr(B, Ground)$ if $Value(Fr(B, Ground), 2)$ is different from -10. (Even if the sort for parameters is enumerated, that is $\mathcal{P}$ is *given*, we can imagine having an unrelated parameter – unrelated to forces applied on the block – with a value of -10 or an undefined value not constrained to be different from -10, that would result in a similar anomalous/unintended model.) The latter model goes against the physical chain of causality, and is an anomalous model.

4.3.2 Using Approach II

The example domain can be described as shown below. We ignore $RemoveHF$ action here, because its effects are complex and verbose but are also not directly relevant as $RemoveHF$ is not part of the narrative.

$$Initiates(ApplyHF(frc, b, v), F_{AHF}(frc, b, v), t) \tag{C16}$$

$$[\neg HoldsAt(F_{NAHF}(b), t) \wedge Initiates(A, F_{AHF}(frc, b, v), t)] \longrightarrow Initiates(A, F_{NAHF}(b), t) \tag{C17}$$

$$BreaksTo(ApplyHF(frc, b, v), AHF(frc, b), t, v) \tag{C18}$$

$$[a = ApplyHF(frc, b, v) \wedge (p = NAHF(b) \vee p = NHF(b))] \longrightarrow Breaks(a, p, t) \tag{C19}$$

$$HoldsAt(F_{AHF}(frc, b, v), t) \longrightarrow \tag{C20}$$
$$PartContributorWFluent(NAHF(b), t, F_{AHF}(frc, b, v), AHF(frc, b))$$
$$\exists frc, v.HoldsAt(F_{AHF}(frc, b, v), t) \longrightarrow \tag{C21}$$
$$PartContributorWFluent(NHF(b), t, F_{NAHF}(b), NAHF(b))$$
$$Value(NAHF(b), t) \neq 0 \longrightarrow \tag{C22}$$
$$PartContributorWFluent(NHF(b), t, F_{NAHF}(b), Fr(b, Ground))$$
$$[(p = NAHF(b) \vee p = NHF(b)) \wedge \neg\exists p', f.[HoldsAt(f, t) \wedge \tag{C23}$$
$$PartContributorWFluent(p, t, f, p')] \longrightarrow Value(p, t) = 0$$

$T_{ESI,0}[(C22)]$= $[z = \#sum\langle p', f.HoldsAt(f, t) \wedge PartContributorWFluent($
$NAHF(frc, b), t, f, p')\rangle \wedge z \neq 0] \longrightarrow PartContributor(NHF(b), t, Fr(b, Ground))$.

The circumscription policy for the above domain description is as follows:

$$CIRC_{CECP2}(D) = CIRC[(C1) \wedge \ldots \wedge (C7); Happens,$$
$$InitializedTrue, InitializedFalse]$$
$$\wedge\, CIRC[(C16) \wedge (C17); Initiates, Terminates]$$
$$\wedge\, CIRC[(C18) \wedge (C19) \wedge (EC11) \wedge (EC12); Breaks, BreaksTo]$$
$$\wedge\, CIRC[T_{ESI,0}[(C20) \wedge (C21) \wedge (C22)]; PartContributorWFluent]$$
$$\wedge\, (C23) \wedge (EC16p2) \wedge (C8) \wedge EC$$

We enumerate only the facts about $HoldsAt$, $Value$ and $PartContributor$ for models of $CIRC_{CECP2}(D)$. $CIRC_{CECP2}(D)$ has more than one model. One model has the following facts at time 1 (and times before 1):

$$Value(AHF(Frc, B), 1) = 0 \wedge Value(NAHF(B), 1) = 0$$
$$\wedge Value(NHF(B), 1) = 0 \wedge Value(Fr(B, Ground), 1) = 0,$$

and at time 2 (and times after 1):

$$HoldsAt(F_{AHF}(Frc, B, 10), 2) \wedge HoldsAt(F_{NAHF}(B), 2)$$

$$\wedge\, Value(AHF(Frc, B), 2) = 10$$
$$\wedge\, PartContributor(NAHF(B), 2, F_{AHF}(Frc, B, 10), AHF(Frc, B))$$
$$\wedge\, PartContributor(NHF(B), 2, F_{NAHF}(B), NAHF(B))$$
$$\wedge\, PartContributor(NHF(B), 2, F_{NAHF}(B), Fr(B, Ground))$$
$$\wedge\, Value(NAHF(B), 2) = 10$$
$$\wedge\, Value(NHF(B), 2) = 10 + Value(Fr(B, Ground), 2).$$

Another model which has the same facts as above for time 1 (and times before 1), but has the following facts for time 2 (and times after 1):

$$HoldsAt(F_{AHF}(Frc, B, 10), 2) \wedge HoldsAt(F_{NAHF}(B), 2)$$
$$\wedge\, Value(AHF(Frc, B), 2) = 10$$
$$\wedge\, PartContributorWFluent(NAHF(B), 2, F_{AHF}(Frc, B, 10), AHF(Frc, B))$$
$$\wedge\, \exists p.(PartContributorWFluent(NAHF(B), 2, F_{NAHF}(B), p)$$
$$\wedge\, Value(p, 2) = -10)$$
$$\wedge\, PartContributorWFluent(NHF(B), 2, F_{NAHF}(B), NAHF(B))$$
$$\wedge\, Value(NAHF(F), 2) = 0 \wedge Value(NHF(B), 2) = 0.$$

The difference in the two models with respect to the facts about *PartContributorWFluent* is that the former contains $\{PartContributorWFluent(NHF(B), 2, F_{NAHF}(B), Fr(B, Ground))\}$ whereas the latter contains $\{\exists p.PartContributorWFluent(NAHF(B), 2, F_{NAHF}(B), p)\}$, where p is necessarily different from $Fr(B, Ground)$ if $Value(Fr(B, Ground), 2)$ is different from -10. The latter is an anomalous model.

4.3.3 Using Approach III

The example domain can be described as shown below. The quantity, $AHF(frc, b)$, is not required in this approach. As before, DCAs are underlined.

$$Initiates(ApplyHF(frc, b, v), F_{AHF}(frc, b, v), t) \qquad \text{(C24)}$$

$$Terminates(RemoveHF(frc, b, v), F_{AHF}(frc, b, v), t) \tag{C25}$$

$$[(a = ApplyHF(frc, b, v) \vee a = RemoveHF(frc, b, v)) \wedge (p = NAHF(b) \vee p = NHF(b))] \longrightarrow Breaks(a, p, t) \tag{C26}$$

$$HoldsAt(AHF(frc, b, v), t) \longrightarrow PartValue(NAHF(b), t, \underline{frc}, v) \tag{C27}$$

$$\exists frc, v.HoldsAt(AHF(frc, b, v), t) \longrightarrow PartValue(NHF(b), t, \underline{b}, Value(NAHF(b), t)) \tag{C28}$$

$$\exists frc, v.HoldsAt(AHF(frc, b, v), t) \longrightarrow PartValue(NHF(b), t, \underline{Ground}, Value(Fr(b, Ground), t)) \tag{C29}$$

$$[(p = NAHF(b) \vee p = NHF(b)) \wedge \neg\exists px, r.PartValue(p, t, px, r) \longrightarrow Value(p, t) = 0 \tag{C30}$$

The circumscription policy for the above domain description is as follows:

$$\begin{aligned} CIRC_{CECP3}(D) = & CIRC[(C2) \wedge \ldots \wedge (C7); Happens, \\ & \qquad InitializedTrue, InitializedFalse] \\ & \wedge CIRC[(C24) \wedge (C25); Initiates, Terminates] \\ & \wedge CIRC[(C26) \wedge (EC11) \wedge (EC12); Breaks, BreaksTo] \\ & \wedge CIRC[T_{ESI,0}[(C27) \wedge (C28) \wedge (C29)]; PartValue] \\ & \wedge (C30) \wedge (EC16p) \wedge (C8) \wedge EC \end{aligned}$$

The condition $\exists frc, v.HoldsAt(AHF(frc, b, v), t)$ in Axiom (C29) is similar to $\exists frc.(Value(\ AHF(frc, b), t) \neq 0)$ in Axiom (C14). $CIRC_{CECP3}(D)$ has more than one model. In contrast, $CIRC_{CECP1}(D)$ had just one model when Axiom (C14) was used, and more than one model only when Axiom (C14) was replaced with Axiom (C14′).

We enumerate only the facts about $HoldsAt$, $Value$ and $PartContributor$ for models of $CIRC_{CECP3}(D)$. One model has the following facts at time 1 (and times before 1):

$$Value(NAHF(B), 1) = 0 \wedge Value(NHF(B), 1) = 0$$

$$\wedge\, Value(Fr(B, Ground), 1) = 0,$$

and at time 2 (and times after 1):

$$\begin{aligned}
&HoldsAt(AHF(Frc, B, 10), 2) \wedge PartValue(NAHF(B), 2, \underline{Frc}, 10) \\
&\wedge\, PartValue(NHF(B), 2, \underline{B}, 10) \\
&\wedge\, PartValue(NHF(B), 2, \underline{Ground}, Value(Fr(B, Ground), 2)) \\
&\wedge\, Value(NAHF(B), 2) = 10 \\
&\wedge\, Value(NHF(B), 2) = 10 + Value(Fr(B, Ground), 2).
\end{aligned}$$

Another model which has the same facts as above for time 1 (and times before that), but has the following facts for time 2 (and times after 1):

$$\begin{aligned}
&HoldsAt(AHF(F, B, 10), 2) \wedge PartValue(NAHF(B), 2, \underline{F}, 10) \\
&\wedge \exists px, r.[PartValue(NAHF(B), 2, \underline{px}, r) \wedge PartValue(NHF(B), 2, \underline{B}, 10 + r) \\
&\qquad \wedge\, PartValue(NHF(B), 2, \underline{Ground}, Value(Fr(B, Ground), 2)) \\
&\qquad \wedge\, r \neq 0 \wedge Value(NAHF(F), 2) = 10 + r \\
&\qquad \wedge\, Value(NHF(B), 2) = 10 + r + Value(Fr(B, Ground), 2)]
\end{aligned}$$

The difference in the two models with respect to the facts about $PartValue$ is that the former contains $\{PartValue(NHF(B), 2, \underline{B}, 10)\}$ whereas the latter contains $\{\exists px.PartValue(\ NAHF(B),\ 2, \underline{px}, R), PartValue(NHF(B), 2, \underline{B}, 10 + R)\}$, for any real value R different from 0.

To sum up, circumscription is inadequate for determining active continuous additive effects for all the three approaches. Alternative approaches to those three are possible, but we conjecture that circumscription would be inadequate for any similar approaches. This provokes investigation for an alternative transformation (preferably, circumscription-like) for encoding the necessary conditions for active additive effects. In the remainder we will just use $PartValue$ to describe continuous additive effects, but the results can be extended and applied to Approach I as well (which then precludes consideration for Approach II).

4.4 Indirect Discrete Additive Effects

Erdem and Gabaldon (2005,2006) differentiate between two types of additive effects. In the first type, when there is "too much increase/decrease" in the value of a quantity by concurrent additive effects then the "excess" is carried over to another quantity. In the second type, concurrent additive effects affecting a quantity have the same effect on another dependent quantity.

Consider the following example for the first type. Similar to the tanks example from (Erdem & Gabaldon, 2005), let a small tank (S) is suspended over a bigger tank (B). The small tank is filled discretely (that is the amount changes instantaneously) by acts $Fill(s, k, r)$ where r is the amount of water contributed by k-th (k is just a unique identifier) filling action to small tank s. When the total amount of water contributed exceeds the capacity of the small tank (denoted by quantity $Capacity(S)$) the remainder fills up the big tank. Let $Amount(S)$ ($Amount(B)$) denotes the amount of water in the small (big) tank. We can describe the above domain as shown below, with help of an auxiliary quantity $FillTotal(S)$ which denotes the total water contributed by all concurrent $Fill(S, k, r)$ acts. The effects on $FillTotal(S)$ are described using $BreaksPartTo$ relation (Section 4.2.2).

$$BreaksPartTo(Fill(s,k,r), FillTotal(S), t, r) \tag{E1}$$

$$\begin{aligned} &RightLimit(FillTotal(s), t, r) \\ &\wedge Value(Amount(s), t) + r \leq Value(Capacity(s), t) \end{aligned} \tag{E2}$$

$$\wedge BreaksPartBy(a, FillTotal(s), t, r_a) \longrightarrow BreaksPartTo(a, Amount(s), t, r_a)$$

$$\begin{aligned} &RightLimit(FillTotal(s), t, r) \wedge Value(Amount(s), t) \\ &\quad + r > Value(Capacity(s), t) \wedge BreaksPartTo(a, FillTotal(s), t, r_a) \\ &\qquad \longrightarrow BreaksTo(a, Amount(s), t, Value(Capacity(s), t)) \end{aligned} \tag{E3}$$

$$\begin{aligned} &RightLimit(FillTotal(s), t, r) \wedge Value(Amount(s), t) \\ &\quad + r > Value(Capacity(s), t) \wedge BreaksPartTo(a, FillTotal(s), t, r_a) \\ &\qquad \longrightarrow BreaksTo(a, Amount(B), t, Value(Amount(B), t) \\ &\qquad\quad + Value(Amount(s), t) + r - Value(Capacity(s), t)) \end{aligned} \tag{E4}$$

Axiom (E2) states that if the total amount of water in the small tank after new contributions would be less than its capacity then each fill act contributes additively. Axiom (E3) states that if the total amount of water in the small tank after new contributions would be more than the capacity then the amount of water in the tank changes to its capacity, which is represented by each fill act changing the amount of water in it nonadditively to the same amount equal to its capacity. And in the latter case, each fill act changes the amount of water in the big tank nonadditively to the same amount equal to the excess contribution to the small tank added with the amount already in it.

In case there are multiple small tanks suspended over the big tank then the effect described in Axiom (E4) should be additive. We can reformulate Axiom (E4) as below.

$$\begin{aligned}
&RightLimit(FillTotal(s), t, r) \\
&\wedge Value(Amount(s), t) + r > Value(Capacity(s), t) \\
&\wedge n = \#count\langle r', a'.BreaksPartTo(a', FillTotal(s), t, r') \wedge Happens(a', t)\rangle \quad \text{(E5)} \\
&\wedge BreaksPartTo(a, FillTotal(s), t, r_a) \longrightarrow BreaksPartBy(a, Amount(B), \\
&\qquad\qquad t, (Value(Amount(s), t) + r - Value(Capacity(s), t))/n)
\end{aligned}$$

Axiom (E5) expresses the effect of Axiom (E4) in terms of additive effects split equally across all responsible actions.

Similar to the case of continuous additive effects (Section 4.3), if r in $BreaksPartBy(a, FillTotal(s), t, r)$ for any action a can be negative circumscription would not be enough for determining active discrete additive effects (without anomalous additive effects).

To cover the second case, consider the following modification to the above example. Let the small tank is lying inside the big tank, and assume that the excess water poured into the small tank magically disappears. So the amount of water in the big tank goes up by the same amount that it goes up in the small tank. First

we assume that there is just one small tank inside the big tank.

$$BreaksPartTo(Fill(s,k,r), FillTotal(s), t, r) \tag{E6}$$

$$\begin{aligned} &RightLimit(FillTotal(s),t,r) \wedge Value(Amount(s),t) \\ &\quad + r \leq Value(Capacity(s),t) \wedge BreaksPartTo(a, FillTotal(s), t, r_a) \\ &\qquad \longrightarrow BreaksPartBy(a, Amount(s), t, r_a) \end{aligned} \tag{E7}$$

$$\begin{aligned} &RightLimit(FillTotal(s),t,r) \wedge Value(Amount(s),t) \\ &\quad + r > Value(Capacity(s),t) \wedge BreaksPartTo(a, FillTotal(s), t, r_a) \\ &\qquad \longrightarrow BreaksTo(a, Amount(s), t, Value(Capacity(s),t)) \end{aligned} \tag{E8}$$

$$\begin{aligned} &RightLimit(Amount(s),t,r) \wedge BreaksPartTo(a, FillTotal(s), t, r_a) \\ &\quad \longrightarrow BreaksTo(a, Amount(B), t, Value(Amount(B),t) \\ &\qquad + r - Value(Amount(s),t)) \end{aligned} \tag{E9}$$

If there are multiple small tanks inside the big tank then we can reformulate Axiom (E9) as below.

$$\begin{aligned} &RightLimit(Amount(s),t,r) \wedge BreaksPartTo(a, FillTotal(s), t, r_a) \wedge n = \\ &\quad \#count\langle r', a'.BreaksPartTo(a', FillTotal(s), t, r') \wedge Happens(a', t)\rangle \\ &\qquad \longrightarrow BreaksPartBy(a, Amount(B), t, (r - Value(Amount(s),t))/n) \end{aligned} \tag{E10}$$

4.5 Example III: 2-D Multiple Blocks Problem

With the extensions proposed in Section 4 (and a correct nonmonotonic reasoning technique to go with it, possible with the $CIRC^A$ transformation, which we describe in Section 4.6), it becomes possible to write domain-specific axioms in a way that they are applicable in a wider range of scenarios. We illustrate one case of that involving a 2-D multiple blocks problem. Multiple blocks are stacked over one other such that a block may have any number of blocks on it, but none are on two different blocks and none of them touch sideways at any time. Other than those assumptions, no specific configuration of blocks is needed. The coefficients of static and kinetic frictions between any two surfaces are given. The goal is to determine the

Table 4.3: Actions, predicates, quantities, fluents used in the multiple blocks problem.

Term	Type	Description
$ApplyHF(frc, b, r)$	Action	Apply horizontal force frc of value r on block b
$ApplyVF(frc, b, r)$	Action	Apply vertical force frc of value r on block b
$PlaceOn(b_1, b_2, v)$	Action	Act of placing block b_1 with initial velocity v on block b_2
$RemoveHF(frc, b, r)$	Action	Stop applied horizontal force frc of value r on block b
$RemoveVF(frc, b, r)$	Action	Stop applied vertical force frc of value r on block b
$IndOn(b_1, b_2)$	Predicate	block b_1 is transitively on a block b_2
$AHF(frc, b, r)$	Fluent	Block b is subject external horizontal force frc of value r
$AVF(frc, b, r)$	Fluent	Block b is subject to external vertical force frc of value r
$Conn(b_1, b_2)$	Fluent	blocks b_1 and b_2 share a common base other than the *Ground*.
$On(b_1, b_2)$	Fluent	block b_1 is directly on block b_2
$PtMbl(b)$	Fluent	block b is potentially mobile
$SL(b_1, b_2)$	Fluent	block b_1 is sliding to the left relative to a block b_2
$SR(b_1, b_2)$	Fluent	block b_1 is sliding to the right relative to a block b_2
$CKF(b_1, b_2)$	Quantity	the coefficient of kinetic friction between blocks b_1 and b_2
$CSF(b_1, b_2)$	Quantity	the coefficient of static friction between blocks b_1 and b_2
$FbBA(b_1, b_2)$	Quantity	the force applied from above on block b_1 by block b_2
$Fr(b_1, b_2)$	Quantity	the friction force applied on block b_1 by block b_2
$Mass(b)$	Quantity	Mass of block b
$NAHF(b)$	Quantity	Net external horizontal force on block b
$NAVF(b)$	Quantity	Net external vertical force on block b
$NHF(b)$	Quantity	Net horizontal force on block b
$Pos(b)$	Quantity	Horizontal position of the center of mass of block b
$Wt(b)$	Quantity	the weight of block b

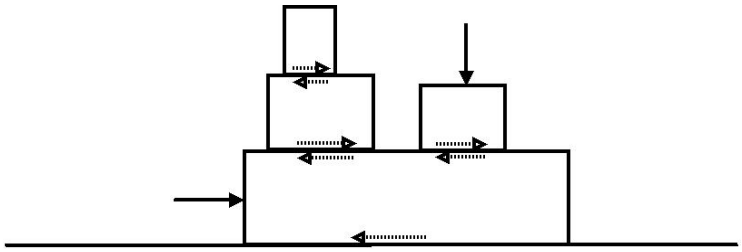

Figure 4.3: An example configuration of multiple blocks. The solid lines (with arrow-heads) denote external forces, and the dotted lines denote friction forces with arrows drawn under the assumption that the lowest block has a tendency to move to the right while the rest of the blocks have a tendency to stay behind relatively.

accelerations and velocities of the blocks when horizontal and/or vertical forces are applied to the blocks and/or a block with possibly non-zero initial velocity is placed on top of one of the blocks. Figure 4.3 depicts an example configuration of blocks and externally applied forces. The multiple blocks problems is a generalization of the single block problem discussed in Section 4.3.

The domain is axiomatized through Axioms (D1)–(D37). Description of additive effects is given in Section 4.5.1. The actions, predicates, quantities and fluents used for the descriptions are listed in the Table 4.3. The following symbols are used in abbreviated form: $Value$ (Val), $PartValue$ ($PtVal$), $HoldsAt$ ($Holds$), $Ground$ ($Grnd$).

We also assume that the net vertical force on any block is in the direction of gravity, that is there is no motion in the vertical direction (Axiom (D1)). Axiom (D2) expresses that any block lies entirely on one block. The ground is treated as a special block with velocity and acceleration assumed to be constant zero (Axiom (D3)).

$$Val(NAVF(b), t) + Val(Wt(b), t) > 0 \tag{D1}$$

$$[Holds(On(b_1, b_2), t) \wedge Holds(On(b_1, b_3), t)] \longrightarrow b_2 = b_3 \tag{D2}$$

$$[p = \delta(Grnd) \vee p = \delta(\delta(Grnd))] \longrightarrow Val(p, t) = 0 \tag{D3}$$

Application or removal of a force in horizontal or vertical directions affects

persistence of some fluents $AHF(frc, b, v)$ and $AVF(frc, b, v)$, and the act of placing a block on another block affects persistence of some fluent $On(b_1, b_2)$ as per Axioms (D4) and (D5). $PlaceOn(b_1, b_2, v)$ changes the velocity of block b_1 to v (Axiom (D6)) and its position to that of b_2 (Axiom (D7)).

$$\begin{aligned} Initiates(a, f, t) \longleftarrow [&(a = ApplyHF(frc, b, v) \wedge f = AHF(frc, b, v)) \\ &\vee (a = ApplyVF(frc, b, v) \wedge f = AVF(frc, b, v)) \\ &\vee (a = PlaceOn(b_1, b_2, v) \wedge f = On(b_1, b_2))] \end{aligned} \tag{D4}$$

$$\begin{aligned} Terminates(a, f, t) \longleftarrow [&(a = RemoveHF(frc, b, v) \wedge f = AHF(frc, b, v)) \\ &\vee (a = RemoveVF(frc, b, v) \wedge f = AVF(frc, b, v))] \end{aligned} \tag{D5}$$

$$BreaksTo(PlaceOn(b_1, b_2, v), \delta(Pos(b_1)), t, v) \tag{D6}$$

$$BreaksTo(PlaceOn(b_1, b_2, v), Pos(b_1), t, Value(Pos(b_2), t)) \tag{D7}$$

The following are non-frame fluents: $Conn(b_1, b_2)$, $PtMbl(b)$, $SR(b_1, b_2)$, $SL(b_1, b_2)$, i.e. their states are neither initialized true or false nor initiated or terminated directly by actions. Axiom (D8) states that the $On(b_1, b_2)$ relationship is asymmetric and irreflexive. Axiom (D9) defines the predicate $IndOn$. Transitive closures cannot be defined in First-order Logic (Grädel, 1992), therefore we use a predicate instead of a fluent to represent the indirectly-on relationship and circumscribe the basic axioms that derive the transitive relations. Axioms (D10), (D11) describe the fluents connected and potentially-mobile, respectively. If two blocks share a common base, then a force on one potentially affects the other. If a block is applied a net external horizontal force or has a non-zero velocity, then it potentially moves the other connected blocks.

$$Holds(On(b_1, b_2), t) \longrightarrow [b_1 \neq b_2 \wedge \neg Holds(On(b_2, b_1), t)] \tag{D8}$$

$$\begin{aligned} &CIRC[IndOn(b_1, b_3, t) \longleftrightarrow \\ &\qquad [Holds(On(b_1, b_3), t) \vee (Holds(On(b_1, b_2), t) \wedge IndOn(b_2, b_3, t)); IndOn] \end{aligned} \tag{D9}$$

$$\begin{aligned} Holds(Conn(b_1, b_2), t) \longleftrightarrow [&b_1 \neq Grnd \wedge b_2 \neq Grnd \\ &\wedge (IndOn(b_1, b_2, t) \vee IndOn(b_2, b_1, t) \end{aligned} \tag{D10}$$

$$\vee\,(IndOn(b_1,b_3,t)\wedge IndOn(b_2,b_3,t)\wedge b_3\neq Grnd))]$$

$$Holds(PotMbl(b_1),t)\longleftrightarrow \tag{D11}$$
$$[(Val(\delta(Pos(b)),t)\neq 0\vee Val(NAHF(b),t)\neq 0)\wedge Holds(Conn(b,b_1),t)]$$

Axioms (D12)–(D16) describe the effects of a block on top sliding to the right, sliding to the left, or not sliding relative to the surface below. Axioms (D12)–(D14) describe the effects when velocities of the blocks in contact are the same. The direction of friction is then determined by fluents $SR(b_1,b_2)$ and $SL(b_1,b_2)$. Axioms (D15)–(D16) describe the effects when velocities of the blocks in contact are different and the direction of friction is determined from their relative velocity. Axioms (D12)–(D16) give the equations and mathematical constraints on friction that hold in each of the different cases. For example, according to Axiom (D12), when b_1 is sliding to the right with respect to b_2, the equation $Val(Fr(b_1,b_2),t)=-1\times Val(CKF(b_1,b_2),t)\times Val(FbBA(b_2,b_1),t)$ and the constraint $Val(\delta(\delta(Pos(b_1))),t)-Val(\delta(\delta(Pos(b_2))),t)>0$ are active.

$$[Holds(On(b_1,b_2),t)\wedge Holds(SR(b_1,b_2),t)$$
$$\wedge Val(\delta(Pos(b_1)),t)=Val(\delta(Pos(b_2)),t)]\longrightarrow \tag{D12}$$
$$[Val(Fr(b_1,b_2),t)=-1\times Val(CKF(b_1,b_2),t)\times Val(FbBA(b_2,b_1),t)$$
$$\wedge Val(\delta(\delta(Pos(b_1))),t)-Val(\delta(\delta(Pos(b_2))),t)>0\wedge\neg Holds(SL(b_1,b_2),t)]$$
$$[Holds(On(b_1,b_2),t)\wedge Holds(SL(b_1,b_2),t)$$
$$\wedge Val(\delta(Pos(b_1)),t)=Val(\delta(Pos(b_2)),t)]\longrightarrow \tag{D13}$$
$$[Val(Fr(b_1,b_2),t)=+1\times Val(CKF(b_1,b_2),t)\times Val(FbBA(b_2,b_1),t)$$
$$\wedge Val(\delta(\delta(Pos(b_1))),t)-Val(\delta(\delta(Pos(b_2))),t)<0\wedge\neg Holds(SR(b_1,b_2),t)]$$
$$[Holds(On(b_1,b_2),t)\wedge\neg Holds(SR(b_1,b_2),t)\wedge\neg Holds(SL(b_1,b_2),t)$$
$$\wedge Val(\delta(Pos(b_1)),t)=Val(\delta(Pos(b_2)),t)]\longrightarrow \tag{D14}$$
$$[Val(\delta(\delta(Pos(b_1))),t)=Val(\delta(\delta(Pos(b_2))),t)$$
$$\wedge|Val(Fr(b_1,b_2),t)|\leqslant Val(CSF(b_1,b_2),t)\times Val(FbBA(b_2,b_1),t)]$$
$$[Holds(On(b_1,b_2),t)\wedge Val(\delta(Pos(b_1)),t)>Val(\delta(Pos(b_2)),t)]\longrightarrow \tag{D15}$$

$$[Val(Fr(b_1,b_2),t) = -1 \times Val(CKF(b_1,b_2),t) \times Val(FbBA(b_2,b_1),t)]$$
$$[Holds(On(b_1,b_2),t) \wedge Val(\delta(Pos(b_1)),t) < Val(\delta(Pos(b_2)),t)] \longrightarrow \quad \text{(D16)}$$
$$[Val(Fr(b_1,b_2),t) = +1 \times Val(CKF(b_1,b_2),t) \times Val(FbBA(b_2,b_1),t)]$$

Axiom (D17) states that any two blocks in contact apply equal and opposite friction forces on each other. Axiom (D18) gives the equation for determining the acceleration of a block.

$$Holds(On(b_1,b_2),t) \longrightarrow Val(Fr(b_1,b_2),t) = -1 \times Val(Fr(b_2,b_1),t) \quad \text{(D17)}$$
$$b \neq Grnd \longrightarrow Val(\delta(\delta(Pos(b))),t) = Val(NHF(b),t)/Val(Mass(b),t) \quad \text{(D18)}$$

4.5.1 Additive Effects

There are various forces that need to be combined to get the net horizontal and vertical forces. Axioms (D19)–(D24) define individual contributions to net externally applied horizontal force, net externally applied vertical force, force by block above, and net horizontal force. The DCAs are underlined.

$NAHF$ is computed by summing AHF (Axiom (D19)). Similarly, $NAVF$ is computed by summing AVF (Axiom (D20)). The weight of a block above and any vertical forces applied on the block above contribute to $FbBA$ (Axiom (D21)). Any forces due to blocks above the block above also contribute to $FbBA$ (Axiom (D22)). Finally, the net horizontal force is sum of the externally applied forces (Axiom (D23)) and friction forces from blocks in contact (Axiom (D24)). Notice that Axioms (D19)–(D24) are very general, not constrained to any specific configuration (within the general assumptions declared in the beginning), which is not viable in the existing formalisms.

$$Holds(AHF(frc,b,v),t) \longrightarrow PtVal(NAHF(b),t,\underline{frc},v) \quad \text{(D19)}$$
$$Holds(AVF(frc,b,v),t) \longrightarrow PtVal(NAVF(b),t,\underline{frc},v) \quad \text{(D20)}$$
$$Holds(On(b_2,b_1),t) \longrightarrow [PtVal(FbBA(b_1,b_2),t,\underline{Wt(b_2)},Val(Wt(b_2),t)) \quad \text{(D21)}$$
$$\wedge\, PtVal(FbBA(b_1,b_2),t,\underline{NAVF(b_2)},Val(NAVF(b_2),t))]$$
$$[IndOn(b_3,b_2,t) \wedge Holds(On(b_2,b_1),t)] \longrightarrow$$

$$[PtVal(FbBA(b_1,b_2),t,\underline{Wt(b_3)},Val(Wt(b_3),t)) \wedge PtVal(FbBA(b_1,b_2),t,\underline{NAVF(b_3)},Val(NAVF(b_3),t))] \quad \text{(D22)}$$

$$\exists frc,v.Holds(AHF(frc,b,v),t) \longrightarrow PtVal(NHF(b),t,\underline{b},Val(NAHF(b),t)) \quad \text{(D23)}$$

$$[Holds(On(b_1,b_2),t) \wedge Holds(PotMbl(b_1),t)] \longrightarrow [PtVal(NHF(b_1),t,\underline{b_2},Val(Fr(b_1,b_2),t)) \wedge PtVal(NHF(b_2),t,\underline{b_1},Val(Fr(b_2,b_1),t))] \quad \text{(D24)}$$

Axiom (D22) can equivalently be axiomatized as a recursive definition:

$$[IndOn(b_3,b_2,t) \wedge Holds(On(b_2,b_1),t)] \longrightarrow PtVal(FbBA(b_1,b_2),t,\underline{b_3},Val(FbBA(b_2,b_3),t))$$

In the recursive definition, however, $FbBA(b_1,b_2)$ is dependent on $FbBA(b_2,b_3)$ for its value. Such cyclical dependencies, between $FbBA$ and $FbBA$ for example, make the analysis of CIRC^A transformations harder. We will see in Section 4.6.3.1 that CIRC^A transformation of syntactically non-cyclical descriptions of additive effects is equivalent to the completion.

Axioms (D25)–(D27) define the value of additive parameters when no additive values are known.

$$\neg\exists px,v.PtVal(NAHF(b),t,px,v) \longrightarrow Val(NAHF(b),t)=0 \quad \text{(D25)}$$

$$\neg\exists px,v.PtVal(NAVF(b),t,px,v) \longrightarrow Val(NAVF(b),t)=0 \quad \text{(D26)}$$

$$\neg\exists px,v.PtVal(NHF(b),t,px,v) \longrightarrow Val(NHF(b),t)=0 \quad \text{(D27)}$$

Axioms (D28)–(D36) express relationships between quantities and actions which potentially break the continuity in their values. The axioms for breaks in the continuities of the values of additive quantities should typically closely correspond with the axioms about their additive values. For example, Axioms (D29)–(D32) and (D34)–(D35) closely correspond with Axioms (D19)–(D24). Likewise, for non-

additive quantities.

$$InitiatesOrTerminates(a, f, t) \equiv^{def} [Initiates(a, f, t) \vee Terminates(a, f, t)] \quad \text{(D28)}$$

$$Breaks(a, NAHF(b), t) \longleftarrow InitiatesOrTerminates(a, AHF(frc, b, v), t) \quad \text{(D29)}$$

$$Breaks(a, NAVF(b), t) \longleftarrow InitiatesOrTerminates(a, AVF(frc, b, v), t) \quad \text{(D30)}$$

$$Breaks(a, FbBA(b_1, b_2), t) \longleftarrow InitiatesOrTerminates(a, On(b_2, b_1), t) \quad \text{(D31)}$$

$$Breaks(a, FbBA(b_1, b_2), t) \longleftarrow [Holds(IndOn(b_2, b_1), t) \wedge Breaks(a, NAVF(b_2), t)] \quad \text{(D32)}$$

$$Breaks(a, p, t) \longleftarrow [(p = Fr(b_1, b_2) \vee p = Fr(b_2, b_1)) \wedge Holds(PotMbl(b_1), t) \wedge Holds(On(b_1, b_2), t) \wedge \exists b, b3.((p' = NAHF(b) \vee p' = FbBA(b, b_3)) \wedge Breaks(a, p', t) \wedge Holds(Conn(b, b_1), t))] \quad \text{(D33)}$$

$$Breaks(a, NHF(b), t) \longleftarrow Breaks(a, NAHF(b), t) \quad \text{(D34)}$$

$$Breaks(a, p, t) \longleftarrow [(p = NHF(b_1) \vee p = NHF(b_2)) \wedge Holds(On(b_1, b_2), t) \wedge Holds(PotMbl(b_1), t) \wedge Breaks(a, Fr(b_1, b_2), t)] \quad \text{(D35)}$$

$$Breaks(a, p, t) \longleftarrow [Breaks(a, NHF(b), t) \wedge (p = \delta(\delta(Pos(b))) \vee p = \delta(Pos(b)))] \quad \text{(D36)}$$

Axiom (D37) asserts the unique name assumptions.

$$UNA[ApplyHF, ApplyVF, PlaceOn, RemoveHF, RemoveVF] \wedge UNA[Conn, On, PtMbl, SL, SR] \wedge UNA[CKF, CSF, FbBA, Fr, Mass, NAHF, NAVF, NHF, Pos, Wt, \delta] \quad \text{(D37)}$$

Finally, let D_M refers to (D1)–(D37). D_M axiomatizes scenario-independent aspects of the domain, and can be partitioned as shown below. Scenario-specific aspects such as the configuration of blocks and external action occurrences can be described by particularizing $Nar(D_M)$.

$$Eff(D_M) = [(D4) \wedge (D5)]$$

$$Inst(D_M) = [(D6) \wedge (D7) \wedge (D28) \wedge \ldots \wedge (D36)]$$
$$Con(D_M) = [(D3) \wedge (D12) \wedge \ldots \wedge (D18) \wedge (D25) \wedge \ldots \wedge (D27)]$$
$$Con_P(D_M) = [(D19) \wedge \ldots \wedge (D24)]$$
$$Una(D_M) = [(D37)]$$
$$Cnst(D_M) = [(D8) \wedge \ldots \wedge (D11)]$$
$$Rem(D_M) = [(D1) \wedge (D2)]$$

4.6 Default Reasoning in the Extended Event Calculus

We extend the circumscription policy $\text{CIRC}_{CEC}(D)$ to model the additional default assumption that by default a given additive effect is not active at a given time point, carrying on with the forced separation strategy. The new circumscription policy, $CIRC_{CECA}(D)$, is given below.[12] The new parts are underlined.

$$\begin{aligned}
CIRC_{CECA}(D) = {} & CIRC[Nar(D); Happens, InitialisedTrue, InitialisedFalse] \\
& \wedge CIRC[Eff(D); Initiates, Terminates] \\
& \wedge CIRC[Inst(D) \wedge (EC11) \wedge (EC12) \wedge \underline{(EC18p)}; Breaks; BreaksTo] \\
& \wedge \underline{CIRC[Inst_P(D); BreaksPartBy]} \\
& \wedge \underline{CIRC^A[T_{ESI}[Con_P(D)]; PartValue]} \\
& \wedge [Con(D) \wedge \underline{(EC16p)}] \wedge Cnst(D) \wedge Rem(D) \wedge Una(D) \wedge [EC \wedge \underline{(EC17p)}].
\end{aligned}$$

The value of additive quantities appearing in the antecedents of $Con_P(D)$ in terms of their additive values is accounted for by the ESI-reformulation (Definition 4.2.1) of the axioms. Note that $T_{ESI}[Con_P(D)]$ is a DD-derivative (Definition 3.3.1) of $T_{ESI,0}[Con_P(D)]$. The ESI-reformulation of Axiom (D23) is given below as an example.

$$\begin{aligned}
T_{ESI}[(D23)] = {} & [\exists f, v. Holds(AHF(f, b, v), t) \wedge p = NAHF(b) \\
& \wedge [(\#count\langle r, px. PtVal(p, t, px, r)\rangle \neq 0 \wedge z = \#sum\langle r, px. PtVal(p, t, px, r)\rangle) \vee \\
& (\#count\langle r, px. PtVal(p, t, px, r)\rangle = 0 \wedge z = Val(p, t))]]
\end{aligned}$$

[12] A in CECA indicates use of **a**ggregates or special description of **a**dditive effects in the axioms.

$$\longrightarrow PtVal(NHF(b), t, \underline{b}, z)$$

For the circumscription policy for Approach I involving the $PartContributor$ predicate, we can redefine $PartContributor$ relation as a quaternary relation such that $PartContributor(P_1, T, P_2, R)$ denotes that quantity P_1 has an additive value of $Value(P_2, T) = R$ at time T, and reformulate Axiom (EC16p1) as

$$[\exists r', p.PartContributor(p_1, t, p, r') \wedge s = \#sum\langle r, p_2.PartContributor(p_1, t, p_2, r)\rangle] \longrightarrow Value(p_1, t) = s.$$

Then $Con_P(D)$ axioms can be circumscribed as:

$$CIRC^A[T_{ESI}[Con_P(D)]; PartContributor].$$

Recall that in the single block example, Section 4.3.3, $Con_P(D) = (C27) \wedge (C28) \wedge (C29)$. Let $F(PartValue)$= $T_{ESI}[Con_P(D)]$= $T_{ESI,0}[Con_P(D)]$ and M= $\{PartValue(NAHF(B), 2, \underline{F},$
$10), \exists px.PartValue(NAHF(B), 2, \underline{px}, R), PartValue(NHF(B), 2, \underline{B}, 10 + R)\}$, for any non-zero real value R. Also recall that M is a model of $CIRC[F(PartValue); PartValue]$. M is however not a model of $CIRC^A[F(PartValue); PartValue]$ because $\{PartValue(NAHF(B), 2, \underline{F}, 10), PartValue(NHF(B), 2, \underline{B}, 10 + R)\}$ is a model of $F^{**}(partValue)$, where $PartValue$ is replaced by variable predicate symbol $partValue$ in the **-transformation (Definition 3.1.2) of F.

We present a few results towards correctness of the formalization in Section 4.6.1. The circumscription and the $CIRC^A$ transformations of formulas in $CIRC_{CECA}(D)$ policy are equivalent to the completion under different constraints, as we show in Sections 4.6.2 and 4.6.3.

4.6.1 On Correctness

Notation 4.6.1. We assume that $Inst_P(D)$, like $Con_P(D)$, is in implicative form. Further, we assume during this discussion on correctness that the antecedents of $Inst_P(D)$ and $Con_P(D)$ do not contain predicates $BreaksPartBy$, $BreaksTo$, $Brea$-

ks, or $PartValue$. The antecedents may however contain $Value(Par, t)$s for some additive parameters Par. We also assume that D is sound with respect to restrictions that a parameter is at no time instance subject to effects described in $Inst_P(D)$ (likewise, $Con_P(D)$) and those described in $Inst(D)$ (likewise, $Con(D)$) simultaneously.

Definition 4.6.2. We say that a fact is *antecedent-justified* in a model for an implicative formula if there exists an axiom in the implicative formula whose consequent is the fact (possibly after variable substitution) and the corresponding antecedent (possibly empty) is true. If there exists no such axiom then the fact is said to be *antecedent-unjustified*.

Antecedent-justified and justified (Chapter 3) are overlapping notions in the context of implicative formulas, but the former is stronger than the latter in general. For example, P(1) in the ag-minimal model of Fo_{12}, {P(1), Q(2)}, is antecedent-unjustified. However, antecedent-justified and justified are same notions in the context of $Inst_P(D)$ and $Con_P(D)$ when the aggregate predicate symbols, $BreaksPartBy$ and $PartValue$, do not occur in the antecedents.

$CIRC[Inst_P(D); BreaksPartBy]$ implies that only antecedent-justified $BreaksPartBy$ facts are true; see Proposition 4.6.7 in Section 4.6.2, derived using Proposition 2 from (Lifschitz, 1994).

When formulas contain aggregate summation, the standard minimal models may be weak, that is they may contain unjustified facts; cf. examples Fo_1 and Fo_4 (Section 3.1) and the single block example (Section 4.3.3). Intuitively, weak models contain additive effects that yield a very different future, different from that given by additive effects in the non-weak models, but are not necessitated by current states of the world where they are inferred active (hence unjustified). And hence, weak models go against the physical chain of causality. We reckon the models with unjustified additive effects an unintended consequence of the mechanism used for computing the cumulative effect, that is the aggregate summation. The CIRC^A transformation of formulas return formulas that are satisfied only and by all the non-weak models of the former; cf. Proposition 3.2.2 (Section 3.2). For some theories, the afore-stated unintended consequence may actually be useful, and for

such applications a combination of the standard minimum model semantics and non-weak minimal model semantics or alternative nonmonotonic reasoning techniques may be considered.

Since $Con_P(D)$ is in implicative form and its consequences are $PartValue$ predicates which occur only within aggregate formulas in the antecedents, by Proposition 3.2.3 (Section 3.2), $CIRC^A[T_{ESI,0}[Con_P(D)]; PartValue]$ implies that only justified $PartValue$ facts are true. When a value different from summation is chosen when $PartValue(p, t, px, r)$ is not satisfied, $CIRC^A[T_{ESI}[Con_P(D)]; PartValue]$ may return models with unjustified $PartValue$ facts if the non-existence is checked using the formula, $\neg\exists r, px.PartValue(p, t, px, r)$; cf. example Fo_{13} in Section 3.2. That is not the case if aggregate formulas are used instead to check the same condition; cf. Proposition 3.3.1 (Section 3.3). Again by Proposition 3.2.3, $CIRC^A[T_{ESI}[$ $Con_P(D)]$; $PartValue]$ implies that only justified $PartValue$ facts are inferred.

Therefore, by CIRC_{CECA}, only antecedent-justified additive effects are inferred to be active, that is only antecedent-justified $BreaksPartBy$ and $PartValue$ facts are inferred. Proposition 4.6.1 follows directly from Proposition 2 from (Lifschitz, 1994) and Proposition 3.2.3.

Proposition 4.6.1. *$CIRC[Inst_P(D); BreaksPartBy] \longrightarrow Comp(Inst_P(D))$, and $CIRC^A[T_{ESI}[Con_P(D)]; PartValue] \longrightarrow Comp(T_{ESI}[Con_P(D)])$.*

The second implication in Proposition 4.6.1 does not hold in the other direction as CIRC^A transformation rules out circular justifications, which are allowed by the completion (cf. Section 3.2 and example Fo_2). But, given the result of Proposition 4.6.1, $Inst_P(D)$ and $Con_P(D)$ axioms can be faithfully mapped into $Inst(D_s)$ and $Con(D_s)$ axioms (interpreted by the CIRC_{CEC} policy) for specific instances of domain D, denoted by D_s. Some axioms in $Inst_P(D)$ and $Con_P(D)$ may describe multiple additive effects. Such effects are described by separate axioms for the domain D_s. For example, Axiom (B14) describes effect of $Scoop(s, v)$ action for any s, v (Section 4.2.2). If up to two scoop actions, $Scoop(1, v)$ and $Scoop(2, v)$, are possible in D_s, then Axiom (B14) is replaced by the following couple of axioms.

$$BreaksPartBy(Scoop(1, v), Level(v), t, -1 \times R) \longleftarrow Value(Level(v), t) \geqslant R$$

$$BreaksPartBy(Scoop(2, v), Level(v), t, -1 \times R) \longleftarrow Value(Level(v), t) \geqslant R$$

Similarly, Axiom (D19) describes the effect of applying a force on a block b (Section 4.5.1). If up to two forces, Frc_1 and Frc_2, are possible in D_s, then Axiom (D19) is replaced by the following couple of axioms. (DCAs are underlined as before.)

$$Holds(AHF(Frc_1, b, v), t) \longrightarrow PtVal(NAHF(b), t, \underline{Frc_1}, v)$$
$$Holds(AHF(Frc_2, b, v), t) \longrightarrow PtVal(NAHF(b), t, \underline{Frc_2}, v)$$

The axioms for D_s, constructed from $Inst_P(D)$ and $Con_P(D)$, are denoted by $Inst_P(D_s)$ and $Con_P(D_s)$ respectively. After the effects that can be active concurrently and are additive are represented through separate axioms for D_s, in $Inst_P(D_s)$ and $Con_P(D_s)$, they can be mapped into axioms describing the net effect.

We assume, without loss of generality, that axioms describing discrete additive effects on a parameter Par are of the form given below, for $0 < i \leq n$, for some finite n. Act_i and Act_j, $i \neq j$, may refer to the same actions. λ_i denotes the mathematical expression that gives the value of an additive discrete change in the value of Par (and Ψ_i denotes the antecedent condition).

$$[\Psi_i \wedge v_i = \lambda_i] \longrightarrow BreaksPartBy(Act_i, Par, t, v_i)$$

The axioms describing additive direct effects of actions on a parameter can be mapped into n axioms describing the combined effects on the parameter as shown below, for each $0 < i \leq n$. We use $Happens$ preconditions for describing the effects of concurrent actions (Miller & Shanahan, 2002).

$$[\Big[\bigwedge_{0<j<i, i<j\leq n} (Happens(Act_j, t) \wedge \Psi_j \wedge v_j = \lambda_j) \vee ((\neg Happens(Act_j, t) \vee \neg\Psi_j) \wedge v_j = 0)\Big] \wedge \Psi_i \wedge v = Value(Par, t) + \sum_{0<j<i, i<j\leq n} v_j + \lambda_i]$$
$$\longrightarrow BreaksTo(Act_i, Par, t, v)$$

Similarly, we assume, without loss of generality, that axioms describing continuous additive effects on a parameter Par are of the form given below, for $0 < i \leq n$, for some finite n. Here, λ_i denotes the expression that gives an additive value of Par.

$$[\Psi_i \wedge v_i = \lambda_i] \longrightarrow PartValue(Par, t, px_i, v_i)$$

The axioms describing the additive values of a parameter can be mapped into an axiom describing the net value of the parameter as shown below.

$$[\Big[\bigwedge_{0<j\leq n} (\Psi_j \wedge v_j = \lambda_j) \vee (\neg\Psi_j \wedge v_j = 0)\Big] \wedge \bigvee_{0<j\leq n} \Psi_j \wedge v = \sum_{0<j\leq n} v_j]$$
$$\longrightarrow Value(Par, t) = v$$

The axioms describing the net effects, constructed from axioms in $Inst_P(D_s)$ and $Con_P(D_s)$, are denoted by $Inst(D_s)$ and $Con(D_s)$ respectively. Lemma 4.6.2 follows from the way $Inst(D_s)$ is constructed from $Inst_P(D_s)$, $Inst_P(D_s)$ is constructed from $Inst_P(D)$, $Con(D_s)$ is constructed from $Con_P(D_s)$, and $Con_P(D_s)$ is constructed from $Con_P(D)$. Proposition 4.6.1 and Lemma 4.6.2 can be combined to give Proposition 4.6.3.

Lemma 4.6.2. *Let M_P be a model of $[Comp(Inst_P(D_s)) \wedge (EC16p)]$. Then there exists a model M of $[Inst(D_s) \wedge (EC10)]$, and vice-versa, such that $M_P[Happens]$ $= M[Happens]$ and*

$$M_P \models RightLimit(p, t, v) \longleftrightarrow M \models RightLimit(p, t, v)$$

Likewise, let M_P be a model of $[Comp(Con_P(D_s)) \wedge (EC18p)]$, then there exists a model M of $[Con(D_s)]$, and vice-versa, such that

$$M_P \models Value(p, t) = v \longleftrightarrow M \models Value(p, t) = v$$

Proposition 4.6.3. *Let M_P be a model of $[CIRC[Inst_P(D_s); BreaksPartBy] \wedge$ $(EC16p)]$. There exists a model M of $[Inst(D_s) \wedge (EC10)]$ such that $M_P[Happens]$*

$= M[Happens]$ *and*

$$M_P \models RightLimit(p,t,v) \longleftrightarrow M \models RightLimit(p,t,v)$$

Likewise, let M_P *be a model of* $[CIRC^A[Con_P(D_s); PartValue] \wedge (EC18p)]$*, then there exists a model* M *of* $[Con(D_s)]$ *such that*

$$M_P \models Value(p,t) = v \longleftrightarrow M \models Value(p,t) = v$$

The results of Proposition 4.6.3 can be extrapolated to $Inst_P(D)$ and $Con_P(D)$, the generalized (that is, scenario-independent) representations of $Inst_P(D_s)$ and $Con_P(D_s)$, with imposition of constraints on $RightLimit(p,t,v)$ and $Value(p,t) = v$, restricting them to scenario-specific inferences. Crucially, the mappings given above are possible only because the active additive effects inferred from $Inst_P(D)$ (or $Inst_P(D_s)$) and $Con_P(D)$ (or $Con_P(D_s)$) are all justified.

4.6.2 Restrictions on Domain-Specific Axioms and First-order Reducibility

If the preconditions for axioms describing $Happens$ facts, the effects of actions on fluents using $Initiates$, $Terminates$ and the instantaneous effects of actions on parameters using $Breaks$, $BreaksTo$, $BreaksPartBy$ do not mention the respective predicates, then circumscription of corresponding domain axioms is equivalent to the predicate completion.

Propositions 4.6.4 and 4.6.5 are same as Propositions 2 and 3 from (Miller & Shanahan, 1996), and they follow directly from Propositions 2 and 14 from (Lifschitz, 1994).

Notation 4.6.3. We assume, without loss of generality, $Nar(D)$, $Eff(D)$, $Inst(D)$, $Inst_P(D)$ are in Clark Normal Form.

Proposition 4.6.4. *Let* $Nar(D) = [Happens(a,t) \longleftarrow \Delta(a,t)]$*. If* $\Delta(a,t)$ *does not mention the predicate Happens then*

$$CIRC[Nar(D); Happens] \equiv Comp(Nar(D)).$$

Proposition 4.6.5. *Let* $Eff(D) = [(Initiates(a, f, t) \longleftarrow \Delta_1(a, f, t)) \wedge (Terminates(a,\ f, t) \longleftarrow \Delta_2(a, f, t))]$. *If none of* $\Delta_i(a, f, t)$, $1 \leqslant i \leqslant 3$, *mention the predicates* $Initiates$ *or* $Terminates$ *then*

$$CIRC[Eff(D); Initiates, Terminates] \equiv Comp(Eff(D)).$$

We review Proposition 1 from (Miller & Shanahan, 1996) below. It is used for proving Proposition 4.6.6.

Proposition 1 (Miller & Shanahan, 1996). Let $Inst(D) = [(Breaks(a, p, t) \longleftarrow \Delta_1(a, p, t)) \wedge (BreaksTo(a,\ p,\ t,\ r) \longleftarrow \Delta_2(a, p, t, r))]$. If none of $\Delta_1(a, p, t)$ and $\Delta_2(a, p, t, r)$ mention $Breaks$ or $BreaksTo$ then $CIRC[Inst(D) \wedge (EC11) \wedge (EC12); Breaks; BreaksTo]$ entails

$$Breaks(a, p, t) \longleftrightarrow \exists p_1(p = \delta(p_1) \wedge Breaks(a, p_1, t)) \vee \Delta_1(a, p, t) \vee \exists r(\Delta_2(a, p, t, r)).$$

Proposition 4.6.6. *Let* $Inst(D) = [(Breaks(a, p, t) \longleftarrow \Delta_1(a, p, t)) \wedge (BreaksTo(a, p, t, r) \longleftarrow \Delta_2(a, p, t, r))]$. *If none of* $\Delta_1(a, p, t)$ *and* $\Delta_2(a, p, t, r)$ *mention Breaks or BreaksTo then CIRC[Inst(D)* $\wedge(EC11) \wedge(EC12) \wedge(EC17p)$*; Breaks; BreaksTo] entails*

$$\begin{aligned} Breaks(a, p, t) \longleftrightarrow [&\exists p_1(p = \delta(p_1) \wedge Breaks(a, p_1, t)) \vee \Delta_1(a, p, t) \\ &\vee \exists r(\Delta_2(a, p, t, r) \vee BreaksPartBy(a, p, t, r))]. \end{aligned}$$

Proof. $[(Breaks(a, p, t) \longleftarrow \Delta_1(a, p, t)) \wedge (Breaks(a, p, t) \longleftarrow BreaksPartBy(a, p, t, r))] \equiv [(Breaks(a, p, t) \longleftarrow \Delta_1'(a, p, t))]$, where $\Delta_1'(a, p, t) = \Delta_1(a, p, t) \vee \exists r.BreaksPartBy(a, p, t, r)$. Result follows from Proposition 1 from (Miller & Shanahan, 1996) because $\Delta_1'(a, p, t)$ and $\Delta_2(a,\ p,\ t,\ r)$ do not mention $Breaks$ or $BreaksTo$. □

Proposition 4.6.7. *Let* $Inst_P(D) = [(BreaksPartBy(a, p, t, r) \longleftarrow \Delta(a, p, t, r))]$. *If* $\Delta(a, p, t, r)$ *does not mention BreaksPartBy then*

$$CIRC[Inst_P(D); BreaksPartBy] \equiv Comp(Inst_P(D)).$$

Proof. Follows directly from Proposition 2 from (Lifschitz, 1994). □

We next define a dependency relation between additive parameters, and show that the CIRC^A transformation of $Con_P(D)$ is equivalent to the completion when there is no cyclical dependency.

4.6.3 Dependencies between Additive Parameters

Function symbols with the range in $\mathcal{P}$ (the parameter sort) are referred to as *parameter types*. If $Pt(\mathbf{x})$ is a parameter, where $\mathbf{x}$ is a tuple of zero or more variables, then Pt is a parameter type. And by extension, $\delta \circ Pt$, where $\circ$ denotes composition of functions, and higher derivatives, are also parameter types. Further, if $Pt(\mathbf{x})$ is an additive parameter then Pt is referred to as an additive parameter type. Parameter types that are not additive are referred to as non-additive parameter types.

Non-additive parameter types Pt_1 and Pt_2 are said to be mutually dependent, denoted by Pt_1 $mut{-}depends$ Pt_2, if they are in a (mathematical) functional relationship in some axiom of $Con_P(D)$. For example, Fr $mut{-}depends$ CKF (and vice-versa) by Axiom (D15). The $mut{-}depends$ relationship is symmetric (and transitive).

Parameter type Pt_1 depends on Pt_2, denoted by Pt_1 $dependsOn$ Pt_2, if any of the following conditions are satisfied, for some axiom $Ax \in Con(D) \wedge Con_P(D)$ that includes Pt_1 and Pt_2 and defines an additive or the non-additive value of Pt_1 given by some expression E, that is the consequence is of the form $Value(Pt_1(\mathbf{x}), t) = E$ or $PartValue(Pt_1(\mathbf{x}), t, px, E)$.

a) $Ax \in Con_P(D)$ and Pt_2 appears in E. NHF $dependsOn$ $NAHF$ by Axiom (D23), for example.

b) $Ax \in Con_P(D)$ and Pt_2 appears in the antecedent.

c) $Ax \in Con(D)$ and Pt_2 is either an additive parameter type or Pt_2 is used in a constraint in the antecedent. Fr $dependsOn$ $FbBA$ and Vel by Axiom (D15), for example.

Direct dependencies are determined by the conditions above, and transitive dependencies are inferred from them and $mut{-}depends$ relations using $dependsOn \circ mut{-}depends_{tc} \circ dependsOn$, where $mut{-}depends_{tc}$ denotes the transitive closure

of *mut−depends* relation minus tuples in *dependsOn* relation, $\bigcup_{n=0}^{\infty}(mut{-}depends \setminus dependsOn)^n$.

Intuitively, if Pt_1 *dependsOn* Pt_2, then Pt_2 potentially influences the (additive) value of Pt_1, at some time, either by contributing to the (additive) value of Pt_1 or because the (additive) value of Pt_1 is conditional on a constraint on its value.

The following direct dependencies involving additive parameter types can be inferred for the multiple blocks example. (a) *Acc* on NHF by Axiom (D18), (b) NHF on $NAHF$ by Axiom (D23), and on Fr by Axiom (D24), (c) Fr on $FbBA$ (and Vel) by Axioms (D12)–(D16), and (d) $FbBA$ on $NAVF$ (and Wt) by Axioms (D21)–(D22).

Since, by Axiom (37), $UNA[NAHF, NAVF, FbBA, NHF]$ is valid, the additive parameter types have no cyclical dependencies.

Notation 4.6.4. $PVA(D) = T_{ESI}[Con_P(D)]$.[13] Ω_D denotes the set of additive parameter types in $PVA(D)$.

Definition 4.6.5. When additive parameter types have no cyclical dependencies, then Ω_D can be partitioned into $\{\Omega_1, \ldots, \Omega_m\}$ for some $m \geq 1$ such that if $g \in \Omega_i$ depends on $h \in \Omega_j$ then $i > j$. Such a partitioning of Ω_D is referred to as an *AP-partition.*[14]

Definition 4.6.6. If for every $PartValue$ predicates in $PVA(D)$, the first term is of the form $Pt(\mathbf{x})$, for some parameter type $Pt \in \Omega_D$, then $PVA(D)$ is said to *always have explicit parameter type.*

For example, $\Omega_{D_M} = \{NAHF, NAVF, FbBA, NHF\}$ and it can be partitioned into $\{\{NAHF,\ NAVF\},\ \{FbBA\},\ \{NHF\}\}$ based on the dependencies enumerated earlier, which were acyclic. Also, $PVA(D_M)$ always has explicit parameter type.

Definition 4.6.7. If $PVA(D)$ always has explicit parameter type, additive parameter types have no cyclic dependencies, then given an AP-partition $\{\Omega_1, \ldots, \Omega_m\}$,

[13] PVA is short for 'partial value axioms'

[14] AP for **A**dditive **P**arameter types.

$PVA(D)$ can be partitioned into $PVA_1(D) \wedge \ldots \wedge PVA_m(D)$ such that axioms with first term of the consequent as $Pt(\mathbf{x})$ are in $PVA_i(D)$ if $Pt \in \Omega_i$. Such a partition is referred to as a *PVA-partition*. We assume $PVA_0(D)$ is $\emptyset$, that is all additive parameters appear in the first term of some $PartValue$ predicate of a consequent.

4.6.3.1 Acyclic Dependencies and CIRC^A Transformation

When additive parameters have acyclic dependencies, CIRC^A transformation of $PVA(D)$ is equivalent to the predicate completion (Lemma 4.6.8).

Notation 4.6.8. We assume that $PVA(D)$ always has explicit parameter type, predicates in the antecedents with predicate symbol $PartValue$ occur in an aggregate expression or not inside a negated formula, additive parameter types have no cyclical dependencies, $\{\Omega_1, \ldots, \Omega_m\}$ is an AP-partition of Ω_D, and $PVA_1(D) \wedge \ldots \wedge PVA_m(D)$ is a PVA-partition. As assumed earlier, $Con_P(D)$ does not contain aggregate expressions.

Lemma 4.6.8. $CIRC^A[PVA(D); PartValue] \equiv \bigwedge_{j=1}^{m} Comp(PVA_j(D))$

Proof. $PartValue$ is the sole aggregate predicate in $Con_P(D)$ and if there are no cyclical dependencies between parameter types, then $Con_P(D)$ and $T_{ESI,0}[Con_P(D)]$ are 1st term hierarchical in $PartValue$. Further, given the restriction on $PartValue$ predicates in the antecedents, $Con_P(D)$ and $T_{ESI,0}[Con_P(D)]$ are also 1st term positive hierarchical in $PartValue$. $PVA(D)$, that is $T_{ESI}[Con_P(D)]$, is a DD-derivative of $T_{ESI,0}[Con_P(D)]$ with the $Value$ function as the $Default$ function. The result follows from Lemma 3.4.2 and Proposition 3.4.9. □

The predicate completion of $PVA(D)$ does not interfere with ESI-reformulation and Lemma 4.6.9 states that the ESI-reformulation of $Con_P(D)$ can be reversed without any loss.

Lemma 4.6.9. $Comp(T_{ESI}[Con_P(D)]) \wedge (EC18p) \equiv Comp(Con_P(D)) \wedge (EC18p)$

Proof. According to the ESI-reformulation, variable z equals the summation, $\#sum\ \langle r, px.PartValue(p,\ t,\ px,\ r)\rangle$, when additive parameter p is subject to additive effects, or $Value(p,t)$ otherwise. By Axiom (EC18p), $Value(p,t)$ equals the summa-

tion when p is subject to additive effects. Since $Value$ is a function, variable z can be replaced by $Value(p,t)$ in $Comp(PVA(D))$. □

Lemmas 4.6.8 and 4.6.9 can be combined to give Proposition 4.6.10, with which we can show that CIRC^A transformation of $Con_P(D_M)$ is equivalent to the predicate completion (Proposition 4.6.11).

Proposition 4.6.10. $CIRC^A[T_{ESI}[Con_P(D)]; PartValue] \equiv Comp(Con_P(D)) \wedge (EC18p)$

Proposition 4.6.11. $CIRC^A[T_{ESI}[Con_P(D_M)]; PartValue] \equiv Comp(Con_P(D_M)) \wedge (EC18p)$

Proof. As noted earlier, $PVA(D_M)$ always has explicit parameter type and the additive parameter types have only acyclic dependencies. Result follows from Proposition 4.6.10. □

4.7 Discussion

We remark on the following aspects of the extensions: (a) solution to the frame problem, (b) CIRC^A transformation, (c) ESI-reformulation, and (d) denoting parameters by a sort.

Shanahan (1997) stated that "One of the most curious things about the frame problem as a subject of enquiry is that, like the mind-body problem in philosophy, there's no universal agreement as to what would constitute a solution.", which we believe remains true. He goes on to say that "Yet at the same time, the tools that are brought to bear on the problem are as precise and rigorous as those that are brought to bear on any great unsolved problems in mathematics". The key to the solution to the frame problem for the Event Calculus is the splitting of the theory into different parts that are circumscribed separately. Separation ensures that the domain theory and narrative descriptions are circumscribed independently and temporal projection does not interfere with the minimization (of the extension of predicates). The $CIRC_{CECA}(D)$ policy continues with that tradition, and discrete additive effects and continuous additive effects are circumscribed independently. Circumscription,

used for encoding the relevant necessary conditions in the Event Calculus, cannot be used to encode the necessary conditions for additive effects in the extended Event Calculus, because additive effects are summed using aggregate expressions. Circumscription was originally defined for first-order logic without aggregate expressions. We extend that transformation to first-order logic with aggregate expressions, where the transformation for aggregate expressions is different (from that we would get if circumscription transformation was also applied to aggregate expressions). The new transformation, CIRC^A transformation, permits us to encode the necessary conditions for active additive effects in the extended Event Calculus (wherein circular justifications are not allowed).

While we have had to introduce a novel nonmonotonic reasoning technique, CIRC^A transformation is generally applicable. It is also not tailored specifically for the problem of reasoning about actions or additive effects. CIRC^A transformation can be used in applications where (some) aggregate formulas in first-order logic are to be interpreted as constraints.

ESI-reformulation (Definition 4.2.1) is required for a couple of reasons. Firstly, the transformation helps in the separation of the value of an additive parameter determined by $Con(D)$ from that which is determined by $Con_P(D)$. The value of the parameter is expressed directly in terms of the function $Value$ in the former case and by an aggregate summation in the latter case. By way of this separation, the extent of $Value$ can be considered to be fixed when the extent of $PartValue$ in $Con_P(D)$ is minimized. Secondly, the ESI-reformulation helps rewriting axioms in $Con_P(D)$ such that Axiom (EC18p) which infers the value of a function (as opposed to a general relation), is not required by itself. This simplifies the analysis for circumscription of $Con_P(D)$. The predicate completion result of Lemma 4.6.8 is very advantageous. By Lemma 4.6.9, the completion of $T_{ESI}[Con_P(D)]$ is equivalent to the completion of $Con_P(D)$ in conjunction with Axiom (EC18p). That is, ESI-reformulation can be reversed in those conditions.

Some of the complexities in proofs for Proposition 3.4.9 and Lemma 4.6.8 in comparison with that for Propositions 3.4.7 and 3.4.8 are due to denotation of parameters as terms instead of predicates. Historically, fluents and parameters have

been denoted by sorts, that is, they have been reified, in Event Calculus (Shanahan, 1999a; Miller & Shanahan, 1996). Reification (Lifschitz, Morgenstern, & Plaisted, 2008) makes the language more expressive. An advantage of reification, for example, is that the domain-independent Axioms (EC16p)–(EC18p) were defined for all possible parameters without resorting to any second-order logic.

4.8 Summary

In Section 4.1, we discussed the limitations of explicit enumeration of combined effects of additive effects and of using recursion or iteration for computing the combined effects using an example of a tank filled by multiple moving pipes. We showed that descriptions cannot be general or additive-elaboration tolerant, and tend to be long and verbose. We introduced the extensions to the Event Calculus in Section 4, illustrated with help of the tank with multiple pipes example, and circumscription policy for the extended Event Calculus. We discussed the limitations of using circumscription for nonmonotonic reasoning over continuous additive effects using another example involving a single block subject to multiple horizontal forces in Section 4.3. Circumscription yielded models which violated the physical laws of causality. In Section 4.4 we discussed descriptions of indirect discrete additive effects in the extended Event Calculus. Cicumscription is not sufficient for discrete additive effects either in presence of indirect effects. We generalized the previous single block example to multiple blocks subject to multiple horizontal and vertical forces, and demonstrated the facility of the extended Event Calculus, in Section 4.5. In Section 4.6, we defined the default reasoning strategy for the extended Event Calculus, discussed its correctness, and defined the conditions under which the relevant CIRC^A transformations are equivalent to the completion.

CHAPTER 5
CONSTRUCTING MODELS OF CONTINUOUS-TIME EVENT CALCULUS

Recall that time-varying properties are differentiated into fluents and quantities (or parameters) in the Event Calculus. Frame fluents are persistent by default and quantities are continuous by default. Any change in the state of fluents and any break in the continuity of a quantity is caused by action occurrences. We refer to such time points at which there is a discontinuous change in the state of fluents or in the values of quantities as *landmark points*, or just landmarks, in the spirit of landmark points in Qualitative Reasoning (Forbus, 1984; Kuipers, 2001).

We first reformulate Event Calculus axioms as transitions over landmarks, and introduce new relations and functions relevant to performing reasoning or derivations in Section 5.1. We cover different aspects by which model construction, or in general any form of reasoning, may become impractical, and introduce axiomatized restrictions to the Event Calculus to detect and avoid those aspects in Section 5.2. Also in Section 5.2, we discuss that the default assumption that quantities are continuous by default while necessary is not sufficient, and the default assumption that quantities are constant by default if continuous also seems reasonable. We introduce new sets of axioms, syntactically derived from user-defined descriptions, that facilitate in reasoning, in particular allowing for separation of logical reasoning and solving of ordinary differential and other equations in Section 5.3. This separation allows us to use off the shelf reasoners for logic reasoning, for example.

Inspired by Kim, Lee, and Palla (2009) we reformulate Event Calculus theories as answer-set programs, and use answer-set solvers for logical reasoning. We extend Kim, Lee, and Palla's (2009) results for their version of Event Calculus which does not support ODE-based description of continuous-changes to the Event Calculus which does support that, and its extension for distinct descriptions of additive

Portions of this chapter to appear in: Khandelwal, A., & Fox, P. (2013). Constructing Models for Continuous-Time and Continuous-Change Event Calculus. Artificial Intelligence Journal (AIJ).

effects, in Section 5.4. In particular, Event Calculus theories are reformulated as HEX-programs (Eiter et al., 2005) which extend answer-set programs with external computations, required for real number arithmetic and comparisons. We finally describe the process of constructing models for given, numerical, and finite domains, based on an initial state and narratives of exogenous action occurrences, in Section 5.5. A domain is said to be *given*, *numerical*, and *finite* if (a) all the fluents and quantities are given and finite, (b) constant terms denoting real values, including times of occurrences, are numbers, not constant symbols, and (c) whenever there is a break in continuity of some quantities, their values must be determinable from known discontinuous changes in some of those quantities and the axiomatically described equations.

We remark on our implementation and other aspects of the model construction in Section 5.6. Finally, we present models generated by our implementation for some example domains in Section 5.7.

5.1 Reformulating Event Calculus and New Terminologies

Axioms (EC1)–(EC4) define the states of frame (or primary) fluents at arbitrary times. The states of the frame fluents change only immediately after an action occurrence, after which they remain unchanged at least until the next action occurrence. So, we can reformulate the semantics of Event Calculus in terms of transitions over time-points where actions occur. And, it is sufficient to reason about states of the frame fluents at the times of action occurrences, from which the states at other times can be readily inferred. That way only the instances of time at which some actions occur, from an everywhere dense domain (non-negative real numbers) of instances of time, are relevant to the reasoning[15]. We discuss determination of next instance of time at which some actions occur, for a given time instance, in Section 5.3.3.

$$t < t_1 < Next(t) \longrightarrow [HoldsAt(f, t_1) \longleftrightarrow HoldsAt(f, Next(t))] \tag{EC30}$$

$$\exists p.\neg Continuous(p, t) \longrightarrow \exists a.Happens(a, t) \tag{EC31}$$

[15]Note that in our version of Event Calculus actions occur instantaneously

The two axioms given above can be inferred from the semantics of Event Calculus. Axiom (EC30) states that state of the fluent is unchanged in the interval $(t, Next(t)]$, and is derivable from the Event Calculus as shown in Proposition 5.1.1.

Proposition 5.1.1. $[(EC1) \wedge \ldots \wedge (EC6) \wedge (EC13) \wedge (EC14) \wedge (EC15)] \longrightarrow (EC30)$

Axiom (EC31) follows trivially from (EC8). It states that if at any point some quantity is not continuous then some action must have occurred at that time. That is, if there is a discontinuous change in the value of some quantity at some time t it cannot be the case that no action occurs at time t. An unexplained change in continuity (unexplained because of no causal action occurrence) is an important cause for inconsistency and should be detected by any reasoner.

Proposition 5.1.2. $(EC8) \longrightarrow (EC31)$

We reformulate Event Calculus domain-independent axioms to be interpreted according to (EC30), that is in terms of transitions over landmarks. The reformulated semantics is given below by Axioms (ECP1)–(ECP42), which also include the semantics for new relations and functions relevant in deductions, and explicit assertion of some constraints which are implicit in the original semantics.

$$t < Next(t) \tag{ECP1}$$

$$[t < t_1 \wedge t_1 < Next(t)] \longrightarrow \neg Happens(a, t_1) \tag{ECP2}$$

$$[Happens(a_1, t_1) \wedge t < t_1] \longrightarrow \exists a. Happens(a, Next(t)) \tag{ECP3}$$

$$\neg \exists a, t_1. (t < t_1 \wedge Happens(a, t_1)) \longrightarrow Next(t) = MaxTime \tag{ECP4}$$

$$t <= MaxTime \tag{ECP5}$$

$$[t < t_1 < Next(t) \wedge Primary(f)] \longrightarrow \tag{ECP6}$$
$$[HoldsAt(f, t_1) \longleftrightarrow HoldsAt(f, Next(t))]$$

$$Primary(f) \equiv^{def} [InitializedTrue(f) \vee InitializedFalse(f)] \tag{ECP7}$$

$$Secondary(f) \longrightarrow \neg Primary(f) \tag{ECP8}$$

$$HoldsAt(f, 0) \longleftarrow InitializedTrue(f) \tag{ECP9}$$

$$\neg HoldsAt(f, 0) \longleftarrow InitializedFalse(f) \tag{ECP10}$$

$$HoldsAt(f, Next(t)) \tag{ECP11}$$
$$\longleftarrow [Primary(f) \wedge HoldsAt(f,t) \wedge \neg ClippedAt(f,t)]$$
$$\neg HoldsAt(f, Next(t)) \tag{ECP12}$$
$$\longleftarrow [Primary(f) \wedge \neg Holds(f,t) \wedge \neg DeclippedAt(f,t)]$$
$$HoldsAt(f, Next(t)) \longleftarrow [Happens(a,t) \wedge Initiates(a,f,t)] \tag{ECP13}$$
$$\neg HoldsAt(f, Next(t)) \longleftarrow [Happens(a,t) \wedge Terminates(a,f,t)] \tag{ECP14}$$
$$ClippedAt(f,t) \equiv^{def} \exists a[Happens(a,t) \wedge Terminates(a,f,t)] \tag{ECP15}$$
$$DeclippedAt(f,t) \equiv^{def} \exists a[Happens(a,t) \wedge Initiates(a,f,t)] \tag{ECP16}$$
$$Started(f,t) \equiv^{def} \tag{ECP17}$$
$$[HoldsAt(f,t) \vee \exists a(Happens(a,t) \wedge Initiates(a,f,t))]$$
$$Stopped(f,t) \equiv^{def} \tag{ECP18}$$
$$[\neg HoldsAt(f,t) \vee \exists a(Happens(a,t) \wedge Terminates(a,f,t))]$$
$$Initiated(f,t) \equiv^{def} \tag{ECP19}$$
$$[Started(f,t) \wedge \neg\exists a(Happens(a,t) \wedge Terminates(a,f,t))]$$
$$Terminated(f,t) \equiv^{def} \tag{ECP20}$$
$$[Stopped(f,t) \wedge \neg\exists a(Happens(a,t) \wedge Initiates(a,f,t))]$$
$$InitialValue(p,v) \longrightarrow Value(p,0) = v \tag{ECP21}$$
$$LeftContinuous(p,t) \tag{ECP22}$$
$$\neg[Happens(a,t) \wedge Breaks(a,p,t)] \longrightarrow Continuous(p,t) \tag{ECP23}$$
$$\neg[Happens(a,t) \wedge Breaks(a,\delta(p),t)] \longrightarrow Differentiable(p,t) \tag{ECP24}$$
$$[BreaksTo(a,p,t,r) \wedge Happens(a,t)] \longrightarrow RightLimit(p,t,r) \tag{ECP25}$$
$$[\exists a,r.(BreaksPartBy(a,p,t,r) \wedge Happens(a,t)) \wedge s = Value(p,t)+ \tag{ECP26}$$
$$\#sum\langle r,a.(BreaksPartBy(a,p,t,r) \wedge Happens(a,t))\rangle] \longrightarrow RightLimit(p,t,s)$$
$$BreaksTo(a,p,t,r) \longrightarrow Breaks(a,p,t) \tag{ECP27}$$
$$BreaksPartBy(a,p,t,r) \longrightarrow Breaks(a,p,t) \tag{ECP28}$$
$$Breaks(a,p,t) \longrightarrow Breaks(a,\delta(p),t) \tag{ECP29}$$
$$[RightLimit(p,t,r_1) \wedge RightLimit(p,t,r_2)] \longrightarrow r_1 = r_2 \tag{ECP30}$$

$$Continuous(p,t) \longrightarrow RightLimit(p,t,Value(p,t)) \quad \text{(ECP31)}$$

$$[\exists r, px.PartValue(p,t,px,r) \wedge s = \#sum\langle r, px.PartValue(p,t,px,r)\rangle] \longrightarrow Value(p,t) = s \quad \text{(ECP32)}$$

$$[\#count\langle r, px.PartRightLimit(p,t,px,r)\rangle \neq 0 \wedge s = \#sum\langle r, px.PartRightLimit(p,t,px,r)\rangle] \longrightarrow RightLimit(p,t,s) \quad \text{(ECP33)}$$

$$t < Successor(t) \quad \text{(ECP34)}$$

$$t < t_1 \longrightarrow Successor(t) < t_1 \quad \text{(ECP35)}$$

$$t < PotNext(t) \quad \text{(ECP36)}$$

$$PotNext(t) \leq Next(t) \quad \text{(ECP37)}$$

$$\exists a.Happens(a, PotNext(t)) \longrightarrow Next(t) = PotNext(t) \quad \text{(ECP38)}$$

$$p_2 = \delta(p_1) \longrightarrow Derivative(p_2, 1, p_1) \quad \text{(ECP39)}$$

$$[p_3 = \delta(p_2) \wedge p_2 = \delta(p_1) \wedge Derivative(p_2, n, p_1)] \longrightarrow Derivative(p_3, n+1, p_1) \quad \text{(ECP40)}$$

$$[EqnHoldsAt(eq, p_1, t) \wedge \neg\exists p', n.Derivative(p_1, n, p')] \longrightarrow EqnHoldsFor(p_1, t) \quad \text{(ECP41)}$$

$$[EqnHoldsAt(eq, p_2, t) \wedge Derivative(p_2, n, p_1) \wedge \neg\exists p', n'.Derivative(p_1, n', p')] \longrightarrow EqnHoldsFor(p_1, t) \quad \text{(ECP42)}$$

$Next(t)$ after the last occurrence of any action is not tightly defined by $(EC13) \wedge (EC14) \wedge (EC15)$. We introduce the notion of maximum time, denoted by $MaxTime$ constant, which in theory is ∞ but in practice could be any finite time for which the last observation is required. An *observation* refers to the state of a fluent at a given time or the value of a quantity at a given time. Axiom (ECP4) tightly defines $Next(t)$ for times which have no later occurrences.

We introduce the following new relations which can be used in domain descriptions: $Secondary$ and $InitialValue$. $Secondary(f)$ denotes that fluent f is secondary and not subject to default persistence, and may be used to explicitly specify that a given fluent is secondary, rather than implicitly by way of not specifying its initial state. In this version of Event Calculus, there is a strict division

between primary fluents and secondary fluents (Axiom (ECP8)). $InitialValue(p, v)$ denotes that p has an initial value v (Axiom (ECP20)).

Axiom (ECP30) explicitly asserts that *RightLimit* is unique at any given time, which follows from its mathematical definition. Axiom (ECP31) explicitly asserts that if a quantity is continuous at a given time then its right limit value at the time is equal to its value at the time (and its left limit value at the time).

We introduce the following new relations: $PartRightLimit$, $Derivative$, $EqnHoldsAt$, $EqnHoldsFor$, and $UnknReqdRightLimit$, and the following functions: $Successor$ and $PotNext$, which are helpful in derivations.

$PartRightLimit(P, T, PX, R)$ denotes that at time T, quantity P has an additive right limit of R, uniquely identified by PX. After a break in continuity of a quantity (for example, after some water is scooped from a tank), while the quantity is subject to continuous additive effects (for example, water flowing out of the outlets at the bottom of the tank), $PartRightLimit$s are used to determine the additive values of the quantity (the rate of outflow from the tank) immediately after the break occurs. The right limit values are deduced from $PartRightLimit$s using Axiom (ECP33).

$Derivative(P_2, N, P_1)$ denotes that quantity P_2 is the N-th order derivative of p_1, $N > 0$, and is defined by Axioms (ECP39)–(ECP40).

We introduce a new sort for constants to refer to the equations, which we will discuss in Section 5.3.2. For now EQ is a constant and eq is a variable of the aforementioned sort. $EqnHoldsAt(EQ, P, T)$ denotes that equation EQ that gives the value for quantity P is active at time T. A quantity is a *base quantity* if it is not a derivative of other quantity. $EqnHoldsFor(P, T)$ denotes that some equation is active for a base quantity P or its derivatives, and is defined by Axioms (ECP41)–(ECP42). $UnknReqdRightLimit(T)$ denotes that the value of some quantity is required immediately after time T for determining the value of another quantity or truthness of the antecedent of an axiom but the right limit value at time T is unknown (and thereby a value from infinitely many possibilities would have to be picked).

$Successor : \mathcal{T} \mapsto \mathcal{T}$. $Successor(T)$ is used to denote a time that immediately

follows T. It is larger than T but smaller than every other time greater than T, as expressed in Axioms (ECP34)–(ECP35). Note that for any given time T_1, T_1 is different from $Successor(t)$ for any time t.

$PotNext : \mathcal{T} \mapsto \mathcal{T}$. $PotNext(T)$ is used to denote a time after T, that can potentially be a point of next action occurrence. Some basic characteristics of this function are given in Axioms (ECP36)–(ECP38), and its relevance is discussed in Section 5.3.3.

The circumscription policy under the new semantics, $CIRC_{CECP}(D)$, which follows from $CIRC_{CECA}(D)$, is given below.

$$
\begin{aligned}
CIRC_{CECP}(D) = {} & CIRC[Nar(D); Happens, InitialisedTrue, InitialisedFalse, \\
& \qquad\qquad InitialValue, Secondary] \\
& \wedge CIRC[Eff(D); Initiates, Terminates] \\
& \wedge CIRC[Inst(D) \wedge (ECP27) \wedge (ECP28) \wedge (ECP29); Breaks; BreaksTo] \\
& \wedge CIRC[Inst_P(D); BreaksPartBy] \\
& \wedge CIRC^A[T_{ESI}[Con_P(D)]; PartValue] \\
& \wedge Con(D) \wedge Cnst(D) \wedge Rem(D) \wedge Una(D) \wedge ECP,
\end{aligned}
$$

where $ECP = (ECP1) \wedge \ldots \wedge (ECP26) \wedge (ECP30) \wedge \ldots \wedge (ECP33) \wedge (ECP34) \wedge \ldots \wedge (ECP42)$.

5.2 Some Practical Considerations

Event Calculus is first-order logic based and the domain of time (non-negative reals) in continuous-time Event Calculus is everywhere dense. With those two features, reasoning about Event Calculus descriptions, particularly model construction, can become impractical under different scenarios. For instance, if non-frame fluents are not *tightly* defined, they can take arbitrary values even at time instances where no actions occur. Or, if sufficiently many initial values are not known to obtain a particular solution for a given ODE, the trajectory of change cannot be inferred deterministically. Or, due to incomplete specification an action may happen or trigger

repetitively or a description may entail a vicious cycle of triggered actions. Or, if an equation of trajectory depends on the value of a higher-order derivative of some quantity which can only be known by differentiating (known) equations for lower derivatives of the quantity, then even though it does not complicate reasoning, it complicates separation of logic reasoning and equations solving. Finally, events may be triggered with infinitesimal delays, which again does not complicate reasoning, but it complicates determination of time instance of the next action occurrences by solving of equations (See Section 5.3.3 for equations for determining the time instance of the next action occurrences).

We discuss the afore-mentioned scenarios with examples. Except in the last two scenarios, we introduce axiomatized restrictions, listed below, to detect and avoid those scenarios. In other cases, we discuss alternative formulations with the same or similar effects.

$$[t < t_1 < Next(t) \wedge Secondary(f)] \longrightarrow [HoldsAt(f, t_1) \longleftrightarrow HoldsAt(f, Next(t))] \quad \text{(ECPR1)}$$

$$[Continuous(p, t) \wedge \neg\exists p', n.Derivative(p, n, p') \wedge \neg EqnHoldsFor(p, t)] \longrightarrow Value(\delta(p), t) = 0 \quad \text{(ECPR2)}$$

$$\exists v.InitialValue(p, v) \quad \text{(ECPR3)}$$

$$\exists eq.EqnHoldsAt(eq, p_2, t) \wedge Derivative(p_2, n, p_1) \longrightarrow \exists r.RightLimit(p_1, t, r) \quad \text{(ECPR4)}$$

$$\neg\exists UnknReqdRightLimit(t) \quad \text{(ECPR5)}$$

$$\exists a.Happens(a, t) \longrightarrow \neg\exists a.Happens(a, Successor(t)) \quad \text{(ECPR6)}$$

Observation 5.2.1. While the persistence of frame fluents is guaranteed between any two consecutive occurrences the same is not true in general for non-frame fluents. Consider a simple domain description with a single fluent F given below.

$$D_1 = Secondary(F)$$

$CIRC_{CECP}(D_1)$ has a model where for some T, $HoldsAt(F, t)$ is true for $t = T$ but

does not hold immediately after T.

Observation 5.2.1 shows that the state of secondary fluents can change arbitrarily over an everywhere dense time if the secondary fluent is not completely defined. Intuitively, the state of secondary fluents is *not completely defined* if values of some of them can be changed at the immediately next instance without affecting the continuities of quantities. (The states of primary fluents are tightly defined by Axioms (ECP6) and (ECP9)–(ECP14)). But it is difficult to check that secondary fluents are completely defined over an interval over real line. Therefore, we choose to restrict the definition of secondary (non-frame) fluents to those fluents whose values are not directly changed by any action but whose values change only at those time points where some actions occur, through Axiom (ECPR1). So state change of a secondary fluent is not directly associated with occurrences of specific actions, but is now explicitly associated with occurrence of some action.

Observation 5.2.2. Assume a time interval (T_1, T_2), $T_1 < T_2$, and a base quantity Q. Let no action occurs in that interval, no equation for the value of Q or its derivatives is known to hold in the interval, Q and its derivatives are continuous at T_1, and the values of Q and its derivatives are known at time T_1. As an example consider a relatively general domain description D_2:

$$D_2 = [InitialValue(Q, R_1) \wedge \ldots \wedge InitialValue(\delta^n(Q), R_n)$$
$$\wedge (\neg\exists r.InitalValue(q, r) \longrightarrow Value(q, t) = 0)].$$

$CIRC_{CECP}(D_2)$ has at least two models with $Value(Q, T)$ given by two different polynomials of time. In the first model the value of Q is given by: $Value(Q, t) = \frac{R_n}{n!}t^n + \ldots + \frac{R_0}{0!}$. In the second model the value of Q is given by: $Value(Q, t) = t^{n+1} + \frac{R_n}{n!}t^n + \ldots + \frac{R_0}{0!}t^0$. Note that both the polynomials are everywhere continuous and so are all their derivatives.

Observation 5.2.2 shows that if no equation is known for a given quantity at the times where it is continuous allows for the value of the quantity to change continuously beyond that in many different ways. That is, there is no unique path of change in the value in such cases. One way to avoid such a non-determinism

would be to require that some equation for any base quantity or its derivatives must be known at all times that the base quantity is continuous.

$$[Continuous(p,t) \wedge \neg\exists p', n.Derivative(p,n,p')] \longrightarrow EqnHoldsFor(p,t)$$

However that would be cumbersome for the users. Observation 5.2.2 shows that quantities are continuous by default is a necessary default assumption but not sufficient, and quantities are constant by default if continuous is also needed. Thus we introduce Axiom (ECPR2) according to which a base parameter is constant if it is not known to be subject to any continuous change. (If any of the derivatives of a base quantity has a non-zero right limit after an action occurrence and no equation for change is known for the base quantity immediately after the occurrence an inconsistency is detected.) It is not clear to us if the new default assumption can be axiomatized using just the relations and functions of Miller and Shanahan's (1996) formalism.

Observation 5.2.3. Assume a time interval (T_1, T_2), $T_1 < T_2$, and a base quantity Q such that its derivatives of order less than k (here, k can be zero) are continuous and k-th and higher derivatives can be potentially discontinuous. Let the equation for n-th order, $n > k$, derivative of Q is known to hold in the interval, but the right limit at time T_1 is not known for at-least one of o-th derivative, $k \leq o < n$. Consider a simple example domain described below.

$$D_3 = [InitialValue(Q, R_1) \wedge InitialValue(\delta(Q), R_2) \wedge InitialValue(\delta^2(Q), R_3)$$
$$\wedge Happens(A, 0) \wedge Breaks(A, \delta(Q), T) \wedge Value(\delta^2(Q), t) = 2]$$

$CIRC_{CECP}(D_3)$ has many models. We know of a general solution for value of Q: $Value(Q,t) = t^2 + c \times t + R_1$, for some c. That is, $Value(Q,t)$ changes according to $t^2 + c \times t + R_1$ in all the models, but since c is not fixed there are many paths by which $Value(Q,T)$ changes.

Observation 5.2.3 shows that if in an interval the value of a quantity is given by a differential equation of order n, $n > 0$, then for a definitive path of change,

particular values of $n-1$-st and all the lower derivatives (including 0-th derivative) must be known at the beginning of the interval. So we introduce the constraints represented by Axioms (ECPR3)–(ECPR4). Axiom (ECPR3) states that the initial values for all parameters must be given, and Axiom (ECPR4) states that if equation for a quantity is known then right limits of any of its lower derivatives must also be given. These right limits of lower derivatives serve as the initial values required for obtaining a particular solution for the (system of) ODEs active at any given time.

Observation 5.2.4. Even if all the required initial values for given (system of) ODEs is known the values of higher derivatives of quantities involved in the ODEs may not be known without solving the differential equations. And these derivatives may determine other equations that are active. Consider an example domain given below, where the value of Q_3 is dependent on second order derivative of Q_2.

$$\begin{aligned} D_4 =& ([p = V_1 \vee p = \delta(V_1) \vee p = Q_2 \vee p = \delta(Q_2) \vee p = \delta(\delta(Q_2)) \vee p = Q_2 \\ & \qquad \vee p = \delta(Q_2)] \longrightarrow InitialValue(p, 0)) \\ & \wedge Value(\delta(Q_1), T) = 1 \\ & \wedge Value(\delta(Q_2), T) = 2 \times Value(Q_1, T) \\ & \wedge (Value(\delta(\delta(Q_2)), T) > 0 \longrightarrow Value(Q_3, T) = 4) \\ & \wedge (Value(\delta(\delta(Q_2)), T) \leq 0 \longrightarrow Value(Q_3, T) = 0) \\ & \wedge Breaks(A, Q_2, T) \wedge Breaks(A, Q_3, T) \\ & \wedge Happens(A, 0) \end{aligned}$$

The value of the second order derivative of Q_2 is unknown immediately after break in continuity of its value or that of its lower derivatives without solving a first order ODE. An alternative way of formulating the axioms that give the value for Q_3 is to represent the different scenarios when $Value(\delta(\delta(Q_2)), T) > 0$ and $Value(\delta(\delta(Q_2)), T) \leq 0$ by a secondary fluent as shown below.

$$\begin{aligned} & Secondary(F) \\ & \wedge (HoldsAt(F, t) \longrightarrow Value(\delta(\delta(Q_2)), T) > 0) \end{aligned}$$

$$\wedge\ (\neg HoldsAt(F,t) \longrightarrow Value(\delta(\delta(Q_2)),T) \leq 0)$$
$$\wedge\ HoldsAt(F,t) \longrightarrow Value(Q_3,T) = 4$$
$$\wedge\ \neg HoldsAt(F,t) \longrightarrow Value(Q_3,T) = 0$$

This way both the models are considered when in one F holds and $Value(\delta(\delta(Q_2)),t) > 0$ (and $Value(Q_3,T) = 4$) and in other F does not hold true and $Value(\delta(\delta(Q_2)),t) > 0$ (and $Value(Q_3,t) = 0$). One of the models is rejected after the ODEs have been solved, depending on $Value(\delta(\delta(Q_2)),t)$.

Observation 5.2.4 shows that even if all the required initial values for a particular solution for an ODE are known, there may be other axioms that require values of higher derivatives of the quantities involved in the ODE, which could only be known by solving the equations and calculating the higher order derivatives. Their values may determine equations for other quantities. This is relatively innocuous if such dependencies between values of higher derivatives and equations for quantities are acyclic, as quantities could be stratified and the ODEs in different strata can be solved one after the other. Things get more complicated however when there are cyclic dependencies. To avoid those complexities we impose another restriction stated in Axiom (ECPR5) that any quantity whose value is required at an immediately following time to determine the value and equations for other quantities, etc., its right limit must be known at the given time. Note that the right limit value for a quantity may be unknown at a given time only if it is discontinuous at the given time. The axioms for determining the $UnknReqdRightLimit$ relation are discussed later in Section 5.3.

Axiom (ECPR5) is restrictive but often one or more secondary fluents may be introduced to express dependencies between values of higher derivatives of some quantities and equations for other quantities as shown in Observation 5.2.4.

Observation 5.2.5. Incomplete domain specification (Chapter 4, Mueller, 2006) could lead to repetitive firing of the same action. Consider the example domain given below.

$$D_5 = [Initiates(TurnOn, On, t)$$

$$\wedge\,(HoldsAt(On,t) \longrightarrow Happens(TurnOff,t))$$
$$\wedge\,HoldsAt(On,0)]$$

D_5 is incomplete. $TurnOff$ does not affect the state of On. As a result the action $TurnOff$ happens repetitively.

Observation 5.2.5 discusses repeated firing of events as a result of incomplete domain description.

Observation 5.2.6. A domain description can get into a vicious cycle (Shanahan, 1999b) where alternate actions happen successively. Consider the example domain given below.

$$\begin{aligned} D_6 = [&Initiates(TurnOn,On,t)\\ &\wedge Terminates(TurnOff,On,t)\\ &\wedge (HoldsAt(On,t) \longrightarrow Happens(TurnOff,t))\\ &\wedge (\neg HoldsAt(On,t) \longrightarrow Happens(TurnOn,t))\\ &\wedge HoldsAt(On,0) \end{aligned}$$

D_6 has one component such that just when it is turned on, it is triggered to be turned off and vice-versa. So the system keeps turning on and off in a vicious cycle.

Observation 5.2.6 discusses a case of vicious cycle. Domain descriptions such as D_6 could be practically useful, as the vicious cycles may have been unintentionally introduced.

Observation 5.2.7. Events can be triggered with an infinitesimal delay. Consider the example domain given below.

$$\begin{aligned} D_7 = [&Initiates(TurnOn_1,On_1,t)\\ &\wedge Terminates(TurnOff_2,On_2,t)\\ &\wedge (\neg HoldsAt(On_1,t) \longrightarrow Happens(TurnOn_1,t))\\ &\wedge ([HoldsAt(On_1,t) \wedge HoldsAt(On_2,t)] \longrightarrow Happens(TurnOff_2,t)) \end{aligned}$$

$$\wedge \neg HoldsAt(On_1, 0)$$
$$\wedge HoldsAt(On_2, 0)$$

There are two components such that when the first is turned on while the second is already on, the second component is triggered to be turned off. The second component is turned off after an infinitesimal gap of first component being turned on. Such triggered events can also be expressed as causal constraints. For example, the fourth axiom of D_7 can be formulated as:

$$[Started(On_1, t) \wedge Initiated(On_2, t)] \longrightarrow Happens(TurnOff_2, t)$$

In this case, the second component is simultaneously turned off as the first is turned on. That is, there is no gap between those two events.

Observation 5.2.7 discusses a case of events triggered with infinitesimal delay, but without any vicious cycle, which can be also described as causal constraints. When expressed as causal constraints the triggered event occurs without a delay.

To avoid, and sometimes detect, repeated firing and vicious cycles over real time we make the *finite separation assumption*, that is any two event occurrences are required to be finitely separated in time, as asserted by the constraint in Axiom (ECPR6).

5.3 New Sets of Axioms From Given Domain Descriptions

We create the following new sets of axioms:

1. $Con_{RL}(D)$ (from $Con(D)$) and $Con_{PRL}(D)$ (from $Con_P(D)$): To determine right limits particularly at the time points where some quantities are not continuous.

2. $ChkRL(D)$ (from $Occ_P(D)$, $Con(D)$, $Con_P(D)$, $PCnst(D)$, and $FCnst(D)$): To determine if all the right values that are required to completely determine active equations, constraints, etc. are known.

3. $Eqn(D)$ (from $Con(D)$) and $AEqn(D)$ (from $Con_P(D)$): To determine active equations.

4. $CnstEqn(D)$ (from $PCnst(D)$): To determine active mathematical constraints.

5. $ChkCnd(D)$ (from $Occ_P(D)$, $Con(D)$, $Con_P(D)$, $PCnst(D)$, and $FCnst(D)$): To determine the mathematical conditions relevant to computing the next point of change either due to unexpected break in continuity of a parameter or due to an occurrence of an event.

$Inst(D)$, $Inst_P(D)$, or $Eff(D)$ are not used in the derivations of $ChkCnd(D)$ or $ChkRL(D)$ because those axioms are relevant only when there is an action occurrence; check Axioms (ECP13)–(ECP14) and (ECP25)–(ECP26).

Syntax for domain-specific axioms in the Event Calculus are reviewed in 2.2.2. The syntax for axioms about additive effects is given next.

$Inst_P(D)$ consists of axioms of the form:

- $\gamma \longrightarrow BreaksPartBy(a, p, t, r)$.

$Con_P(D)$ consists of axioms of the form:

- $\gamma \longrightarrow PartValue(p, t, px, me)$.

Axioms (ECPR7)–(ECPR8) can be derived from (ECPR6) and (ECPR1). We interpret the right limit value of a quantity at a time point t as its value at $Successor(t)$, as expressed in Axiom (ECPR9).

$$HoldsAt(f, Successor(t)) \longleftrightarrow HoldsAt(f, Next(t)) \quad \text{(ECPR7)}$$

$$HoldsAt(f, PotNext(t)) \longleftrightarrow HoldsAt(f, Next(t)) \quad \text{(ECPR8)}$$

$$RightLimit(p, t, r) \longrightarrow Value(p, Successor(t)) = r \quad \text{(ECPR9)}$$

We discuss $Con_{RL}(D)$, $Con_{PRL}(D)$ and $ChkRL(D)$ axioms in Section 5.3.1, $Eqn(D)$ and $CnstEqn(D)$ in Section 5.3.2, and $ChkCnd(D)$ in Section 5.3.3. Finally, we discuss the new circumscription policy and separation of logical reasoning and solving of equations in Section 5.3.4.

5.3.1 Right Limits After Breaks in Continuities

$Con_{RL}(D)$ is derived from $Con(D)$ axioms, $\gamma \longrightarrow Value(p,t) = me$, by

(a) replacing each $HoldsAt(f,t)$ in γ by $HoldsAt(f, Successor(t))$,

(b) replacing each $Value(p,t)$ in γ or me by a fresh variable v_p unique to each p and conjoining $RightLimit(p,t,v_p)$ to γ, and let me' refer to the transformed me, and

(c) replacing $Value(p,t) = me$ in the consequent by $RightLimit(p,t,me')$.

And, from axioms, $\neg\exists px, r.PartValue(p,t,\ px,r) \longrightarrow Value(p,t) = 0$, by

(a) replacing $PartValue(p,t,px,r)$ by $PartRightLimit(p,t,px,r)$, and

(b) replacing $Value(p,t) = 0$ by $RightLimit(p,t,0)$.

Similarly, $Con_{PRL}(D)$ is derived from $Con_P(D)$ by

(a) replacing each $HoldsAt(f,t)$ in γ by $HoldsAt(f, Successor(t))$,

(b) replacing each $Value(p,t)$ in γ or me by a fresh variable v_p, unique to each p and conjoining $RightLimit(p,t,v_p)$ to γ. Let me' refer to the transformed me, and

(c) replacing $PartValue(p,t,px,me)$ in the consequent by $PartRightLimit(p,t,px,me')$.

Let $RL(D)$= $Con_{RL}(D) \wedge Con_{PRL}(D) \wedge (ECP25) \wedge (ECP26) \wedge (ECP30) \wedge (ECP31) \wedge (ECP33)$. $RL(D)$ denotes the axioms relevant to the computation of right limits. Then the known right limit values for quantities is given by:

$$CIRC^A[RL(D); RightLimit, PartRightLimit].$$

Proposition 5.3.1 shows that any model of $CIRC^A[RL(D); RightLimit, PartRightLimit]$ satisfies the completion of $RL(D)$.

Proposition 5.3.1.

$$CIRC^A[RL(D); RightLimit, PartRightLimit] \longrightarrow Comp(RL(D)).$$

Proof. $RL(D)$ in Clark Normal Form would be of the form:

$$\begin{aligned}&\forall p,t,r(\phi_1(p,t,r) \longrightarrow RightLimit(p,t,r))\\ &\wedge \forall p,t,px,r(\phi_2(p,t,px,r) \longrightarrow PartRightLimit(p,t,px,r)).\end{aligned}$$

Further, let $F(RightLimit, PartRightLimit)= RL(D)$. The only aggregate expressions in $RL(D)$ is due to (ECP33). Therefore, $F^{**}(rightLimit, partRightLimit)$ is derived from F by replacing $RightLimit$ with $rightLimit$, replacing $PartRightLimit$ not in the aggregate expressions by $partRightLimit$ and replacing

$$\begin{aligned}&\#count\langle r', px'.PartRightLimit(p,t,px',r')\rangle \neq 0\\ &\wedge r = \#sum\langle r', px'.PartRightLimit(p,t,px',r')\rangle\end{aligned}$$

from (ECP33) by,

$$\begin{aligned}&(\#count\langle r', px'.PartRightLimit(p,t,px',r')\rangle \neq 0\\ &\quad\wedge \#count\langle r', px'.partRightLimit(p,t,px',r')\rangle \neq 0)\\ &\wedge(r = \#sum\langle r', px'.PartRightLimit(p,t,px',r')\rangle\\ &\quad\wedge r = \#sum\langle r', px'.partRightLimit(p,t,px',r')\rangle).\end{aligned}$$

All occurrences of $RightLimit$s are positive, that is they are not inside any negation, in $\phi_1(p,t,r)$ as well as $\phi_2(p,t,px,r)$. Assume that $RightLimit(P,T,R)$ is true in a model M of $CIRC^A[RL(D); RightLimit, PartRightLimit]$ but $\phi_1(P,T,R)$ is not true. Let $M_1 = M \setminus RightLimit(P,T,R)$ (that is, M_1 consists of all facts of M except $RightLimit(P,T,R)$). Then M_1 is a model of F^{**} since all occurrences of $RightLimit$s are positive in F.

$PartRightLimit$ occurs only in $\phi_1(p,t,r)$ and also only inside the aggregate expressions. Assume that $PartRightLimit(P,T,PX,R')$ and $RightLimit(P,T,R)$

is true in a model M of $CIRC^A[RL(D); RightLimit, PartRightLimit]$ but $\phi_2(P, T, PX, R')$ is not true. Let $M_1 = M \setminus PartRightLimit(P, T, PX, R')$ (that is, M_1 consists of all facts of M except $PartRightLimit(P, T, PX, R')$). Then M_1 is a model of F^{**} since only possible change due to the one less fact is that $\phi_1^{**}(P, T, R)$ is not true. □

Observation 5.3.1. $RL(D)$ imposes directionality. Consider the example domain given below.

$$\begin{aligned} D_8 = [&(Value(P,t) \neq 10 \longrightarrow Value(Q,t) = 20) \\ &\wedge (Value(P,t) = 10 \longrightarrow Value(Q,t) = 30) \\ &\wedge Value(Q,t) = 30] \end{aligned}$$

$CIRC_{CECA}(D_8)$ has a unique model such that $Value(Q,t) = 30 \wedge Value(P,t) = 10 \wedge Q \neq P$ holds true for any time t. Consider $Con_{RL}(D_8) \wedge Con_{PRL}(D_8)$:

$$\begin{aligned} &([RightLimit(P,t,r_P) \wedge r_P \neq 10] \longrightarrow RightLimit(Q,t,20)) \\ &\wedge ([RightLimit(P,t,r_P) \wedge r_P = 10] \longrightarrow RightLimit(Q,t,30)) \\ &\wedge RightLimit(Q,t,30) \end{aligned}$$

$CIRC^A[RL(D_8); RightLimit, PartRightLimit]$ has one model where $RightLimit(Q,t,30)$ holds true for any time t. $RL(D_8)$ not only misses inferring the unique possible value for the right limit of P, it is happily oblivious about the right limit value for P.

Observation 5.3.1 shows that $RL(D)$ imposes directionality while inferring the right limit values for quantities. By design of $Con_{RL}(D) \wedge Con_{PRL}(D)$, axioms in $Con(D) \wedge Con_P(D)$ are interpreted as logic programs with the (bottom-up) fixpoint semantics (Fitting, 2002) in $RL(D)$. This is limiting but advantageous in that right limit values of quantities are either known or derived from known right limit values of other quantities using known equations.

Observation 5.3.2. By the way that $Con_{PRL}(D)$ is derived from $Con_P(D)$ there is a potential that due to some unknown right limits, right limits of quantities are

computed incorrectly, which can trickle down into deducing incorrect right limit values for other quantities too. Consider the simple example domain below.

$$D_9 = [PartValue(P_2, t, PX, Value(P_1, t)) \\ \wedge (\neg\exists px, r.PartValue(P_2, t, px, r) \longrightarrow Value(P_2, t) = 0)]$$

Then $Con_{RL}(D_9)$ and $Con_{PRL}(D_9)$ are derived to be:

$$(RightLimit(P_1, t, r) \longrightarrow PartRightLimit(P_2, t, PX, r)) \\ \wedge (\neg\exists px, r.PartRightLimit(P_2, t, px, r) \longrightarrow RightLimit(P_2, t, 0))$$

Now consider $CIRC^A[RL(D_9); RightLimit, PartRightLimit]$. Since the right limit value for P_1 is not given for any time t, we infer $RightLimit(P_2, t, 0)$ for any t. That is however not the case because right limit for P_2 at time t should equal the right limit for P_1 at time t, which is not given and hence could take any value.

Observation 5.3.2 shows that right limit values of quantities may be computed incorrectly from $RL(D)$ if some required right limits of quantities are unknown. This is especially, and only, the case when $Con_P(D)$ is nonempty. If $Con_P(D)$ is empty then $CIRC^A[RL(D); RightLimit, PartRightLimit] \equiv CIRC[RL(D); RightLimit]$. Since all occurrences of $RightLimit$ in the antecedents (as well as the consequents) of $RL(D)$ are positive, using Corrollary 2 from (Lifschitz, 1994), $CIRC[RL(D); RightLimit]$ has a unique model which contains facts about $RightLimit$ common in all the models of $RL(D)$. So, right limit values for some quantities may be unknown but all the known right limits would be correct.

We can detect such cases where incorrect right limit values are inferred due to some unknown right limit values with the help of $ChkRL(D)$ axioms.

$ChkRL(D)$ is derived from $Occ_P(D)$, $Con(D)$, $Con_P(D)$, $MCnst(D)$, and $FCnst(D)$ by

(a) replacing each $HoldsAt(f, t)$ in each γ by $HoldsAt(f, Successor(t))$

(b) for each $Value(p, t)$ in the antecedent of any axiom, any $Value(p, t)$ in me in the consequent of $Con(D)$ or $Con_P(D)$, and each $Value(p, t)$ in me_1 or me_2 in the

consequent of $MCnst(D)$, replacing $Value(p,t)$ by a fresh variable z_p unique to p and conjoining

$$((RightLimit(p,t,z_p) \wedge rl_p = 1) \vee (\neg\exists r.RightLimit(p,t,r) \wedge rl_p = 0)),$$

where rl_p is another fresh variable unique to p and r is a fresh variable too,

(c) replacing each mathematical constraint, $me_1 \succeq me_2$, in the antecedent by

$$((\prod_{p \in me_1, me_2} rl_p \neq 0 \wedge me_1 \succeq me_2) \vee (\prod_{p \in me_1, me_2} rl_p = 0)),$$

(d) conjoining $\prod_{p \in \gamma} rl_p = 0$ to the antecedent,

(e) replacing the consequent by $UnknReqdRightLimit(t)$.

In the second step two variables are introduced for each quantity p whose value is used in a given axiom. z_p denotes the right limit value if it is known and rl_p denotes whether the right limit is known. In the third step all mathematical constraints are reformulated such that they are checked only if the right limit values for all the quantities involved are known. $p \in me_1, me_2$ is used to denote those ps for which $Value(p,t)$s occur in me_1 or me_2. The condition in the fourth step checks if any of the required right limit values are not given. If at a given time some required right limit values are not known, and if the $me_1 \succeq me_2$ are satisfied for all constraints for which the required right limit values are known, then $UnknReqdRightLimit(t)$ is inferred true for that time t.

Consider, for example, the following $Con_P(D_{10})$ axiom:

$$D_{10} = [HoldsAt(F,t) \wedge Value(P_4,t) > Value(P_3,t)] \longrightarrow PartValue(P_5,t,PX,Value(P_1,t) + Value(P_2,t))$$

Then $ChkRL(D_{10})$ is derived to be:

$$[HoldsAt(F, Successor(t))$$

$$\wedge ((RightLimit(P_1, t, z_{P_1}) \wedge rl_{P_1} = 1) \vee (\neg\exists r.RightLimit(P_1, t, r) \wedge rl_{P_1} = 0))$$
$$\wedge ((RightLimit(P_2, t, z_{P_2}) \wedge rl_{P_2} = 1) \vee (\neg\exists r.RightLimit(P_2, t, r) \wedge rl_{P_2} = 0))$$
$$\wedge ((RightLimit(P_3, t, z_{P_3}) \wedge rl_{P_3} = 1) \vee (\neg\exists r.RightLimit(P_3, t, r) \wedge rl_{P_3} = 0)$$
$$\wedge ((RightLimit(P_4, t, z_{P_4}) \wedge rl_{P_4} = 1) \vee (\neg\exists r.RightLimit(P_4, t, r) \wedge rl_{P_4} = 0))$$
$$\wedge ((rl_{P_4} \times rl_{P_3} \neq 0 \wedge z_{P_4} > z_{P_3}) \vee rl_{P_4} \times rl_{P_3} = 0)$$
$$\wedge rl_{P_4} \times rl_{P_3} \times rl_{P_2} \times rl_{P_1} = 0] \longrightarrow UnknReqdRightLimit(t).$$

Finally whether $UnknReqdRightLimit(t)$ is true at any given time is determined via:

$$CIRC[ChkRL(D); UnknReqdRightLimit].$$

Let $ChkRL_{Con}(D)$ denote the subset of $ChkRL(D)$ axioms derived from $Con($ $D) \wedge Con_P(D)$, and $CIRC_{RL}(D)$= $CIRC^A[RL(D); RightLimit, PartRightLimit]$ $\wedge$ $CIRC[ChkRL_{Con}(D);$ $UnknReqdRightLimit]$. Proposition 5.3.2 states that if all the required right limit values for quantities are inferred to be known from axioms in $ChkRL_{Con}(D)$ then $CIRC^A[RL(D); RightLimit,$ $PartRightLimit]$ gives all the correct right limit values for quantities (right limits for some quantities may potentially be unknown, but the known right limits are correct). And using that Proposition 5.3.3 states that in addition to all required right limit values being known, if it is known that there exists exactly one set of correct right limit values then $CIRC^A[RL(D); RightLimit, PartRightLimit]$ give those correct set of values.

Proposition 5.3.2. *If for given time T, $CIRC_{RL}(D) \models \neg UnknReqdRightLimit(t)$, for all $t \leq T$, then the models for $CIRC^A[RL(D); RightLimit, PartRightLimit] \wedge$ $(ECPR9)$ are valid models for $Con(D) \wedge CIRC^A[T_{ESI}[Con_P(D)]; PartValue] \wedge$ $(ECP25) \wedge (ECP26) \wedge (ECP30) \wedge (ECP31) \wedge (ECP33)$ for the values of quantities at $Successor(T)$. That is, all the right limit values at time T provide valid values for the corresponding quantities at $Successor(T)$.*

Proof. By construction of $ChkRL_{Con}(D)$ if $UnknReqdRightLimit(T)$ is not true then all required right limit values for quantities are known. Let $RL'(D)$ is derived

from $RL(D)$ by replacing $RightLimit(p, t, v_p)$ by $v_p = Value(p, Successor(t))$ for each $RightLimit(p, t, v_p)$ in the antecedents of $Con_{RL}(D)$ and $Con_{PRL}(D)$.

Then, since $UnknReqdRightLimit(T)$ is false, a model for $CIRC^A[RL(D); RightLimit, PartRightLimit]$ is also a model for $CIRC^A[RL'(D); RightLimit, PartRightLimit]$ independent of any other unknown (not required) right limit values of parameters. The same holds if in addition we replace the consequents of the form $RightLimit(p, t, r)$ by $Value(p, Successor(t)) = r$. Further, the same holds if all $PartRightLimit(p, t, px, r)$ were replaced by $PartValue(p, Successor(t), px, r)$ and we consider $CIRC^A[T_{ESI}[RL'(D)]; PartValue]$.

Now a model of $CIRC^A[Con(D) \wedge Con_P(D) \wedge (ECP25) \wedge (ECP26) \wedge (ECP30) \wedge (ECP31) \wedge (ECP33); PartValue]$ is also a model of $Con(D) \wedge (ECP25) \wedge (ECP26) \wedge (ECP30) \wedge (ECP31) \wedge (ECP33) \wedge CIRC^A[T_{ESI}[Con_P(D)]; PartValue]$. □

Proposition 5.3.3. *Let* $CIRC_{RL}(D) = CIRC^A[RL(D); RightLimit, PartRightLimit] \wedge CIRC[ChkRL_{Con}(D); UnknReqdRightLimit]$. *If for a given time* T,

$$CIRC_{RL}(D) \models \neg UnknReqdRightLimit(T), \text{ and}$$

$Con(D) \wedge CIRC^A[T_{ESI}[Con_P(D)]; PartValue] \wedge (ECP25) \wedge (ECP26) \wedge (ECP30) \wedge (ECP31) \wedge (ECP33)$ *has just one model at* $Successor(T)$ *then* $CIRC^A[RL(D); RightLimit, PartRightLimit] \wedge (ECPR9)$ *gives the unique model.*

Proof. From Proposition 5.3.2 we know that if $\neg UnknReqdRightLimit(T)$ is true then $CIRC^A[RL(D); RightLimit, PartRightLimit] \wedge (ECPR9)$ at time T is a valid model for $Con(D) \wedge CIRC^A[T_{ESI}[Con_P(D)]; PartValue] \wedge (ECP25) \wedge (ECP26) \wedge (ECP30) \wedge (ECP31) \wedge (ECP33)$ at time $Successor(T)$. If the latter formula is known to have just one model then the valid model is the intended model. □

5.3.2 Active Equations

We introduce a new sort, $\mathcal{EQ}$, for uniquely identifying DAEs and mathematical constraints. And, we introduce predicates $AEqnHoldsAt$ and $CnstHoldsAt$. Recall that $EqnHoldsAt(EQ, P, T)$ denotes that equation EQ that gives the value for quantity P is active at time T. Similarly, $AEqnHoldsAt(EQ, P, T)$ denotes that

an equation EQ that gives an additive value for quantity P is active at time T. $CnstHoldsAt(EQ, T)$ denotes that constraint EQ is active at time T.

$Eqn(D)$ is derived from $Con(D)$ by replacing $Value(p, t) = me$ in the consequent by $EqnHoldsAt(Eq(\mathbf{x}), p, t)$ where Eq is unique for each axiom, $Eq(\mathbf{x})$ is of sort $\mathcal{EQ}$, and the tuple of arguments $\mathbf{x}$ is determined from the variable operands – either variable quantities or variable real-valued arguments of fluents – in me.

$AEqn(D)$ is derived similarly from $Con_P(D)$ by replacing $PartValue(p, t, px, me)$ in the consequent by $AEqnHoldsAt(Eq(\mathbf{x}), p, t)$ where Eq is unique for each axiom, $Eq(\mathbf{x})$ is of sort $\mathcal{EQ}$, and the tuple of arguments $\mathbf{x}$ is determined from the variable operands in me.

$CnstEqn(D)$ is derived similarly from $PCnst(D)$ by replacing $me_1 \succeq me_2$ in the consequent by $CnstHoldsAt(Eq(\mathbf{x}), t)$ where Eq is unique for each axiom, $Eq(\mathbf{x})$ is of sort $\mathcal{EQ}$, and the tuple of arguments $\mathbf{x}$ is determined from the variable operands in me_1 and me_2.

Consider the example domain below.

$$\begin{aligned} D_{11} =& ([HoldsAt(F(x), t) \wedge Value(p_1, t) > Value(p_2, t) \wedge y = Value(p_3, t)] \\ & \longrightarrow Value(p_5, t) = Value(p_4, t) \times x \times y) \\ & \wedge ([HoldsAt(F(x), t) \wedge Value(p_1, t) > Value(p_2, t) \wedge y = Value(p_3, t)] \\ & \longrightarrow PartValue(p_5, t, x, Value(p_4, t) \times x \times y)) \\ & \wedge ([HoldsAt(F(x), t) \wedge Value(p_1, t) > Value(p_2, t) \wedge y = Value(p_3, t)] \\ & \longrightarrow Value(p_5, t) > Value(p_4, t) \times x \times y) \end{aligned}$$

Then, $Eqn(D_{11}) \wedge AEqn(D_{11}) \wedge CnstEqn(D_{11})$ is derived to be:

$$\begin{aligned} & ([HoldsAt(F(x), t) \wedge Value(p_1, t) > Value(p_2, t) \wedge y = Value(p_3, t)] \\ & \longrightarrow EqnHoldsAt(Eq1(p_4, x, p_3), p_5, t)) \\ & \wedge ([HoldsAt(F(x), t) \wedge Value(p_1, t) > Value(p_2, t) \wedge y = Value(p_3, t)] \\ & \longrightarrow AEqnHoldsAt(Eq2(p_4, x, p_3), p_5, t)) \\ & \wedge ([HoldsAt(F(x), t) \wedge Value(p_1, t) > Value(p_2, t) \wedge y = Value(p_3, t)] \end{aligned}$$

$$\longrightarrow CnstHoldsAt(Eq3(p_5, p_4, x, p_3), p_5, t))$$

The DAEs and constraints active at any given time are given by:

$$CIRC[Eqn(D) \wedge AEqn(D) \wedge CnstEqn(D); EqnHoldsAt, \\ AEqnHoldsAt, CnstHoldsAt].$$

5.3.3 Equations for Determining Next Occurrence of Break in Continuity

We introduce a new relation $ChkCndAt$. $ChkCndAt(EQ, T)$ denotes that mathematical condition EQ, which may be a conjunction of sub-conditions, is active at time T.

Antecedent of an axiom, γ, can be partitioned as $\gamma_q \wedge \gamma_{\neg q}$, such that γ_q involves conditions on values of quantities and $\gamma_{\neg q}$ does not.

We construct $ChkCnd(D)$ from the antecedents of $Occ_P(D) \wedge Con(D) \wedge Con_P(D) \wedge PCnst(D) \wedge FCnst(D)$. For each antecedent γ, construct

$$\gamma_{\neg q} \longrightarrow CnstHoldsAt(Eq(\mathbf{x}), t),$$

where Eq is unique for each axiom, $Eq(\mathbf{x})$ is of sort $\mathcal{EQ}$, and the tuple of arguments $\mathbf{x}$ is determined from the variable real values in γ_q.

Changes in the state of γ_q is relevant only if $\gamma_{\neg q}$ is true. Also, the state of $\gamma_{\neg q}$ remains the same between any two consecutive action occurrences. Assume that $\gamma_{\neg q}$ is true immediately after an action occurrence. Now if γ_q is also true immediately after an occurrence then the time at which γ_q becomes false may be relevant in determining the next action occurrence. Similarly for vice-versa.

Consider the example domain below.

$$D_{12} = ([HoldsAt(F(x), t) \wedge Value(p_1, t) > Value(p_2, t) \wedge y = Value(p_3, t) \times x \wedge \\ y + Value(p_4, t) > 0] \longrightarrow Value(p_6, t) = Value(p_5, t) \times x \times y)$$

Then, $ChkCnd(D_{12})$ is derived to be:

$$HoldsAt(F(x), t) \longrightarrow ChkCndAt(Eq4(p_1, p_2, p_3, x, p_4), t)$$

Since any change in the continuity of any quantity or change in the state of any secondary fluent (as well as any primary fluent) must be accompanied (or rather caused by) an action occurrence, it may seem sufficient to consider $Occ_P(D)$ for the derivation of $ChkCnd(D)$. Axioms outside of $Occ_P(D)$ are also used in the derivation for consistency checking, to detect any unwarranted change in the continuity of quantities or change in the states of secondary fluents.

Let there be a change in the states of some active conditions at T or immediately after T. Then $T = PotNext(T_0)$ for some $T_0 < T$. At time T we know the state of all the fluents which is the same as the state after T_0, and the values of all the quantities from the equations active after T_0. We can recompute the values at time T and $Successor(T)$. If there is a break in the continuity of any quantity at T or $Successor(T)$ without any action to cause the same then that would result in inconsistency by way of multiple values for $Value(q, T)$ or $Value(q, Successor(T))$ for some quantities q. Likewise if the state of a secondary fluent is required to be changed at T or $Successor(T)$ without any action occurrence, it would result in inconsistency by way of $HoldsAt(f, T)$ required to be both true and false simultaneously.

The conditions active at any given time are given by:

$$CIRC[ChkCnd(D); ChkCndAt].$$

Let Δ denotes a very small quantity greater than zero, and $Cnd[t] = f[t] - g[t] \geq 0 \wedge h[t] < 0$ where Cnd, f, g, and h are some mathematical functions of time. Consider that the condition $Cnd[t]$ is active since time 0. The earliest time after 0 at which the truthness of $Cnd[t]$ changes can be determined as follows.

If $Cnd[0]$ is true then we solve the equations $f[t] - g[t] + \Delta = 0$ and $h[t] = 0$ independently for values of t. The minimum of the solutions such that $t > 0$ is the earliest time at which truthness changes, that is the earliest time when $Cnd[t]$ is

false. The two equations solved above follows from the logic that if any of the two conditions are falsified then their conjunction is falsified too.

Instead, if $Cnd[0]$ is false then we solve the equations $f[t] - g[t] = 0 \wedge h[t] < 0$ and $h[t] + \Delta = 0 \wedge f[t] - g[t] >= 0$ independently for values of t. Here solving $f[t] - g[t] = 0 \wedge h[t] < 0$ refers to solving the equation $f[t] - g[t] = 0$ and retaining those solutions for which $h[t] < 0$ also holds. Again, the minimum of the solutions such that $t > 0$ is the earliest time at which truthness changes, that is the earliest time when $Cnd[t]$ is true. The two equations solved above follows from the logic that for the conjunction to become true one of the conjuncts must already be true and the other must just become true (or both just become true simultaneously).

5.3.4 The Revised Event Calculus Theory

The circumscription policy under the new semantics with the restrictions, $CIRC_{CECPR}(D)$, is given below.

$$
\begin{aligned}
CIRC_{CECPR}(D) = {} & CIRC[Nar(D); Happens, InitialisedTrue, InitialisedFalse, \\
& \qquad InitialValue, Secondary] \\
& \wedge CIRC[Eff(D); Initiates, Terminates] \\
& \wedge CIRC[Inst(D) \wedge (ECP27) \wedge (ECP28) \wedge (ECP29); Breaks, BreaksTo] \\
& \wedge CIRC[Inst_P(D); BreaksPartBy] \\
& \wedge CIRC^A[T_{ESI}[Con_P(D)]; PartValue] \\
& \wedge CIRC^A[Con_{RL}(D) \wedge Con_{PRL}(D) \wedge (ECP25) \wedge (ECP26) \wedge (ECP31) \\
& \qquad \wedge (ECP33); RightLimit, PartRightLimit] \\
& \wedge CIRC[ChkRL(D); UnknReqdRightLimit] \\
& \wedge CIRC[Eqn(D) \wedge AEqn(D) \wedge CnstEqn(D) \wedge ChkCnd(D); EqnHoldsAt, \\
& \qquad AEqnHoldsAt, CnstHoldsAt, ChkCndAt] \\
& \wedge Con(D) \wedge Cnst(D) \wedge Rem(D) \wedge Una(D) \wedge ECPR,
\end{aligned}
$$

where $ECPR$= $(ECP1) \wedge \ldots \wedge (ECP24) \wedge (ECP30) \wedge (ECP32) \wedge (ECP34) \wedge \ldots \wedge (ECP42) \wedge (ECPR1) \wedge \ldots \wedge (ECPR9)$.

The creation of new axioms –$Con_{RL}(D) \wedge Con_{PRL}(D) \wedge ChkRL(D) \wedge Eqn(D) \wedge AEqn(D) \wedge CnstEqn(D) \wedge ChkCnd(D)$ – derived from the user-defined axioms and impositions of restrictions – $(ECPR1) \wedge \ldots \wedge (ECPR6)$– facilitates not just in integrity checking but also in separation of logical reasoning and solving of equations.

Let T_1 be either 0 or a point where an action occurs. $Eqn(D) \wedge AEqn(D) \wedge CnstEqn(D)$, as part of $CIRC_{CECPR}(D)$, help via logical reasoning in inferring a set of DAEs and mathematical constraints that are active immediately after T_1, which are solved with help of right limit values of quantities at time T_1 for obtaining particular solutions to the active DAEs. And, $ChkCnd(D)$ help infer mathematical conditions which are solely responsible to trigger a change in the above set of DAEs and mathematical constraints in some future time. Let some changes are triggered at T_2. Then logical reasoning over $CIRC_{CECPR}(D)$ is used to detect any inconsistent changes in the states of quantities and/or fluents or the next occurrence of an action. Also, T_2 and the values of quantities between T_1 and T_2 are determined purely by reasoning over active DAEs, mathematical constraints, and relevant mathematical conditions. The above is what we mean by separation of logical reasoning and solving of equations. And, such a separation allows for using off the shelf reasoners, and stitching them together to design a reasoner for the Event Calculus.

5.4 Event Calculus to Answer-Set Programs

Here we extend the results on reformulation of Event Calculus theories as answer-set programs given by Lee and Palla (2012), from the version of the Event Calculus covered by them to the version covered in the article. Recall that in the version considered in (Lee & Palla, 2012b) there is no distinction between fluents and quantities, continuous-changes are described only via functions of time (not ODEs), and aggregates are not used.

Recall that the Splitting Theorem states that if $\mathbf{P}$ and $\mathbf{G}$ are disjoint tuples of distinct predicate constants in F and G such that F is negative on $\mathbf{Q}$ and G is negative on $\mathbf{P}$ and each strongly connected component of predicate dependency graph of $F \wedge G$ is a subset of $\mathbf{P}$ or a subset of $\mathbf{Q}$, then $SM[F \wedge G; \mathbf{P}, \mathbf{Q}] \equiv SM[F; \mathbf{P}] \wedge SM[G; \mathbf{Q}]$. The result holds for stable model semantics of first-order

formulas without aggregates. We can extend the result to first-order formulas with aggregate expressions.

Definition 5.4.1. A first-order formula F with aggregates is referred to as a *simple* formula if every aggregate expression in the formula obeys the following two restrictions:

- it has a simple antecedent occurrence (that is, it occurs in the antecedent of exactly one implication in F and contains no aggregate expression).
- it has no implication (particularly, it contains no negation).

First-order formulas without aggregates are trivially simple formulas. Also, ag-canonical formulas are simple formulas.

The definitions of positive occurrence and negated occurrence in simple formulas is given below. The definitions of predicate dependency graph, and others, for simple formulas need no change.

An occurrence of predicate constant or any other subexpression in a simple formula is called *positive* if it does not belong to an aggregate expression and the number of implications containing that occurrence in the antecedents is even. (It is strictly positive if that number is 0, as before.) An occurrence of a predicate constant in a simple formula is *negated* if it belongs to an aggregate expression or a subformula of the form $\neg F$, and *nonnegated* otherwise.

The Splitting Theorem holds for simple formulas too. In particular, Lemma 1 from (Ferraris et al., 2009) is true for simple formulas too. Rest of the proof follows exactly.

Lemma 5.4.1. *For a simple formula F, $(\mathbf{p} \leq \mathbf{P}) \wedge F^*(\mathbf{p}) \longrightarrow F$ is logically valid.*

Proof. by induction on F. □

Proposition 5.4.2. *Let F, G be simple formulas, and let $\mathbf{P}$, $\mathbf{Q}$ be disjoint tuples of distinct predicate constants. If (i) each strongly connected component of $DG_{\mathbf{PQ}}[F \wedge G]$ is a subset of $\mathbf{P}$ or a subset of $\mathbf{Q}$. (ii) F is negative on $\mathbf{Q}$, and (iii) G is negative on $\mathbf{P}$ then $SM[F \wedge G; \mathbf{P}, \mathbf{Q}] \equiv SM[F; \mathbf{P}] \wedge SM[G, \mathbf{Q}]$.*

By Proposition 3.5.5 $CIRC^A[F;\mathbf{P}] \longleftrightarrow SM[F;\mathbf{P}]$ for ag-canonical formulas F. Also then, $CIRC[F;\mathbf{P}] \longleftrightarrow SM[F;\mathbf{P}]$ for canonical formulas F.

The following are true, as remarked in (Lee & Palla, 2012b). $Nar(D)$ is canonical relative to $\{Happens, InitializedTrue, InitializedFalse, InitialValue, Secondary\}$. $Eff(D)$ is canonical relative to $\{Initiates, Terminates\}$. Also, it is easy to check that $Inst(D)$ is canonical relative to $\{Breaks, BreaksTo\}$, $Inst_P(D)$ is canonical relative to $\{BreaksPartBy\}$, $T_{ESI}[Con_P(D)]$ is ag-canonical relative to $\{PartValue\}$, $Con_{RL}(D) \wedge Con_{PRL}(D) \wedge (ECP33)$ is ag-canonical relative to $\{RightLimit, PartRightLimit\}$, $ChkRL(D)$ is canonical relative to $\{UnknReqdRightLimit\}$, and $Eqn(D) \wedge AEqn(D) \wedge CnstEqn(D) \wedge ChkCnd(D)$ is canonical relative to $\{EqnHoldsAt,\ AEqnHoldsAt, CnstHoldsAt, ChkCndAt\}$.

Therefore the $CIRC_{CECPR}(D)$ policy can be rewritten as $SM_{CECPR}(D)$ policy in terms of the SM operator.

$$
\begin{aligned}
SM_{CECPR}(D) = {} & SM[Nar(D); Happens, InitialisedTrue, InitialisedFalse, \\
& \qquad\qquad InitialValue, Secondary] \\
& \wedge SM[Eff(D); Initiates, Terminates] \\
& \wedge SM[Inst(D) \wedge (ECP27) \wedge (ECP28) \wedge (ECP29); Breaks, BreaksTo] \\
& \wedge SM[Inst_P(D); BreaksPartBy] \\
& \wedge SM[T_{ESI}[Con_P(D)]; PartValue] \\
& \wedge SM[Con_{RL}(D) \wedge Con_{PRL}(D) \wedge (ECP25) \wedge (ECP26) \wedge (ECP31) \\
& \qquad\qquad \wedge (ECP33); RightLimit, PartRightLimit] \\
& \wedge SM[ChkRL(D); UnknReqdRightLimit] \\
& \wedge SM[Eqn(D) \wedge AEqn(D) \wedge CnstEqn(D) \wedge ChkCnd(D); EqnHoldsAt, \\
& \qquad\qquad AEqnHoldsAt, CnstHoldsAt, ChkCndAt] \\
& \wedge Con(D) \wedge Cnst(D) \wedge Rem(D) \wedge Una(D) \wedge ECPR
\end{aligned}
$$

We make a few changes next. First we introduce a predicate $Value$ to replace the function $Value$. $Value(P, T, R)$ denotes that value of quantity P is R at time

T. We assume that Axiom (ECPR10) is conjoined with ECPR.

$$[Value(p,t,r_1) \wedge Value(p,t,r_2)] \longrightarrow r_1 = r_2 \qquad \text{(ECPR10)}$$

Next as in (Lee & Palla, 2012b), we rewrite some domain-independent axioms to avoid strictly positive occurrences of predicates $\{Initiates, Terminates, Happens, BreaksTo, BreaksPartBy\}$ by prepending predicates with double negations, $\neg\neg$. For example (ECP25) and (ECP26) are rewritten equivalently into:

$$[\neg\neg BreaksTo \wedge \neg\neg Happens(a,t)] \longrightarrow RightLimit(p,t,r)$$

$$[\exists a,r.(\neg\neg BreaksPartBy(a,p,t,r) \wedge \neg\neg Happens(a,t)) \wedge s = Value(p,t) + \#sum\langle r,a.(BreaksPartBy(a,p,t,r) \wedge Happens(a,t))\rangle] \longrightarrow RightLimit(p,t,s)$$

Earlier in the directed graph of a formula containing (ECP25) and (ECP26) there would be an edge from $RightLimit$ to $Happens$, $BreaksTo$, and $BreaksPartBy$. However, in their rewritten forms, (ECP25) and (ECP26) contribute no edge to the graph. This would allow us to apply the Splitting Theorem on SM_{CECPR}, as discussed below.

Since first-order FLP semantics (recall that HEX-programs are based on the FLP-reduct semantics) and first-order stable model semantics may not coincide for formulas with double negation, we can reformulate such axioms to rid the double negations using auxiliary predicates. For example, $\neg\neg b \longrightarrow a$ can be replaced by $(\neg b' \longrightarrow a) \wedge (\neg b \longrightarrow b') \wedge \neg(b \wedge b')$. The two are equivalent under the first-order stable model semantics.

Proposition 5.4.3. *$SM[\neg\neg b \longrightarrow a]$ is equivalent $SM[(\neg b' \longrightarrow a) \wedge (\neg b \longrightarrow b') \wedge \neg(b \wedge b')]$ modulo interpretation of b', where b' is an auxiliary term distinct from b and a.*

Proof. $SM[\neg\neg b \longrightarrow a] \equiv [(b \longrightarrow a) \wedge (b \longrightarrow a^*)]$, and

$SM[(\neg b' \longrightarrow H) \wedge (\neg b \longrightarrow b') \wedge \neg(b \wedge b')] \equiv [(b \longrightarrow a) \wedge (b \longrightarrow a^*) \wedge (b \longleftrightarrow \neg b')]$. □

Let $ECALL(D)$ denotes the collection of domain-independent axioms of Event Calculus, user-defined domain-specific axioms ($T_{ESI}[Con_P(D)]$ in the case of $Con_P($ $D)$, initially), and the new sets of axioms derived from the domain-specific axioms. Recall that $pr(F)$ denotes the set of all the predicate constants in F, and let MPR denote all the predicates that are minimized in various parts of $CIRC_{CECPR}$. Also recall that $Choice(\mathbf{P})$ denotes the conjunction of choice formulas $\forall \mathbf{x}(P(\mathbf{x}) \vee \neg P(\mathbf{x}))$ for all predicate constants P in $\mathbf{P}$. Using the Splitting Theorem we can rewrite the $SM_{CECPR}(D)$ in terms of a single application of the SM operator to $ECALL(D)$. Similarly to Theorem 2 from (Lee & Palla, 2012b),

$$\begin{aligned} SM_{CECPR}(D) &\equiv SM[ECALL(D); MPR] \\ &\equiv SM[ECALL(D) \wedge Choice(pr(ECALL(D)) \setminus MPR)]. \end{aligned}$$

When the stable model is computed for one single formula, instead of in different parts which are conjoined, $T_{ESI}[Con_P(D)]$ in $ECALL(D)$, that is in $SM[ECALL(D)$ $\wedge Choice(pr(ECALL(D)) \setminus MPR)]$, can equivalently be replaced by $Con_P(D)$. That is, the ESI-reformulation is not required then. Assuming that $ECALL(D)$ contains no double negation or ESI-reformulation, using Theorem 7 from (Bartholomew, Lee, & Meng, 2011), the first-order stable model semantics and the FLP semantics coincide for $ECALL(D)$.

For translation of First-order formulas into disjunctive logic programs (LP) see (Lee & Palla, 2012b); particularly Sections 5 and 6, Definition 9 (*EC2ASP translation*), and Theorem 8 (σ-*equivalence* of an event calculus description T and the answer-set program obtained from T using the EC2ASP translation.). Event Calculus to answer-set program translation (that is, EC2ASP translation) can be straightforwardly extended to the Event Calculus discussed in the paper, which has new predicates and uses aggregates. The user-defined axioms do not contain aggregates, and so $ECALL(D)$ is always a simple formula. In the EC2ASP translation the aggregates in a simple formula require no modification.

For any (second-order) formulas F and G of some signature and any subset σ of that signature, we say that F is σ-*equivalent*) to G if the class of models of

F restricted to σ is identical to the class of models of G restricted to σ. Theorem 8 from (Lee & Palla, 2012b) states that if T is an Event Calculus description of signature σ and F is a First-order logic representation of the program obtained from T by applying EC2ASP translation, then T is σ-equivalent to $SM[F]$. The above result also holds for the version of Event Calculus discussed in the paper.

5.5 Constructing Models for Continuous-Time and Continuous-Change Event Calculus

Answer-set solvers produce *answers* or models for answer-set programs. And since Event Calculus theories can be reformulated as answer-set programs answer-set solvers could be used to produce their models. Answer-set solvers compute answers by grounding the programs and verifying if the grounding is stable. Thus unlike the discrete-time scenario discussed in (Kim, Lee, & Palla, 2009), where time-points are assumed to be all integers from 0 to a maximum integer value, it seems infeasible to compute the models for continuous-time Event Calculus using a single answer-set program. Instead, we can construct models for individual (and few of the) time-points separately. Also, since fluents are persistent and quantities are continuous after an action occurrence up till the next action occurrence, we can focus at constructing models for those times at which actions occur (including when actions are triggered). Recall that while break in the state of any fluent or the continuity of any quantity must be caused by an action occurrence, to detect any unexpected breaks not caused by an action occurrence we construct models for times other than the landmark points too. We refer to those time points by *potential* landmark points. The states of fluents and quantities between consecutive (potential) landmark points can be determined from the states and active equations active at the time of the later occurrence.

Also, the key to the solution to the frame problem in circumscriptive Event Calculus, where a domain description is split into different parts that are circumscribed separately, is that it ensures that the domain theory and narrative descriptions are circumscribed independently and temporal projection does not interfere with the minimization (Shanahan, 1997). $CIRC_{CECA}(D)$, $CIRC_{CECP}(D)$, and

$CIRC_{CECPR}(D)$ are designed in a similar manner (wherein the new parts, axioms about additive effects for example, are circumscribed independently). Since temporal projection does not interfere with the minimization we can build models for t_1 independent of any time t_2 such that $t_1 < t_2$. Therefore we can deduce potential landmarks and construct models for those times successively, starting from the initial time 0. (Also, the models for potential landmarks are *modularly composable* in the sense of *incremental logic programs*(Gebser et al., 2011).)

In addition to continuous-time, we are also dealing with arbitrary real-valued quantities, again unlike (Kim et al., 2009). Again, it is not feasible to consider all possible real values especially when the task of finding answers to answer-set programs begins with groundings of the program. Here the restrictions expressed by $ECPR$ come in handy. Axiom (ECPR3) requires that initial values for all quantities must be given. By (ECPR2), at any time a quantity is continuous some ODE giving its value would be known. According to (ECPR4) if an ODE of order N is active immediately after an action occurrence at any time T then right limit values at time T for all k-th derivatives, $0 \leq k < N$, must be known. The last two restrictions imply that path of change for every quantity between any two consecutive (potential) landmarks is (deterministically) known. By the way of construction of $Con_{RL}(D)$ and $Con_{PRL}(D)$ and from Proposition 5.3.1, the right limit values at the times of action occurrences for any quantity are either given/known or derived from the known right limits at the time using equations given in the domain-specific axioms. If there are finite number of quantities and finite number of equations in a domain description then only finite number of values can be derived from known right limit values and the given equations for different quantities. At a given time of action occurrence those finite possible values can also be deduced from the known right limit values at the time, known from Axioms $(ECP25) \wedge (ECP26) \wedge (ECP31)$, before computing the answers for $RL(D)$. Axiom (ECPR5) and $ChkRL(D)$ ensure that all required right limit values are known.

Thus if the domain is given, numerical, and finite, that is restrictions imposed by $ECPR$ are also obeyed, we can use answer-set solvers for logical reasoning at the initial time-point and (potential) landmarks and immediately after those.

Unlike the condition that fluents and quantities are given and finite, restrictions imposed by $ECPR$ cannot be analyzed statically, and potentially a given domain description may violate $ECPR$ for certain narratives and not for others. One can think of stricter restrictions than $ECPR$ which could facilitate static analysis but we think that would limit the expressiveness. In general, detection of potential violations of $ECPR$ by just analyzing the user-defined domain descriptions, without testing out models for different narratives, is a relevant task for the future.

5.5.1 The CEC Model Builder

Here we describe an implementation of a model-builder for continuous-change, continuous-time Event Calculus (CEC) for given, numerical, and finite domains. It alternatingly performs logical reasoning using `DLVHEX` (Eiter et al., 2006), a prototypical reasoner for HEX-programs, and solving of equations using `Mathematica` libraries (Miyaji & Abbott, 2001).

In this section we switch to variable and constant naming conventions of answer-set programs, which are different from that of first-order logic formulas. Variable names begin with upper-case letters whereas constants are either quoted strings or they begin with lower-case letters. (In the later (sub-)sections we switch back to first-order logic notations and conventions.)

Answer-set solvers natively support answer-set programs with only non-negative integers for numbers and algebraic operations over them. We use external atoms functionality of HEX-programs to build support for real number arithmetic and comparisons. For example, $\&lt[Y_1, Y_2]$ evaluates to true if $Y_1 < Y_2$ and $\&plus[X_1, X_2](Y)$ evaluates to true if $Y = X_1 + X_2$. Non-non-negative-integer real numbers are specified as string literals, e.g. $\text{“}-1''$,$\text{“}2/3''$, $\text{“}4.5''$. To avoid loss of precision during conversions from double to string and back, we internally store the Hexadecimal floating point notations. For example, 0.1 is stored as $\text{“}0x1.999999999999ap-4''$. We have defined external atoms for addition, subtraction, multiplication, division, and power operations. Other mathematical operations such as trigonometric functions like *cosine* are not precluded, and they can be supported in the future.

We use higher order external atoms of the form, $\&aggsum[partValue, P, T,$

$dca, mask](S)$, to sum the last arguments of $partValue(P, T, PX, R)$ relations for

Algorithm 1: Deduce states at maximum time, and landmark points.

Input: ECHEX(D), MAXTIME, map_id_formulas

Output: answers with answer-sets at MAXTIME

```
1  current_time = 0;
2  answers_w_tm = {({}, 0)}// 1 empty answer-set with time-stamp 0;
3  repeat
4      foreach answer_w_time in answers_w_tm do
5          current_time = getTime(answer_w_time);
6          answer = getAnswer(answer_w_time);
7          answers_now = solveASP(ECHEX(D), answer, current_time);
8          foreach answer_now in answers_now do
9              (next,values_at_next )= solveForNext(answer_now, current_time,
               MAXTIME, map_id_formulas);
10             if next ≠ −1 then
11                 answer_w_tm_new = (answer_now ∪ {potNext(current_time,
                   next)} ∪ values_at_next, next );
12                 answers_w_tm_new = answers_w_tm_new ∪ answer_w_tm_new;
13             end
14         end
15     end
16     answers_w_tm = {};
17     foreach answer_w_tm_new in answers_w_tm_new do
18         if getTime(answer_w_tm_new) = MAXTIME then
19             answers = answers ∪ getAnswer(answer_w_tm_new);
20         else
21             answers_w_tm = answers_w_tm ∪ getAnswer(answer_w_tm_new);
22         end
23     end
24 until answers_w_tm is not empty ;
```

Algorithm 2: *solveForNext*: Deduce next landmark.

Input: answer, current_time, MAXTIME, map_id_formulas
Output: values_at_next, next

```
1  initial_values = GetInitialValues(answer, current_time);
2  daes = GetDAEs(answer, current_time, map_id_formulas);
3  constraints = GetConstraints(answer, current_time, map_id_formulas);
4  conditions = GetConditions(answer, current_time, map_id_formulas);
5  next = MAXTIME −current_time;
6  ReplaceVariable(Cnds, T, T+current_time);
7  foreach base parameter bp without a DAE do
8  |  value_bp = value of bp in initial_values;
9  |  add to daes that value of bp is constant value_bp;
10 end
11 ValueFunctions = SolveDAEs (daes, initial_values);
12 if error in solving daes then
13 |  return ({}, −1);
14 end
15 Substitute(constraints,ValueFunctions);
16 Substitute(conditions,ValueFunctions);
17 foreach constraint in constraints do
18 |  if not Satisfies(constraint, 0) then
19 |  |  return ({}, −1);
20 |  end
21 end
22 foreach condition in conditions do
23 |  if Satisfies(condition, 0) then
24 |  |  next = GetMinTimeWhenFalse(condition, next);
25 |  else
26 |  |  next = GetMinTimeWhenTrue(condition, next);
27 |  end
28 end
29 foreach constraint in constraints do
30 |  next = GetMinTimeWhenFalse(constraint, next);
31 end
32 values_at_next = EvaluateAt(ValueFunctions, next);
33 return (values_at_next ,current_time +next );
```

given P and T. Constants *dca* and *mask* denote that the arguments at the position where *mask* appears are to be summed, counted once for each unique argument at the position where *dca* appears. Aggregate summation for the complex relation, $breaksPartBy(A, P, T, R) \wedge happens(A, T)$, can be computed similarly with help of an auxiliary predicate symbol to denote the complex relation.

Given a domain description, we derive the new sets of axioms described in Section 5.3, while simultaneously maintaining a map from identifiers for math formulas used in $EqnHoldsAt$, $AEqnHoldsAt$, $CnstHoldsAt$ and $ChkCndAt$ relations (Sections 5.3.2 and 5.3.3) to the math formulas. Recall that the new sets of axioms are derived syntactically. Then we use the `F2LP` (Lee & Palla, 2009b) tool, by Lee and Palla (2012), to convert first-order logic axioms into disjunctive logic programs in `Lparse` and `Gringo` syntax. Further, we use `F2LPDLV` (Lee & Palla, 2012a) tool, also by Lee and Palla (2012), to obtain disjunctive logic programs in `DLV` syntax. We translate the logic programs in `DLV` syntax into HEX-programs where all comparisons (for example, $Y_1 < Y_2$) and algebraic operations (for example, $Y = X_1 + X_2$) are replaced by corresponding external atoms (for example, $\&lt[Y_1, Y_2]$ and $\&plus[X_1, X_2](Y)$). That is, $ECALL(D)$ (Section 5.4) is translated into HEX-programs by sequentially using `F2LP` and `F2LPDLV` tools and introducing external atoms for real; numbers arithmetic and comparisons. Let $ECHEX(D)$ denote the translation of $ECALL(D)$ into HEX-programs. Further, let $MAXTIME$ denotes maximum time for which an observation is required/requested. Note that all constant symbols in $ECHEX(D)$ are unique.

Algorithm 1 describes the creation of models for successive potential landmarks. The answer sets are maintained with a time-stamp. In Line 7, we compute the answer sets for $ECHEX(D)$ at *current_time* using `DLVHEX`. $SolveASP(arg_1, arg_2, arg_3)$ creates a stable model for the HEX-program arg_1 with input facts arg_2 for time arg_3. The number of answer sets could be zero, one, or more. The answer sets contain information such as right limit values at *current_time* and the DAEs and constraints active at *successor_time* where $successor(current\_time, successor\_time)$

[16,17] is a fact, among other things. As noted earlier, if the domain is given, numerical, and finite (and $ECPR$ is also complied with), only finitely many new real values can be generated from known real values and these real values can be procedurally precomputed before computing the stable models for $Con_{RL}(D) \wedge Con_{PRL}(D)$. Instead of precomputing all possible real values, we benefit from `DLVHEX`'s handling of terms created from external computations, which are properly accounted for in the computed stable models. By way of design of $Con_{RL}(D) \wedge Con_{PRL}(D)$ all relevant real values under the stable model semantics (and $CIRC^A$ transformation semantics) are guaranteed to be computed via external computations.

In Line 9, with help of the answer sets computed in previous steps, we compute the next (potential) landmark and the values of all quantities, including derivatives, at the next (potential) landmark, using *solveForNext* function described in Algorithm 2. If there is an error in solving DAEs or if solution to the DAEs violates the constraints then *solveForNext* returns -1 as the next landmark. In Line 11, we add the values for quantities at the next landmark as facts along with fact $potNext(current\_time, next)$[18] to the current answer set.

Algorithm 2 describes the computation of next potential landmark and the values of quantities at the landmark given a model for the current time. In lines 1 to 4, we obtain right limit values known at *current_time* and the DAEs, constraints, and conditions to check active at *successor_time* such that $successor(current\_time,$ $successor\_time)$ is a fact. The right limit values serve as the initial values for solving the DAEs. Lines 7–10 implement Axiom (ECPR2). DAEs are solved using the `DSolve` function in Mathematica, which returns values of quantities as functions of time. Recall that DAEs consist of just autonomous ODEs, so we treat the initial values of the quantities as their values at time 0. Accordingly, we make adjustments to conditions that contain variable time. We assume that variable

[16] *successor* relation corresponds with the *Successor* function of the Event Calculus, and the fact above stands for $Successor(current\_time) = successor\_time$.

[17] $ECHEX(D)$ contains no ground *successor* relations, that is it contains no facts about *successor*. Constant symbols representing *successor_time* are generated procedurally, unique for each *current_time*, and the corresponding $successor(current\_time, successor\_time)$ fact is appended to $ECHEX(D)$.

[18] *potNext* relation corresponds with the *PotNext* function of the Event Calculus, and the fact above stands for $PotNext(current\_time) = next$.

time is universally denoted by T. For example, assume an external action a occurs at time 10 represented by, $happens(a, T)$:- $T = 10$, in HEX-program notation, $current\_time = 4$, and the condition $T = 10$ is active immediately after 4. Then, in Line 6, $T = 10$ condition is changed to $T + 4 = 10$, which intuitively represents that action a occurs at 6 time units relative to the current time. The next landmark is similarly initialized to maximum time relative to the current time. $Substitute(arg_1, arg_2)$ substitutes the quantities in arg_2 by corresponding functions of time in arg_2, obtained by solving the DAEs. $Satisfies(arg_1, arg_2)$ checks if condition arg_1 is satisfied at a given time arg_2. $GetMinTimeWhenFalse(arg_1, arg_2)$ returns the earliest that condition arg_1 becomes false if it is smaller than arg_2 else returns arg_2. $GetMinTimeWhenTrue(arg_1, arg_2)$ is defined likewise. See Section 5.3.3 for a discussion on equations that can be solved to determine such times. Equations are solved using the `Solve`function in Mathematica. We choose Δ (Section 5.3.3), the smallest values greater than zero, as $2^8 * 10^{-16}$, because "Approximate numbers with machine precision or higher are considered equal if they differ in at most their last seven binary digits (roughly their last two decimal digits)" (http://reference.wolfram.com/mathematica/ref/Equal.html).

5.6 Discussion

The repeat loop in Algorithm 1 ends in finite number of steps under the finite separation assumption if the domain is given and finite (that is, answer sets for $ECHEX(D)$ at any given time are finitely many) and $MAXTIME$ is finite. That is, Algorithm 1 is guaranteed to terminate. A given Event Calculus description may be consistent till some time t but not at t. If t is smaller than $MAXTIME$, Algorithm 1 would detect an inconsistency, but not if it is larger than $MAXTIME$. That is, Algorithm 1 produces all models for a given Event Calculus description that are consistent till and at $MAXTIME$.

$ECHEX(D)$ is partially stratified. For example, stable models for any time T can be computed before $Successor(T)$. Similarly, at any given time, stable models for $Con_{RL}(D) \wedge Con_{PRL}(D)$ can be computed before $ChkRL(D)$, and stable models for $Eqn(D) \wedge AEqn(D) \wedge CnstEqn(D) \wedge ChkCnd(D)$ can be computed last. In the

current implementation we take advantage of the partial stratification (that is, the stable models computed in Line 7 are computed in parts). Computing answers in parts is not just efficient but in many cases `DLVHEX` stalls otherwise due to recursive dependencies, even though relations are locally stratified (Blair, Marek, & Schlipf, 1995).

Furthermore, `DLVHEX` currently does not support recursive rules involving higher-order external atoms. For example, rules of the form

$$partValue(q, T, P, S) \text{:-}$$
$$partValue(p, T, \_, \_) \wedge \&aggsum[partValue, p, T, dca, mask](S),$$

which states that additive quantity q has a partial value of S, where $value(P, T, S)$ holds, uniquely identified by the contributing quantity P, where P itself is an additive quantity, are not supported. Therefore we only allow *stratified* additive quantities, that is additive quantities that have predefined acyclic dependencies, and replace $partValue$ relation by multiple relations of the form $partValue_1$, $partValue_2$, etc. such that $partValue_1$ can be used for additive quantities which depend on no other additive quantities, $partValue_2$ can be used for additive values which depend on only the previous quantities, so on and so forth. Their net values are similarly represented using the relations $value_1$, $value_2$, etc. Furthermore, $Con(D)$ axioms of the form,

$$\neg\exists px, r.PartValue(P, t, px, r) \longrightarrow Value(P, t) = 0,$$

cannot be supported. Such axioms can instead be expressed, for example, using $BreaksTo(A, P, T, 0)$, to represent discontinuous change in the value of some additive quantity P to zero when all continuous additive effects active at T cease to hold immediately after T, which is caused by some action A. Similarly for $breaksPartBy$ we replace $breaksPartBy$ by $breaksPartBy_1$, $breaksPartBy_2$, etc., and introduce $rightLimit_1$, $rightLimit_2$, etc.

Consider a condition (appearing in the antecedent of some axiom), $Value(Q_1, t) = Value(Q_2, t)$, which is initially false. Let $Q_1[t]$ and $Q_2[t]$ denote the values of re-

spective quantities as function of time. For a given time T the two may be equal but $Q_1[T] = Q_2[T]$ may not be (computed as) true because it depends on computations of arbitrary precision. Such computations are not directly dealt with by the answer set solver in Algorithm 1, and they are handled in Algorithm 2. Further, instead of checking at arbitrary times t whether $Q_1[t] = Q_2[t]$, we solve for t in the equation $Q_1[t] - Q_2[t] = 0$. Let the solution to the equation be T. Assuming that T is also the next potential landmark, the values $Q_1[T]$ and $Q_2[T]$ are evaluated and the answer-set solver directly uses those values. Here we rely on `Mathematica`'s maturity and functions such as $Chop$[19] for $Q_1[T]$ and $Q_2[T]$ to have the exact same representations in machine precision. In the future, we will investigate effects of arbitrary precision computations on the models generated by Algorithm 1.[20]

Event Calculus is a narratives-based formalism, that is occurrences of exogenous actions with times of occurrences have to be explicitly specified for any meaningful reasoning, unlike non-narratives based formalisms such as Situation Calculus (Reiter, 1996) and Fluent Calculus (Thielscher, 2001). Formalisms like Situation Calculus are said to deal with *hypothetical* action occurrences where of many actions that can occur in a given situation different subsets of those occur to beget different situations. It is worth noting, however, that endogenous actions or actions that are triggered as a result of some change, for example a ball bouncing off the wall, are modeled similarly in both kind of formalisms. For example, such actions are modeled as *natural* actions in Situation Calculus (Reiter, 1996) and Fluent Calculus (Thielscher, 2001), and are distinguished from exogenous actions in that they are required to occur in all situations where the preconditions for their occurrences are satisfied. Natural actions are modeled naturally in formalisms like Event Calculus where a fact $Happens(A, T)$ is consistently interpreted as an occurrence of action A at time T.

Mueller (2006) identifies following different types of reasoning tasks: deduction or temporal projection, abduction and planning, postdiction, and model finding. Deduction or temporal projection "consists of determining the state that results from performing a sequence of actions, given an initial state". Abduction consists

[19] $Chop[v]$ "replaces approximate real numbers in v that are close to zero by the exact integer".
[20] Goldberg (1991) has a very relevant note on floating-point arithmetic.

of "determining what events might lead from an initial state to a final state", and planning "consists of generating a sequence of actions to bring about a final state starting from an initial state". Postdiction "consists of determining the initial state given a sequence of events and final state". A model finding "consists of generating models from states and events".

Algorithm 1 directly implements temporal projection or simulation, given an initial state and narratives of action occurrences. The generated models can be queried to ascertain the state that results. Given that disjunction is allowed, multiple choices of initial values of quantities can be specified or multiple choices of action occurrences may be specified, with constraints on maximum number of them that can cooccur for example. Also, secondary fluents are allowed, which can be exploited to explore multiple possibilities, possibly through a similar manner of use as discussed in Observation 5.2.4 (Section 5.2). The models generated by Algorithm 1 can be probed to ascertain facts that are true in all the models and facts that are true in some models.

If we reverse the temporal projection process we may be able to implement postdiction for some Event Calculus descriptions. Unlike temporal projection the values of quantities and derivatives are known at a later boundary instead of initial time, but those values can be used to solve for particular solutions of DAEs. Further, in postdiction, at any given landmark we first have the knowledge of the state of the fluents and quantities immediately after the landmark from which we must infer the state at the landmark. In Algorithm 1, we took the advantage of automatic inclusion of externally computed real values in the construction of stable models by `DLVHEX`, and avoided precomputing potential real values that may be relevant for the construction of different stable models. However, for postdiction, the potential real values that may be relevant to the construction of stable models at landmarks have to be precomputed by solving a number of different equations, which may be solved using `Mathematica` for example.

Abduction and Planning can only be performed by guessing events and their sequences and also times of occurrences and constructing models for each of those guesses and checking if the goals are achieved. Since disjunctions are allowed differ-

ent choices may be specified in a single description via disjunctions.

5.7 Examples

The code for the prototype model builder is accessible online (Khandelwal, 2013b). Examples are included in the `test` folder. It does not generate $ChkRL(D)$ axioms and assumes that $ECPR5$ is never violated, that is we assume that the required right limits are always known or can be derived from the known. It accepts first-order logic axioms described in the syntax of F2LP tool (Lee & Palla, 2009b), restricted by the syntax supported by `DLVHEX`. For example, `DLVHEX` does not accept function symbols and allows comparisons and binary operations over simple terms only (that is, complex expressions of the form $X1 > X2+1$ and $Y1 = Y2+Y3*Y4$ are not allowed, for example, and should be expressed as $X3 = X2 + 1 \& X1 > X3$ and $Y5 = Y3 * Y4 \& Y1 = Y2 + Y5$ respectively). Parameterized actions such as $setinmotion(ball, v_x, v_y)$ can be identified with a constant using $action\_n$ relations, where n is the number of arguments in the parameterized representation, as in $action\_3(setinmotion, ball, v_x, v_y, aid)$ (aid is an identifier for the action). Parameterized quantities and fluents can be denoted similarly. Derivatives are denoted using *derivative* relations such that $derivative(DQ, N, BQ)$ expresses that derivative quantity DQ is the N-th derivative of base quantity BQ. (The domain-independent axioms are given defined by (Khandelwal, 2013a).)

We present three examples here. The first example involves a triggered action. The second example involves discrete additive effects and their ramifications. Finally the third example involves continuous additive effects. The F2LP descriptions and the output of the model builder are given in Appendix C. In the following sections we describe the problem and simulation results in natural language.

5.7.1 Inelastic Collision of Two Balls in 2D

This example illustrates triggering of an event. Figure 5.1 depicts two balls placed at points A and B which are set in motion at different times. They eventually collide inelastically and move together. The first ball is placed at A with coordinates $(-10, -10)$ and the second is placed at B with coordinates $(20, 0)$. The first ball is

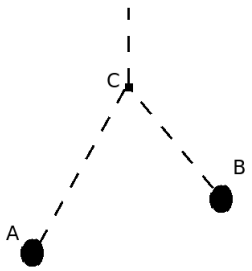

Figure 5.1: Inelastic collision of two balls. Two balls at A and B are thrown along the dashed lines possibly at different times such that they collide inelastically at C.

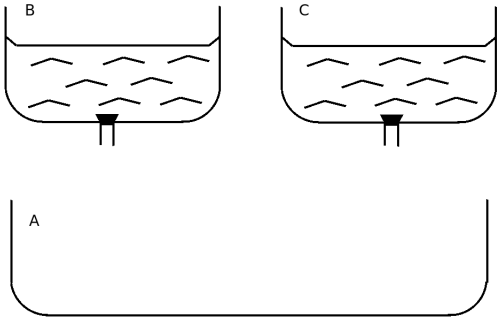

Figure 5.2: Two tanks layered above a bigger tank at the bottom. The tanks above are filled discretely, and excess water spills over into the bottom tank.

set in motion with velocity $(10, 10)$ at time 1, and the second ball is set in motion with velocity $(-10, 10)$ at time 2. The balls collide when they are at the same coordinates and the collision breaks their velocities such that both move in the same direction at the average of their velocities right before the collision.

The result of simulation is that the collision ($aid3$) happens at time 3 after which the velocities of both are $(0, 10)$.

5.7.2 Discrete, Additive Water Flow in Tanks with Spillage into Lower Tank

This example illustrates representation of discrete additive effects and their ramifications (Erdem & Gabaldon, 2006). Figure 5.2 shows tanks B and C are layered above tank A. The capacities of tanks A, B, and C are 8, 3, and 3 respectively. When the tanks above are filled instantaneously (discretely that is) any extra amount of water spills over into the bottom tank. Initially all the tanks are empty. At time

1 three concurrent actions fill tank B by amounts 1, 1, and 2 respectively. (The ramification is that 1 unit is added to tank A.) At time 2 two actions fill tank C by amounts 2 each. (The ramification is that 1 unit is added to tank A.)

The result of simulation is that immediately after time 1 the amount of water in B becomes 3 and that in A becomes 1. And immediately after time 2 the amount of water in C becomes 3 and that in A becomes 2.

5.7.3 Outflow from Upper Tanks to Lower Tanks

This example illustrates representation of continuous additive effects. Figure 5.2 shows tanks B and C have outlets, and they are above tank A. The capacities of tanks A, B and C are 20, 15 and 15 respectively. Tanks B and C are full initially and tank A is empty. The outlet of B is opened at time 1, and that of C is opened at time 2. The rate of outflows are proportional to the amount of water in the tank, and the constant of proportionality is 2. While outlets of the above tanks are open and the above tanks are non-empty they fill tank A. When tank A is full an overflow action is triggered which breaks the rate of change of amount of water in A to 0.

The result of simulation is that tank A overflows at time 2.27.

5.8 Summary

We reformulated Event Calculus axioms as transitions over landmarks, and introduced new relations and functions relevant to performing reasoning or derivations in Section 5.1. We covered different aspects by which model construction, or in general any form of reasoning, may become impractical, and introduced axiomatized restrictions to Event Calculus to detect and avoid those aspects in Section 5.2. We also introduced the default assumption that quantities are constant by default if continuous. We introduced new sets of axioms, syntactically derived from user-defined descriptions, that facilitate reasoning, in particular allowing for separation of logical reasoning and equations solving in Section 5.3. This separation allows using off-the-shelf logic reasoners and equation solvers for implementing a model-builder for Event Calculus. We reformulated Event Calculus theories as answer-set programs, specifically HEX-programs, in Section 5.4 so that we can use answer-set

solvers for logic reasoning. We described the process of constructing models for given, numerical, and finite domains, given an initial state and narratives of exogenous action occurrences, in Section 5.5. We remarked on our implementation and other aspects of the model construction in Section 5.6. Finally, we presented models generated by our implementation for some example domains in Section 5.7.

CHAPTER 6
RELATED WORK

We discuss different formalisms that reason about continuous effects, with remarks on descriptions of additive effects in those formalisms, and automated reasoners for those formalisms in Section 6.1. Then we discuss previous research on distinctive descriptions of additive effects in Section 6.2. We discuss works related to semantics for aggregate formulas and the CIRC^A transformation in Section 6.3. Finally we discuss about external computations and incremental reasoning in answer set programming in Section 6.4.

6.1 Formalisms for Representing Continuous-Changes and Automated Reasoning

Reasoning about continuous change has been studied in many different contexts such as real time systems (Henzinger, 1996), planning (Fox & Long, 2003; McDermott, 2003; Fox & Long, 2006), and reasoning about actions and their effects (Sandewall, 1989a, 1989b; Shanahan, 1990; Herrmann & Thielscher, 1996; Miller, 1996; Miller & Shanahan, 1996; Pinto, 1998; Reiter, 1996; Thielscher, 2001; Chintabathina et al., 2005; Baral, Dzifcak, Tran, & Zhao, 2006).

Hybrid automaton is a mathematical formalism for verification (or model checking) of systems involving discrete and continuous components (Henzinger, 1996). Systems are described by states and transitions between those states. A state of a hybrid automaton is defined by ODE over a finite set of continuous variables (that is, flow conditions) and boundary conditions (or invariant conditions). State transitions happen at boundary conditions or in response to external actions when the jump conditions are satisfied. Continuous additive effects determine the flow conditions and so, they cannot be specified independently. A state has to be created for every possible cumulative effect, that is for every subset of additive effects which may be active simultaneously.

The Planning Domain Definition Language (PDDL) is a family of deterministic

planning modeling languages, and is a standard for planning domains (McDermott et al., 1998). Continuous changes were introduced in PDDL 2.1 (Fox & Long, 2003), and PDDL+ (Fox & Long, 2006) is the most expressive language of the PDDL family and supports mixed discrete-continuous domains. Fox and Long (2006) provided a formal semantics for PDDL+ primitive terms via a mapping to hybrid automata. The cumulative effect on additive parameters is determined before mapping into hybrid automata. Aggregate expressions could be used to account for additive effects considered by Fox and Long (2006). McDermott (2003) proposed an alternative declarative semantics for processes in PDDL based on the branching time structure. Processes and events are interpreted through intervals, and an interpretation is a valid model if respective (pre)conditions and effects are valid in the corresponding intervals. The proposal makes no special case for additive parameters and, unlike Fox and Long (2006), PDDL description of processes are expected to explicitly describe cumulative effects under different circumstances.

Chintabathina et al. (2005) extended Causal Theories (McCain & Turner, 1997) for describing continuous changes in dynamical systems represented through transition diagrams, which can be translated into logic programs under the stable model semantics. Since entire reasoning is performed via an answer set solver they use *discretized* functions as approximations.

Baral et al. (2006) extended high-level action language `A` (Gelfond & Lifschitz, 1998) to represent continuous processes described via ordinary differential equations, to model cellular processes and mechanisms. Their Action Language for continuous processes is based on the semantics of hybrid automaton. Also, it has no notion of breaks in continuities, which is a salient feature of Miller and Shanahan's (1996) formalism. They use reasoners for hybrid automaton for their experiments.

Several logic-based formalisms for reasoning about actions and their discrete effects, such as Situation Calculus, Fluent Calculus, and Event Calculus, have been extended to reason about continuous changes.

Pinto (1998) and Reiter (1996) proposed extensions for reasoning about continuous changes described via functions of time in Situation Calculus (Reiter, 1993; Lin & Reiter, 1994), and Thielscher (2001) made similar extensions for descriptions

of continuous changes described via functions of time in Fluent Calculus. Miller (1996) introduced techniques for incorporating ODEs within circumscribed Situation Calculus. Similar extensions as those proposed for Event Calculus are possible even for Situation Calculus. Cumulative effects on additive quantities can be determined from continuous additive effects active in a situation or discrete additive effects of concurrent actions. There may not be much to gain from implicit summations of discrete additive effects however, as sets of concurrent actions have to be declared explicitly in Situation Calculus, unlike Event Calculus, where actions that happen at coinciding times are deduced to be concurrent.

The development of Event Calculus has spanned over both classical logic and logic programming traditions. The original version of the Event Calculus by Kowalski and Sergot (1986) was formulated as logic programs with negation as failure. More extensive later developments of the Event Calculus have been carried out under the classical logic setting via circumscription (Shanahan, 1999a). Kim, Lee, and Palla (2009) connected the two paradigms by providing a reformulation for axioms of circumscriptive Event Calculus as Answer Set Programs interpreted according to the first-order stable model semantics (Ferraris, Lee, & Lifschitz, 2007).

Shanahan (1990) proposed a method for expressing and reasoning about continuous changes expressed as trajectories in the Event Calculus with logic programming traditions. Herrmann and Thielscher (1996) proposed *process* as a primitive like fluent which generalized the notion of trajectories, and provided a prototypical reasoner based in Prolog. Miller and Shanahan (1996) generalized those notions further by incorporating ODE-based descriptions of processes in the circumscriptive Event Calculus. Sandewall (1989a, 1989b) also proposed a solution for combining logic and differential equations. However, Rayner (1991) showed that Sandewall's approach leaves open the possibility of anomalous models. Consider, for example, a tank with a covered outlet with some water. When the cover is removed, it causes abrupt change in the outflow from the outlet. However, if it is not explicitly specified that the action causes a discontinuity only in the outflow, a scenario where the amount of water in the tank changes abruptly and becomes zero instantaneously is also deducible (Miller, 1996). Miller (1996) and Miller and Shanahan (1996) avoided

anomalous models through the use of minimized *Breaks* predicate, which are used to explicitly enumerate (all) the potential discontinuities caused by different actions.

Current reasoners for Event Calculus allow continuous-change to be described only as functions of times, if at all. Automated theorem proving has been applied for deductions in discrete-time Event Calculus (Mueller, 2005), but sometimes require human guidance. Other reasoners are based either in Logic Programming (Kowalski & Sergot, 1986; Shanahan, 2000), SAT solving (Shanahan & Witkowski, 2004; Mueller, 2004), or answer-set programming (Kim et al., 2009). Implementations using logic programming, for example, cannot handle actions with non-deterministic effects, while those using SAT solving, for example, cannot handle recursive axioms. The implementation using answer-set solvers while very expressive, allows access only to non-negative integer values of continuously varying quantities and allows access to values only at integer time-points. We overcome those limitations by first reformulating the original Event Calculus semantics in terms of transitions over landmarks, and then separating logical and mathematical reasoning where the latter is used to determine the successive landmarks and the former is used for constructing models at the landmarks and immediately after the landmarks.

Some logic-based formalisms such as by Belleghem, Denecker, and De Schreye (1994) reason about continuous changes only qualitatively.

Continuous constraint-satisfaction problems (Neumaier, 2004) are problems concerned with finding suitable bindings for variables such that they satisfy the given set of continuous constraints, possibly containing derivatives, and constraint logic programming (Jaffar & Maher, 1994) combines constraint programming with logic programming. Our work is unique relative to continuous constraint-satisfaction problems and constraint logic programming in that it combines disjunctive logic programming with arbitrary ordinary differential equations (although in specific contexts).

Both Event Calculus and Qualitative Reasoning (Forbus, 1984; Kuipers, 2001) can be used to describe dynamical domains, and they share overlapping notions such as landmarks, but Event Calculus descriptions are more formal. Besides our implementation caters to only quantitative reasoning.

The temporal progression of a given and finite Event Calculus domains can be mapped to states and transitions between the states of a hybrid automaton (Henzinger, 1996). The states of an Event Calculus system are not predefined and they are determined by logical reasoning, unlike hybrid automaton, for which all states and transitions have to be enumerated. Otherwise, the building models for Event Calculus systems is similar to building models for hybrid automaton. However, we cannot do symbolic model checking such as that supported by `HyTech` (Henzinger, Ho, & Wong-Toi, 1997) for linear hybrid automaton.

6.2 Distinct Description of Additive Effects

There has been no significant work towards a special treatment for continuous additive effects. In the context of PDDL+, as noted earlier, the cumulative effect of additive effects is computed before mapping the PDDL+ description to hybrid automaton (Fox & Long, 2003). Belleghem et al. (1994) also dealt with *simultaneous effects* on parameters but their approach is relevant only for qualitative reasoning.

Lee and Lifschitz (2003) introduced a syntactic construct, *increments*, on top of the action language $\mathcal{C}+$, for describing discrete additive effects. Descriptions using the syntactic construct are procedurally translated to the base language, $\mathcal{C}+$. Besides, $\mathcal{C}+$ descriptions are propositional. In contrast, the semantics of $BreaskPartBy$ (as well as $PartValue$) is axiomatized in first-order logic. The benefit, for example, is that our formalism supports inter-dependencies between (the values of) additive parameters that may be simultaneously affected by continuous additive effects, with a well-defined semantics.

Erdem and Gabaldon (2005, 2006) used an increments-like construct within the Situation Calculus formalism for descriptions of direct effects of actions that belong to some *concurrent*. The *concurrent* is a (logical) sort used for describing simultaneous sets of actions in Situation Calculus (Reiter, 1996), and the combined effect of actions in a concurrent is computed by summing the effects of the actions in the concurrent. While Situation Calculus is also a first-order logic formalism like Event Calculus, the summation is however not performed via generic expressions such as generalized quantifiers or aggregate expressions. Erdem and Gabaldon (2005,

2006) also discuss ramifications in the context of discrete additive effects. We discuss representations of ramifications in the extended Event Calculus in Section 4.4. As noted before, in case of general ramifications the $CIRC^A$ transformation is required for default reasoning over discrete additive effects as well.

6.3 Closed-world Reasoning over Aggregate Formulas

Aggregates have been studied extensively in the context of databases and logics for databases but mainly for expressing queries, that is, there is no concept for recursive definitions; see Hella, Libkin, Nurmonen, and Wong (2001) for a summary. There has been much discussion on the correct semantics for formulas with aggregates, especially those with recursive definitions, in logic programming (LP) and answer set programming (ASP). Most proposals for semantics in LP, for example Hilog (Chen, Kifer, & Warren, 1993), and ASP, unlike the semantics by Lee and Meng (2009), are based on Herbrand interpretations (Lifschitz et al., 2008); see Faber et al. (2008) and Lee and Meng (2009) for a summary.

In comparison, aggregates have received less attention in first-order logic (with first-order interpretation), and mostly in the context of first-order stable model semantics (Lee & Meng, 2009; Ferraris & Lifschitz, 2010; Bartholomew et al., 2011). First-order logic with aggregates can be viewed as a special case of first-order logic with generalized quantifiers (Westerståhl, 2008) and Lisdström quantifiers (Lidström, 1966); see Ferraris and Lifschitz (2010) for details.

The CIRC^A transformation is closely related with circumscription-like transformations used in the definitions of first-order stable model semantics (Lee & Meng, 2009; Ferraris & Lifschitz, 2010) and first-order FLP semantics (Bartholomew et al., 2011). The stable model semantics is defined using the $\mathrm{SM}_{\mathbf{P}}[F]$ transformation where $\mathrm{SM}_{\mathbf{P}}[F] = F(\mathbf{P}) \wedge \neg\exists((\mathbf{p} < \mathbf{P}) \wedge F^*(\mathbf{p}))$. A model of F is also stable if it is a model of $\mathrm{SM}_{\mathbf{P}}[F]$. $F^*(\mathbf{p})$ and $F^{**}(\mathbf{p})$ (Definition 3.1.2) are similar to $F(\mathbf{p})$, used in the standard circumscription. $F^{**}(\mathbf{p})$ differs from $F(\mathbf{p})$ when F contains aggregates, and $F^*(\mathbf{p})$ differs from $F(\mathbf{p})$ when F contains negation, in addition to when it contains aggregates. Intuitively, negated formulas are also interpreted as constraints. For example, Fo_2 has only one stable model, $\{\mathrm{Q}(3)\}$, while it has two ag-

minimal models, {Q(3)} and {P(1), Q(2)}. Fo_{13} has only one stable model, {Q(3)}, while it has infinitely many ag-minimal models. $\mathrm{SM}_{P,Q}[Fo_6] \equiv \forall x(\neg P(x) \wedge Q(x))$, whereas $\mathrm{SM}_{P,Q}[Fo_7] \equiv \mathrm{CIRC}^A[Fo_7;\ \mathrm{P},\ \mathrm{Q}] \equiv \mathrm{CIRC}^A[Fo_6;\ \mathrm{P},\ \mathrm{Q}] \equiv \mathrm{CIRC}[Fo_6;\ \mathrm{P},\ \mathrm{Q}] \equiv \forall x((\neg P(x) \wedge Q(x)) \vee (\neg Q(x) \wedge P(x)))$. Kim, Lee, and Palla (2009) (and Lee and Palla, 2012) showed that circumscription and first-order stable model semantics, for formulas without aggregates, coincide on the class of canonical formulas. Canonical formulas are the largest syntactic class identified so far for which the two semantics coincide, and according to the authors, canonical formulas are general enough to cover the circumscriptive Event Calculus in which continuous changes can be described via functions of time. We have defined a similar syntactic class of formulas with aggregates, the ag-canonical class of formulas (Definition 3.5.2), for which CIRC^A transformation and first-order stable model semantics coincide.

The semantics of CIRC^A (as well as ICIRC, Definition 3.4.8) transformation is different from the parallel, prioritized, and pointwise circumscriptions (Lifschitz, 1994, 1987). As shown in Section 3.1, the parallel circumscription accepts weak models for Fo_1 and Fo_4. The prioritized circumscriptions of Fo_5 accept weak models. $\mathrm{CIRC}[Fo_5;\ \mathrm{P} > \mathrm{Q}]$ has infinitely many weak models, {Q(1), Q(-0.6)}, {Q(1), Q(-0.7)},{Q(1), Q(-0.3), Q(-0.4)}, and $\mathrm{CIRC}[Fo_5;\ \mathrm{Q} > \mathrm{P}]$ has a weak model {P(1.5)}. Minimization by pointwise circumscription checks for the impossibility of changing the value of a predicate from true to false at one point (Lifschitz, 1987). {P(0.5), P(0.6)} is a weak model of Fo_1 but it is a model of $\mathrm{Circum}(Fo_1(\mathrm{P});\ \mathrm{P})$, the pointwise circumscription of Fo_1. CIRC^A (and ICIRC) is also different from other variants such as Nested Abnormality Theory (Lifschitz, 1995), Autocircumscription (Perlis, 1988), and value minimization (Baral, Gabaldon, & Provetti, 1998). The latter were devised for first-order logic without aggregates, and cannot eliminate the weak models.

6.4 External Computation and Incremental Reasoning in Answer Set Programming

`Clingo` (Gebser et al., 2011) provides an interface to call `Lua`functions, a lightweight and embeddable scripting language, at certain points during evaluation,

for example, before grounding, after a model has been found, and after termination. While communication between the reasoner and external scripts is possible, it is constrained to happen between specific evaluation phases and is not tightly coupled to and interleaved with model building, in contrast to HEX-programs. While VI-programs (Calimeri, Cozza, & Ianni, 2007) like HEX-programs extend answer set programs with external computation of values (that is, *value invention*) and allow for bidirectional flow of values (that is, terms), unlike HEX-programs, do not support higher-order atoms (that is, VI-programs do not allow *relational* input). As a result, for example, aggregates can be implemented via external atoms in HEX-programs but not in VI-programs. The systems, `DLV-EX` and `DLV-Complex`, extend the `DLV` system with VI-programs (Calimeri et al., 2007). A HEX-program without higher-order external atoms is equivalent to a VI-program (Calimeri et al., 2007).

As noted by Calimeri et al. (2007), although the semantics of HEX-programs is given in terms of an infinite set of symbols, Eiter et al. (2005) do not address explicitly the issue of value invention (Calimeri et al., 2007). By the design of $Con_{RL}(D)$ and $Con_{PRL}(D)$, however, all the relevant new values for computation of stable models of the corresponding HEX-programs can be invented from the known values and equations if Axiom $ECPR5$ is not violated. If the restriction of Axiom $ECPR5$ is not violated then the relevant new values can be invented following a bottom-up fixpoint computation, for example. HEX-programs have also been used for many other applicationssuch as merging belief sets with *complex* merging plans (Redl, Eiter, & Krennwallner, 2011) and ranking services using *fuzzy* HEX-programs (which can be mapped into HEX-programs) (Heymans & Toma, 2008).

`Clingcon` (Gebser et al., 2011) is an answer-set solver for constraint logic programs which extends the answer set solver, `Clingo`, with constraint satisfaction problem, including satisfiability modulo theories (De Moura & Bjørner, 2011), solver. `Clingcon` does not support differential constraints or disjunctions. In comparison, we have extended an answer-set solver with differential equations solver.

Finally, computation of answers (that is, models) at successive landmarks is similar to gradually building answers for incremental logic programs (Gebser et al., 2011), which consist of a triple (B,P,Q) of logic programs. P and Q contain a (sin-

gle) parameter k ranging over the natural numbers. The base program B describes static knowledge, independent of parameter k. Whereas P captures knowledge accumulating with increasing k and Q is specific for each value of k. While k ranges only over natural numbers and is incremented (by 1) in a fixed manner, landmark values are specific to a domain, range over real numbers, and are determined by solving equations of the domain.

CHAPTER 7
CONCLUSION

Through this research we worked with Miller and Shanahan's (1996) Event Calculus formalism because it is the most expressive logic-based formalism that supports ordinary differential equations based descriptions of continuous-changes. We showed that explicit enumeration of the combined effects of concurrently active additive effects makes domain descriptions in Event Calculus and other logic-based formalisms for reasoning about actions and effects very scenario-specific and less elaboration tolerant. Besides, summation through recursion/iteration requires artificial ordering of additive effects. We proposed that additive effects be described independently, as in the existing formalisms, but the combined effects computed using some general expressions for summation according to a predefined domain-independent semantics. We introduced $PartValue$ and $BreaksPartBy$ predicates for distinct descriptions of additive continuous and discrete changes respectively, with their semantics defined via aggregate summation formulas in first-order logic. Next we showed that circumscription extended as is to formulas with aggregates is inadequate for determining the active additive effects as it selects unintended models which violate the physical chain of causality.

We defined a novel circumscription-like transformation, the $CIRC^A$ transformation, which differs from circumscription only in how aggregate expressions are transformed, to encode the default assumptions regarding additive effects. The CIRC^A transformation selects those models of circumscribed formulas that do not have unexpected facts that result from aggregations in the formulas. The CIRC^A transformation is generally applicable and not tailored solely for the problem of reasoning about actions or additive effects. It can be used with any first-order logic based formalism where aggregate formulas are to be interpreted as constraints.

We described a model-builder for continuous-time and continuous-change Event Calculus for constructing models for given, numerical, and finite domains, given an initial state and narratives of exogenous action occurrences. To implement the

model-builder we (a) reformulated some Event Calculus domain-independent axioms and introduced new relations and functions relevant to performing reasoning or derivations, (b) introduced axiomatized restrictions to make reasoning in general more tractable, (c) introduced new sets of axioms, syntactically derived from user-defined domain axioms, that facilitate separation of logical reasoning and solving of ordinary differential, and other, equations, and (d) extended Kim, Lee, and Palla's (2009) result about reformulating Event Calculus as answer-set programs to the Event Calculus wherein continuous-change is described via ordinary differential equations and additive effects are described via aggregates.

We implemented a prototypical model-builder by translating Event Calculus theories into Eiter et al.'s (2005) HEX-programs that extend answer-set programs with external computations, required for real number arithmetic and comparisons. The model-builder constructs models for the theories at landmarks and at the maximum time for which a model is requested by alternatively performing logical reasoning, for determining the models at initial time or at a landmark and immediately after that, and solving ordinary differential and other equations, for computing the next (potential) landmark. The math formulas are solved using `Mathematica` libraries. The states of fluents and the values of quantities between successive landmarks can be determined from their states and values at the later landmark and the ordinary differential equations active at the later landmark.

The model builder is of direct use in temporal projection or simulation. The techniques developed here can be extended to perform postdiction. In the context of planning and abduction, if an oracle is available that picks some narratives of action occurrences, then the model builder can be used to verify if any of the narratives produce the given final state from the given initial state.

Additive effects are very common and with the extensions proposed here, domains can be described in more generality – hence, more favorable to share and reuse – and brevity. Besides, domain descriptions can be extended with new additive effects through additive elaboration – hence, more favorable to modular development. And support for some form of automated reasoning for the Event Calculus would encourage use of logical formalisms/systems for descriptions of dynamical systems

with quantitative descriptions of continuous-changes. Crucially, the above enhancements to the Event Calculus may make it more appealing for consideration as a foundational theory for the process modeling language for the Semantic Web for representations and sharing of continuous processes.

7.1 Future Works

We mainly worked with Miller and Shanahan's (1996) Event Calculus. Since its inception many different versions of Event Calculus have been designed to address different aspects of representation of dynamic systems, such as action preconditions, actions with non-zero durations, actions with delayed effects, etc., which have been nicely summarized by Miller and Shanahan (2002) . While many features of the other versions such as action preconditions are independent of representation of additive effects, inclusions of features such as actions with non-zero durations should be investigated. Instantaneous action occurrences are only an approximate representation of real-world action occurrences which usually have non-zero durations. When actions have non-zero durations, preconditions that must hold at the start of the action and those which must hold throughout the action must be differentiated, for example, which can potentially complicate axiomatization of concurrent effects, including that of additive effects.

Qualitative reasoning (Forbus, 1984; Kuipers, 2001) automates reasoning about continuous aspects of dynamic domains for the purposes of problem solving and planning using qualitative information rather than quantitative information. There are a very few logic-based formalisms for modeling qualitative information, which are also not very expressive, for example (Belleghem, Denecker, & Schreye, 1994), and there have been a few attempts, for example (Davis, 1992), at formalizing non-logic-based qualitative reasoning formalisms such as Qualitative Process Theories (Forbus, 1984) in First-order Logic. Qualitative information is relevant as numerical information may not always be available, and many interesting behaviors can be described qualitatively. For example, if a steel container with water covered with a light plate is heated from the bottom then sometime in the future, the plate begins to move up and down under the influence of pressure build-up from the steam

and partial release of the steam. The numerical values for the temperature of the water, the rate at which the container is heated, etc. are not required to reason about many aspects of the behavior of the water or the plate. Miller and Shanahan (1996) described in some detail how the Event Calculus formalism can be used to axiomatize domains with qualitative values. That line of work should be pursued in the future to implement reasoners for qualitative reasoning building on the techniques developed in here, combined with those developed in the field of Qualitative Reasoning (Forbus, 1984; Kuipers, 2001).

Symbolic model checking is used for formal verification of systems, for example for systems described using linear hybrid automaton (Henzinger et al., 1997), where the properties to be verified are expressed in temporal logics (Galton, 2008). Intuitively, symbolic model checking seems tougher than qualitative reasoning for the Event Calculus but scopes for symbolic model checking are worth exploring in the future, for formal verifications of systems described in the formalism.

Miller and Shanahan's (1996) formalism allows for logic-based descriptions of continuous-changes via autonomous ordinary differential equations. However, many more complex mathematical tools such as partial differential equations (Evans, 2010) and perturbation theories (Broer, Takens, & Hasselblatt, 2010) are used for describing continuous-changes. Logic-based representation of such continuous-changes should be investigated.

The limitations with regards to descriptions of additive effects are not specific to Event Calculus, and other logic-based formalisms for reasoning about actions and their effects such as Fluent Calculus (Thielscher, 2001) and Situation Calculus (Reiter, 1996) also have the same limitations. The approach we took for extending the Event Calculus for distinct and efficient descriptions of additive effects is generally applicable, and it can be applied to Situation and Fluent Calculi. Furthermore, the $CIRC^A$ transformation based nonmonotonic reasoning is directly relevant and applicable in those logic-based formalisms that use circumscription for nonmonotonic reasoning. However, Situation and Fluent Calculi do not support ODE-based descriptions of continuous-changes. ODE-based descriptions of continuous-changes are not only more expressive than descriptions via functions of time, they are also

fundamental to the extension of the Event Calculus for distinct descriptions of continuous additive-effects. Going into the future, formalisms such as Situation and Fluent Calculus can be extended to support ODE-based descriptions, and similar to the extension to the Event Calculus, they can be extended for general, concise and elaboration tolerant descriptions of additive effects.

In order to implement a reasoner for the extended Event Calculus, we implemented the $CIRC^A$ transformation semantics via the first-order stable model semantics, for formulas for which the two semantics coincide. As a side note, we further restricted the formulas to be interpreted under Herbrand interpretations instead of first-order interpretations. For computation of the $CIRC^A$ transformation of more general first-order logic formulas, the automated tools for computing circumscription such as `SCAN` (Gabbay & Ohlbach, 1992) can possibly be extended. This would allow for reasoning over more general first-order logic formulas under the $CIRC^A$ transformation semantics.

Model construction for Event Calculus descriptions given an initial state and narratives of external actions, and by extension given a final state and narratives of external actions, naturally supports temporal projection or simulation and postdiction. To support planning, in addition to temporal projection and postdiction, using Event Calculus descriptions, abductive reasoning (Shanahan, 2000) techniques for descriptions involving ODE-based continuous-changes, and then those involving additive changes, can be investigated.

For the prototypical implementation of the model builder we have only implemented support for algebraic operations over reals in `DLVHEX`. Other operations such as trigonometric functions etc. can be supported straightforwardly. Besides, the support for reasoning over HEX-programs with external computations in the prototypical `DLVHEX` implementation is not robust. For example, `DLVHEX` unexpectedly diagnoses *unsafe* variables in the multiple blocks problem description (Appendix C.6). The `DLVHEX` implementation can be improved to make it more robust. Alternatively, other reasoners for disjunctive logic programs that support external computations, such as `DLV-Complex`[21] (Calimeri et al., 2007), can be tried for logic

[21]However, `DLV-COMPLEX` cannot support external computations for aggregates

reasoning.

An interesting research problem with regards to HEX-programs is to add support for recursive rules involving higher-order external atoms, as in

$$partValue(q, T, P, S) \text{:-}$$
$$partValue(p, T, \_, \_) \wedge \&aggsum[partValue, p, T, dca, mask](S),$$

and to build efficient reasoning mechanisms.

Furthermore, arbitrary precision arithmetic is prone to errors, as well detailed by Goldberg (1991) . Its impact on values in the models generated by our prototypical implementations in terms of percentage accuracy of those values, for example, should be analyzed.

Also, we use an answer-set solver for logic reasoning and answer-set solvers perform reasoning (under full expressivity) by grounding answer-set programs which despite all optimizations is computationally highly intractable. Analyzing the scalability of our prototypical implementation with respect to the size of dynamic domains it can handle, and devising scalable implementations would be crucial for practical use. Moreover, the expressiveness of domain descriptions can be restricted, for example nondeterminstic effects may be disallowed, and more scalable reasoners for (non-disjunctive) logic programs can also be deployed for logic reasoning.

Finally, we have designed the first version of a Web Ontology Language for Descriptions of Continuous-Changes (WOLCC) (Khandelwal & Fox, 2012). WOLCC provides vocabulary for descriptions of a) quantities, relations, and actions, b) direct effects of actions, c) continuous changes and relationship between quantities, and d) data modeling constraints. A hybrid system of mixed discrete-continuous changes can be modeled through WOLCC as a system which is in a state of continuous change (including no change) by default and discontinuous changes are caused by actions. The actions may be external or internal, that is triggered from continuous-changes in the system. Continuous-change can be described via autonomous ordinary differential equations, and typically there is a discontinuous change when the system reaches a boundary condition for one or more ordinary differential equations. With

proper tool support for authoring, parsing, etc. WOLCC can be positioned as a process modeling language for the Semantic Web and can be further improved through feedback from its usage.

REFERENCES

Baeten, J. C. M. (2005, May). A Brief History of Process Algebra. *Theoretical Computer Science*, *335*(2-3), 131–146.

Baker, A. B. (1991). Nonmonotonic Reasoning in the Framework of Situation Calculus. *Artificial Intelligence*, *49*, 5–23.

Baral, C., Dzifcak, J., Tran, N., & Zhao, J. (2006). Reasoning about Actions in Biophysical Systems. In *Technical Report WS-06-03. AAAI 2006 Workshop on Cognitive Robotics.* Menlo Park, CA: AAAI.

Baral, C., Gabaldon, A., & Provetti, A. (1998, July). Value Minimization in Circumscription. *Artificial Intelligence*, *102*(2), 163–186.

Bartholomew, M., Lee, J., & Meng, Y. (2011). First-Order Semantics of Aggregates in Answer Set Programming Via Modified Circumscription. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning.* Menlo Park, CA: AAAI.

Barwise, J. (Ed.). (1977). *Handbook Of Mathematical Logic* (Vol. 90). San Diego, CA: Elsevier.

Belleghem, K. V., Denecker, M., & Schreye, D. D. (1994). Representing Continuous Change in the Abductive Event Calculus. In *Proceedings of the International Conference on Logic Programming* (pp. 225–240). Cambridge, MA, USA: MIT Press.

Bellman, R. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics*, *6*, 679–684.

Blair, H. A., Marek, V. W., & Schlipf, J. S. (1995). The Expressiveness of Locally Stratified Programs. *Annals of Mathematics and Artificial Intelligence*, *15*, 209-229.

Broer, H., Takens, F., & Hasselblatt, B. (Eds.). (2010). *Handbook of Dynamical Systems* (Vol. 3). San Diego, CA: Elsevier Science.

Calimeri, F., Cozza, S., & Ianni, G. (2007). External Sources of Knowledge and Value Invention in Logic Programming. *Annals of Mathematics and Artificial Intelligence*, *50*, 333-361.

Chen, W., Kifer, M., & Warren, D. S. (1993). HiLog: A Foundation for Higher-order Logic Programming. *Journal Of Logic Programming*, *15*(3), 187–230.

Chintabathina, E., Gelfond, M., & Watson, R. (2005). Modeling Hybrid Domains Using Process Description Language. In *Proceedings of Answer Set Programming: Advances in Theory and Implementation* (Vol. 142). CEUR-WS.org. ([Online] http://www.ceur-ws.org/Vol-142/page303.pdf [Retrieved 04/11/2013])

Davis, E. (1992). Axiomatizing Qualitative Process Theory. In B. Nebel, C. Rich, & W. Swartout (Eds.), *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (p. 177-188). San Francisco, CA: Morgan Kaufmann.

Davis, E. (1999). *Guide to Axiomatizing Domains in First-Order Logic.* [Online] https://cs.nyu.edu/faculty/davise/guide.html [Retrieved 04/06/2013].

Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., & Pfeifer, G. (2003). Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (pp. 847–852). San Francisco, CA: Morgan Kaufmann.

De Moura, L., & Bjørner, N. (2011, September). Satisfiability Modulo Theories: Introduction and Applications. *Communications of the ACM*, *54*(9), 69–77.

Doherty, P., & Kvarnstrm, J. (2008). Temporal Action Logics. In V. L. Frank van Harmelen & B. Porter (Eds.), *Handbook of Knowledge Representation* (Vol. 3, p. 709 - 757). San Diego, CA: Elsevier.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2005). A Uniform Integration of Higher-order Reasoning and External Evaluations in Answer-set Programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (pp. 90–96). San Francisco, CA: Morgan Kaufmann.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2006). DLVHEX: A Prover for Semantic-Web Reasoning under the Answer-Set Semantics. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence* (pp. 1073–1074). Washington, DC, USA: IEEE Computer Society.

Erdem, E., & Gabaldon, A. (2005). Cumulative Effects of Concurrent Actions on Numeric-valued Fluents. In *Proceedings of the 20th National Conference on Artificial Intelligence* (pp. 627–632). Menlo Park, CA: AAAI.

Erdem, E., & Gabaldon, A. (2006). Representing Action Domains with Numeric-Valued Fluents. In M. Fisher, W. van der Hoek, B. Konev, & A. Lisitsa (Eds.), *Logics in Artificial Intelligence* (Vol. 4160, p. 151-163). Berlin, Heidelberg: Springer.

Evans, L. C. (2010). *Partial Differential Equations: Second Edition.* Providence, RI: American Mathematical Society.

Faber, W., Leone, N., & Pfeifer, G. (2004). Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)* (pp. 200–212). Berlin, Heidelberg: Springer-Verlag.

Faber, W., Pfeifer, G., Leone, N., Dell'armi, T., & Ielpa, G. (2008, November). Design and Implementation of Aggregate Functions in the DLV System*. *Theory and Practice of Logic Programming*, *8*(5-6), 545–580.

Ferraris, P., Lee, J., & Lifschitz, V. (2007). A New Perspective on Stable Models. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence* (pp. 372–379). San Francisco, CA: Morgan Kaufmann.

Ferraris, P., Lee, J., & Lifschitz, V. (2011). Stable Models and Circumscription. *Artificial Intelligence*, *175*, 236–263.

Ferraris, P., Lee, J., Lifschitz, V., & Palla, R. (2009). Symmetric Splitting in the General Theory of Stable Models. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence* (pp. 797–803). San Francisco, CA:

Morgan Kaufmann.

Ferraris, P., & Lifschitz, V. (2010). On the Stable Model Semantics of First-Order Formulas with Aggregates. In *Proceedings of the 2010 Workshop on Nonmonotonic Reasoning.* ([Online] http://www.cs.sfu.ca/news/conferences/NMR/2010/NMR_2010/Accepted_Papers.html [Retrieved 04/10/2013])

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence* (pp. 608–620). San Francisco, CA, USA: Morgan Kaufmann.

Fitting, M. (2002). Fixpoint Semantics for Logic Programming a Survey. *Theoretical Computer Science*, *278*(12), 25 - 51.

Forbus, K. D. (1984). Qualitative Process Theory. *Artificial Intelligence*, *24*, 85–168.

Fox, M., & Long, D. (2003, December). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, *20*(1), 61–124.

Fox, M., & Long, D. (2006, October). Modelling Mixed Discrete-Continuous Domains for Planning. *Journal of Artificial Intelligence Research*, *27*(1), 235–297.

Gabbay, D. M., & Ohlbach, H. J. (1992). Quantifier Elimination in Second–order Predicate Logic. In B. Nebel, C. Rich, & W. Swartout (Eds.), *Principles of Knowledge Representation and Reasoning (KR92)* (pp. 425–435). San Francisco, CA: Morgan Kaufmann.

Galton, A. (2008). Temporal logic. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy.* Stanford, CA: The Metaphysics Research Lab Center for the Study of Language and Information Stanford University. ([Online] http://plato.stanford.edu/archives/fall2008/entries/logic-temporal/ [Retrieved 04/10/2013])

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., & Schneider, M. (2011, April). Potassco: The Potsdam Answer Set Solving Collection. *AI Communications - Answer Set Programming*, *24*(2), 107–124.

Gelfond, M. (2008). Answer Sets. In V. L. Frank van Harmelen & B. Porter (Eds.), *Handbook of Knowledge Representation* (Vol. 3, p. 285 - 316). San Diego, CA: Elsevier.

Gelfond, M., & Lifschitz, V. (1998). Action Languages. *Electronic Transactions on Artificial Intelligence*, *3*.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (2004). Nonmonotonic Causal Theories. *Artificial Intelligence*, *153*(12), 49 - 104.

Goldberg, D. (1991, March). What Every Computer Scientist Should Know About Floating-point Arithmetic. *ACM Computing Surveys (CSUR)*, *23*(1), 5–48.

Grädel, E. (1992). On Transitive Closure Logic. In E. Börger, G. Jäger, H. Kleine Bning, & M. Richter (Eds.), *Computer Science Logic* (Vol. 626, p. 149-163). Berlin Heidelberg: Springer.

Güémez, J., Valiente, R., Fiolhais, C., & Fiolhais, M. (2003). Experiments with the

Drinking Bird. *American Journal of Physics*, *71*(12), 1257-1263.

Hella, L., Libkin, L., Nurmonen, J., & Wong, L. (2001, July). Logics with Aggregate Operators. *Journal of the ACM (JACM)*, *48*(4), 880–907.

Henzinger, T. (1996). The Theory of Hybrid Automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science. Invited Tutorial* (pp. 278–). Washington DC, USA: IEEE Computer Society.

Henzinger, T., Ho, P.-H., & Wong-Toi, H. (1997). HyTech: A Model Checker for Hybrid Systems. In O. Grumberg (Ed.), *Computer Aided Verification* (Vol. 1254, p. 460-463). Berlin Heidelberg: Springer.

Herrmann, C. S., & Thielscher, M. (1996). Reasoning about Continuous Processes. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1* (pp. 639–644). Menlo Park, CA: AAAI.

Heymans, S., & Toma, I. (2008). Ranking Services Using Fuzzy HEX Programs. In D. Calvanese & G. Lausen (Eds.), *Web Reasoning and Rule Systems* (Vol. 5341, p. 181-196). Berlin Heidelberg: Springer.

Jaffar, J., & Maher, M. J. (1994). Constraint Logic Programming: A Survey. *The Journal of Logic Programming*, *1920, Supplement 1*(0), 503 - 581.

Khandelwal, A. (2013a). *CEC Domain Axioms in DLVHEX-Compatible Syntax.* [Online] https://www.dropbox.com/sh/s4oedu5s80227g9/Hr7XsK4bSS/include/cecaxioms.hpp [Retrieved 04/10/2013].

Khandelwal, A. (2013b). *Code for CEC Model Builder.* [Online] https://www.dropbox.com/sh/s4oedu5s80227g9/TSISHbrM0s [Retrieved 04/10/2013].

Khandelwal, A., & Fox, P. (2012). *Towards A Web Ontology Language for Descriptions of Continuous Processes.* http://www.cs.rpi.edu/~ankesh/wolcc.pdf [Retrieved 04/06/2013].

Kim, T.-W., Lee, J., & Palla, R. (2009). Circumscriptive Event Calculus as Answer Set Programming. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence* (pp. 823–829). San Francisco, CA: Morgan Kaufmann.

Kowalski, R., & Sergot, M. (1986, January). A Logic-based Calculus of Events. *New Generation Computing*, *4*(1), 67–95.

Kuipers, B. (2001). Qualitative Simulation. *Artificial Intelligence*, *29*, 289–338.

Lee, J., & Lifschitz, V. (2003). Describing Additive Fluents in Action Language $\mathcal{C}+$. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (pp. 1079–1084). San Francisco, CA: Morgan Kaufmann.

Lee, J., & Meng, Y. (2009). On Reductive Semantics of Aggregates in Answer Set Programming. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning* (pp. 182–195). Berlin, Heidelberg: Springer-Verlag.

Lee, J., & Palla, R. (2009a). *More Event Calculus Examples and Comparison with DEC Reasoner.* [Online] http://reasoning.eas.asu.edu/f2lp/index_files/EventCalculus.html [Retrieved 04/10/2013].

Lee, J., & Palla, R. (2009b). System `F2LP` Computing Answer Sets of First-Order Formulas. In E. Erdem, F. Lin, & T. Schaub (Eds.), *Logic Programming*

and Nonmonotonic Reasoning (Vol. 5753, p. 515-521). Berlin Heidelberg: Springer.

Lee, J., & Palla, R. (2012a). Reformulating Temporal Action Logics in Answer Set Programming. In J. Hoffmann & B. Selman (Eds.), *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence.* Menlo Park, CA: AAAI.

Lee, J., & Palla, R. (2012b, April). Reformulating the Situation Calculus and the Event Calculus in the General Theory of Stable Models and in Answer Set Programming. *Journal of Artificial Intelligence Research*, *43*(1), 571–620.

Lidström, P. (1966). First-order Predicate Logic with Generalized Quantifiers. *Theoria*, *32*, 186–195.

Lifschitz, V. (1987). Pointwise Circumscription. In *Readings in Non-monotonic Reasoning* (pp. 179–193). San Francisco, CA: Morgan Kaufmann.

Lifschitz, V. (1994). Circumscription. In *Handbook of Logic in Artificial Intelligence and Logic Programming (vol. 3).* New York, NY: Oxford University Press, Inc.

Lifschitz, V. (1995). Nested Abnormality Theories. *Artificial Intelligence*, *74*, 351–365.

Lifschitz, V., Morgenstern, L., & Plaisted, D. (2008). Knowledge Representation and Classical Logic. In Frank van Harmelen and Vladimir Lifschitz and Bruce Porter (Ed.), *Handbook of Knowledge Representation* (Vol. 3, p. 3 - 88). San Diego, CA: Elsevier.

Lin, F. (2008). Situation Calculus. In Frank van Harmelen and Vladimir Lifschitz and Bruce Porter (Ed.), *Handbook of Knowledge Representation* (Vol. 3, p. 3 - 88). San Diego, CA: Elsevier.

Lin, F., & Reiter, R. (1994). State Constraints Revisited. *Journal of Logic and Computation*, *4*(5), 655-677.

McCain, N., & Turner, H. (1997). Causal Theories of Action and Change. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence* (pp. 460–465). Menlo Park, CA: AAAI.

McCarthy, J. (1980). Circumscription - A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, *13*, 27–39.

McDermott, D. (2003). The Formal Semantics of Processes in PDDL. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) Workshop on PDDL* (p. 87-94). Menlo Park, CA: AAAI.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., et al. (1998). *PDDL – The Planning Domain Definition Language* (Tech. Rep. No. CVC TR-98-003/DCS TR-1165). New Haven, CT: Yale Center for Computational Vision and Control.

Miller, R. (1996). A Case Study in Reasoning about Actions and Continuous Change. In *European Conference on Artificial Intelligence (ECAI).* Chichester, UK: John Wiley & Sons.

Miller, R., & Shanahan, M. (1996). Reasoning about Discontinuities in the Event

Calculus. In *Principles of Knowledge Representation and Reasoning (KR).* San Francisco, CA: Morgan Kaufmann.

Miller, R., & Shanahan, M. (2002). Some Alternative Formulations of the Event Calculus. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II.* Berlin, Germany: Springer.

Miyaji, C., & Abbott, P. (2001). *Mathlink Network Programming in Mathematica.* New York, NY, USA: Cambridge University Press.

Mueller, E. T. (2004). Event Calculus Reasoning Through Satisfiability. *Journal of Logic and Computation*, *14*(5), 703–730.

Mueller, E. T. (2005). Reasoning in the Event Calculus using First-Order Automated Theorem Proving. In *Proceedings of the 18th Florida Artificial Intelligence Research Symposium.* Menlo Park, CA: AAAI.

Mueller, E. T. (2006). *Commonsense Reasoning.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Mueller, E. T. (2008). Event Calculus. In Frank van Harmelen and Vladimir Lifschitz and Bruce Porter (Ed.), *Handbook of Knowledge Representation* (Vol. 3, p. 3 - 88). San Diego, CA: Elsevier.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, *77*(4), 541-580.

Neumaier, A. (2004). Complete Search in Continuous Global Optimization and Constraint Satisfaction. *Acta Numerica*, *13*, 271–369.

Perlis, D. (1988). Autocircumscription. *Artificial Intelligence*, *36*, 22323–6.

Pinto, J. A. (1998). Integrating Discrete and Continuous Change in a Logical Framework. *Computational Intelligence*, *14*(1), 39–88.

Przymusinski, T. C. (1989). On the Declarative and Procedural Semantics of Logic Programs. *Journal of Automated Reasoning*, *5*, 167-205.

Rayner, M. (1991). On the Applicability of Nonmonotonic Logic to Formal Reasoning in Continuous Time. *Artificial Intelligence*, *49*(1–3), 345–360.

Redl, C., Eiter, T., & Krennwallner, T. (2011). Declarative Belief Set Merging Using Merging Plans. In R. Rocha & J. Launchbury (Eds.), *Practical Aspects of Declarative Languages* (Vol. 6539, p. 99-114). Berlin Heidelberg: Springer.

Reiter, R. (1993, December). Proving Properties of States in the Situation Calculus. *Artificial Intelligence*, *64*(2), 337–351.

Reiter, R. (1996). Natural Actions, Concurrency and Continuous Time in the Situation Calculus. In *Principles of Knowledge Representation and Reasoning (KR)* (pp. 2–13). Cambridge, MA: Morgan Kaufmann.

Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* Cambridge, MA: MIT Press.

Robinson, A., & Voronkov, A. (Eds.). (2001). *Handbook of Automated Reasoning.* Amsterdam, Netherlands: Elsevier Science.

Sandewall, E. (1989a). Combining Logic and Differential Equations for Describing Real-world Systems. In *Principles of Knowledge Representation and Reasoning (KR)* (pp. 412–420). San Francisco, CA: Morgan Kaufmann.

Sandewall, E. (1989b). Filter Preferential Entailment for the Logic of Action in

almost Continuous Worlds. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2.* San Francisco, CA: Morgan Kaufmann.

Shanahan, M. (1990). Representing Continuous Change in the Event Calculus. In L. C. Aiello (Ed.), *Proceedings of the 9th European Conference on Artificial Intelligence* (p. 598-603). London, UK: Pitman.

Shanahan, M. (1997). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. Cambridge, MA: MIT Press.

Shanahan, M. (1999a). The Event Calculus Explained. In M. J. Wooldridge & M. Veloso (Eds.), *Artificial Intelligence today* (pp. 409–430). Berlin, Heidelberg: Springer-Verlag.

Shanahan, M. (1999b). The Ramification Problem in the Event Calculus. In *Proceedings of the 16th International Joint Conference on Artificial intelligence - Volume 1* (pp. 140–146). San Mateo, CA: Morgan Kaufmann.

Shanahan, M. (2000). An Abductive Event Calculus Planner. *Journal of Logic Programming*, *44*, 207–239.

Shanahan, M., & Witkowski, M. (2004, October). Event Calculus Planning Through Satisfiability. *Journal of Logic and Computation*, *14*(5), 731–745.

Teschl, G. (2012). *Ordinary Differential Equations and Dynamical Systems*. Providence, RI: American Mathematical Society.

Thielscher, M. (2001). The Concurrent, Continuous Fluent Calculus. *Studia Logica*, *67*, 315-331.

Westerståhl, D. (2008). Generalized Quantifiers. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy.* Stanford, CA: The Metaphysics Research Lab Center for the Study of Language and Information Stanford University. ([Online] http://plato.stanford.edu/archives/win2008/entries/generalized-quantifiers/ [Retrieved 04/10/2013])

APPENDIX A
EQUATIONS REASONING USING MATHEMATICA

We give a sample input to mathematica for determining trajectories by solving differential algebraic equations and determining next landmark from the computed trajectories in Section A.1. Then we describe the $FindNextT$ package used to compute next landmarks by constructing equations from given mathematical conditions and finding earliest change in their truth value in Section A.2.

A.1 Example Input to Mathematica for Determining Trajectories and Next Landmark

This is a sample input to `Mathematica` for determining trajectories after time 2 (when outlet of tank C is opened) in the layered water tanks example (Section 5.7). This code determines the next landmark point as 2.27 when tank A overflows.

```
maxtime=4;
currenttime=2;
(*Initial values for solving DAEs*)
qid1AT0=20;qid2AT0=15;qid3AT0=15;
qid4AT0=12.969970751450811;qid5AT0=2.0300292485491904;qid6AT0=15;
qid7AT0=4.0600584970983782;qid8AT0=-4.0600584970983808;qid9AT0=-30;
(*Define derivative relationships*)
qid7=Derivative[1][qid4];qid8=Derivative[1][qid5];
qid9=Derivative[1][qid6];

(*Module to solve DAEs*)
solveDAEs := Module[{},
        sol=Check[DSolve[
          (*DAEs*)
          {qid9[t]==(-2*qid6[t]),qid8[t]==(-2*qid5[t]),
```

```
                    qid7[t]==(2*qid5[t])+(2*qid6[t]),
                (*Initial values*)
                qid1[t]==qid1AT0,qid2[t]==qid2AT0,qid3[t]==qid3AT0,
                qid8[0]==qid8AT0, qid6[0]==qid6AT0, qid5[0]==qid5AT0,
                qid4[0]==qid4AT0, qid3[0]==qid3AT0, qid2[0]==qid2AT0,
                qid1[0]==qid1AT0, qid9[0]==qid9AT0, qid7[0]==qid7AT0},
                (*base quantity functions*)
                {qid1, qid2, qid3, qid4, qid5, qid6},
                (*Independent variable*)
                t],
                daesolveerr];
        If[sol==daesolveerr,Return[{-1,daesolveerr}]];
        If[Length[sol]==0,Return[{-1,nosolution}]];
        If[Length[sol]>1,Return[{-1,manysolutions}]];
        (*Get functions defining the trajectories*)
        fOFqid1[t_]:=qid1[t]/.sol[[1]];fOFqid2[t_]:=qid2[t]/.sol[[1]];
        fOFqid3[t_]:=qid3[t]/.sol[[1]];fOFqid4[t_]:=qid4[t]/.sol[[1]];
        fOFqid5[t_]:=qid5[t]/.sol[[1]];fOFqid6[t_]:=qid6[t]/.sol[[1]];
        fOFqid7[t_]:=fOFqid4'[t];fOFqid8[t_]:=fOFqid5'[t];
        fOFqid9[t_]:=fOFqid6'[t];
        Return[{1}]];

(*Module that checks if solution satisfies constraints and
finds next landmark*)
findLandmark := Module[{},
        (*Check if constraints satisfied. In this example constraints
                are empty, so we have just True*)
        If[!True,Return[{-1,constraintsviolated}]];
        NextT=maxtime-currenttime;
        (*Find next time using FindNextT function over "ChkCnd"s*)
        NextT=FindNextT[(t+currenttime)-1==0,t];
```

```
        NextT=FindNextT[(t+currenttime)-2==0,t];
        NextT=FindNextT[fOFqid6[t]-0>0&&fOFqid4[t]-fOFqid1[t]==0,t];
        NextT=FindNextT[fOFqid5[t]-0>0&&fOFqid4[t]-fOFqid1[t]==0,t];
        NextT=FindNextT[fOFqid4[t]-fOFqid1[t]<0,t];
        landmark=currenttime+NextT;
        landmark=If[landmark==maxtime,maxtime,landmark];
        isMaxTime=If[landmark==maxtime,1,0];
        (*Get values of quantities at next landmark*)
        Map[Composition[Chop, CompactForm],
                {landmark,isMaxTime,fOFqid1[t],fOFqid2[t],
                fOFqid3[t],fOFqid4[t],fOFqid5[t],fOFqid6[t],
                fOFqid7[t],fOFqid8[t],fOFqid9[t]}/.t->NextT]];

(*FindNextT and CompactForm functions are defined in
FindNextT.m package"*)
Get["/home/ankesh/codes/mathematica/FindNextT.m"];
sol1 = solveDAEs;
If[sol1[[1]]!=-1,sol2= findLandmark];
If[sol1[[1]]==-1,sol1,sol2]
```

A.2 FindNextT Package for Finding Next Landmark

The next landmark is determined by solving for the earliest change in the truth value of mathematical conditions such as $(t + currenttime) - 2 == 0$ and $fOFqid6[t] - 0 > 0 \&\& fOFqid4[t] - fOFqid1[t] == 0$, with help of the $FindNextT$ function defined in the $FindNextT$ package.

```
(* ::Package:: *)

BeginPackage["FindNextT`",{"Global`"}]

FindNextT::usage =
        "FindNextT[x] gives the earliest time the constraint x would
```

```
        change sign, i.e. it would change from true to false or vice-
        versa."

CompactForm::usage=
        "CompactForm[x] returns an approximation for numbers that are
        given as roots"

numGT0::usage=""
NextT::usage=
        "NextT can be set to the current known value of NextT, so that
        FindNextT searches only for smaller time."
t::usage=""

Begin["`Private`"]

numGT0=2^8*10^-16;

(* To construct equations from inequalities *)
getEquation[x_]:=x-NextT==0;x===False
getEquation[x_]:=Part[x,1]==0/;
                MemberQ[{GreaterEqual,LessEqual,Equal},Head[x]];
getEquation[x_]:=Part[x,1]-numGT0==0/;Head[x]===Greater;
getEquation[x_]:=Part[x,1]+numGT0==0/;Head[x]===Less;
getEquation[x_]:=(Part[x,1]-numGT0==0||Part[x,1]+numGT0==0)/;
                Head[x]===Unequal;

getMinTime[x_]:=If[Length[x]==0,NextT,Min[t/.x]];

solveForNextT[x_]:=getMinTime[Solve[x&&numGT0<t<NextT,t]];

constraintsList[x_]:=Level[x,{1}];
```

```
allRotations[x_]:=Map[RotateLeft[x,#]&,Range[Length[x]]];

changeFirst2Eqn[x_]:=Map[Apply[And,#]&,
                  Map[ReplacePart[#,1->getEquation[Part[#,1]]]&,x]];

findNextTWhenFalse[x_]:=Min[Map[
                  Composition[solveForNextT,getEquation,Not],
                              constraintsList[x]]]/;Head[x]===And;
findNextTWhenFalse[x_]:=solveForNextT[getEquation[Not[x]]]/;
                              Not[Head[x]===And];

findNextTWhenTrue[x_]:=Min[Map[solveForNextT,
               changeFirst2Eqn[allRotations[constraintsList[x]]]]]/;
                                                  Head[x]===And;
findNextTWhenTrue[x_]:=solveForNextT[getEquation[x]]/;
                                                Not[Head[x]===And];

FindNextT[x_,t_]:=If[x/.t->0,
                        findNextTWhenFalse[x],
                        findNextTWhenTrue[x]];
(*FindNextT[x_,t_]:=findNextTWhenTrue[x]/;Not[x/.t->0];*)

Real2Integer[x_]:=If[Head[x]===Real&&Abs[FractionalPart[x]]<2^-7,
                        IntegerPart[x],
                        If[Head[x]===Real&&FractionalPart[x]==1.,
                                IntegerPart[x]+1,N[x]]]

CompactForm[x_]:=If[Head[x]===Real,
                        Real2Integer[x],
                        If[Head[x]===Integer,x,Real2Integer[N[x]]]];
```

```
End[]
EndPackage[]
```

APPENDIX B
SCOPE AND LIMITATIONS OF THE ENHANCEMENTS TO THE EVENT CALCULUS

We have made two main enhancements to Miller and Shanahan's (1996) Event Calculus. We have made a theoretical extension for efficient descriptions of additive effects and we have implemented a model builder for Event Calculus descriptions.[22] Miller and Shanahan (1996) claimed that continuous-changes can be described by arbitrary (autonomous) ordinary differential equations. Assuming that to be the case, we discuss the scope of the enhancements and limitations imposed specifically in our work for the two enhancements in Sections B.1 and B.2 respectively.

B.1 The Theoretical Extension

We introduce new relations to specify additive effects. $BreaksPartBy(A, P, T, R)$ and $PartValue(P, T, PX, R)$ can be used to represent additive effects from different sources. Discrete changes are caused by actions directly and each action is a unique source of additive effect. For continuous changes, PX is unique for each source of the additive effect. In the discrete case, R may be defined by any mathematical expression that evaluates to a real value. In the continuous case, $Value(P, t) = R$ must denote a well-founded ordinary differential equation. Besides that general principles of axiomatizing domains in first-order logic (for example, Davis, 1999) can be applied to represent additive effects systematically.

In general any quantity can be subject to four types of effects – discrete nonadditive effects, discrete additive effects, continuous nonadditive effects, and continuous additive effects –, typically under different conditions. If at any time values for a

[22] Another crucial enhancement is that we introduced a default assumption that *by default quantities are constant if continuous*. Miller and Shanahan's (1996) formalism includes the default assumption that by default quantities are continuous, but in absence of specific equations for change, quantities can be continuous and change along many different paths. Crucially, when a quantity is not subject to any change, and thereby is constant, in a duration, an equation stating so has to be explicitly specified, for example as $Value(\delta(Q), t) = 0$, for some quantity Q that is constant in a duration. Specification of such equations can not only become tedious and increase the complexity of axiomatic descriptions, intuitively they must hold implicitly/automatically.

quantity in a domain are given by more than one type of effect, the domain description is consistent if and only if the values from the two effects are the same. That is, the different type of effects define complete effects in themselves, and those effects cannot be combined. For instance, each $BreaksTo(a, P, T, r)$, for different a, r, yields a right limit value for quantity P at time T and all $BreaksPartBy(a, P, T, r')$, for any a, r', yield a right limit value for quantity P at time T. Similarly, each $Value(P, T) = r$, for different r, yields the value for quantity P at time T and all $PartValue(P, T, px, r')$, for any px, r', yield the value for quantity P at time T.

Finally, $BreaksPartBy$ relation is not sufficient to describe all types of discrete additive effects, and other relations such as $BreaksPartMinBy$, to represent range restrictions on the cumulative effect, and $BreaksPartTo$, to represent change in the value relative to 0, may be used.

B.2 The Model Builder

We designed an interleaving method of constructing models using the separate logic reasoning and equations solving for finite, given, and numerical domains, for an initial state and narratives of external actions. The designed method naturally supports simulation or temporal projection and can be adapted to work backwards from a given final state to deduce the initial state given a narratives of external actions. Other reasoning tasks can be performed as long as they can be interpreted via simulation or temporal projections. For example, for the planning tasks, if an oracle could produce candidate narratives of external actions then the devised method can be used to verify which of the narratives provided by the oracle produce the given final state from the given initial state.

Since the domains must be numeric, symbolic reasoning is ruled out (that is, constant symbols different from real numbers for denoting the values of quantities cannot be used). Likewise, qualitative values or interval of values are also ruled out. Furthermore, since the domain must be given, at any point of discontinuity the new value after the discontinuity must be specified and enough values for a quantity and its derivatives must be known after the discontinuity so that the exact trajectory of the quantity after the discontinuity can be computed. See Section 5.2 for details.

The advantage of the devised method is that it can be implemented by stringing together off-the-shelf logic reasoners and equation solvers (without modifying the reasoner or the solver). In theory, a reasoner for any given, finite, and numeric Event Calculus domains can be implemented based on the method devised above. In practice there are some restrictions however. For example, there are a very few reasoners that support aggregate formulas. Likewise, many reasoners do not support nondeterministic conclusions. We figured that `DLVHEX` is probably the most expressive of available logic reasoners for building models for Event Calculus descriptions. `DLVHEX` builds on a fairly robust `DLV` reasoner, but the support for external atoms is only prototypical. For example, in the multiple blocks problem `DLVHEX` unexpectedly diagnosed unsafe variables; see Appendix C.6. Furthermore, `DLVHEX` does not support recursive rules involving higher-order external atoms as we discussed in Section 5.6. As a result additive parameters with interdependencies are only supported if the interdependency is strictly acyclic (strictly acyclic in the sense that dependencies are fixed for all times and they are acyclic). We believe this restriction of strict acyclic interdependency is not too restrictive. However, to support acyclic interdependencies many relations $partValue\_n$ and $value\_n$ for $n = 1, 2, \ldots$ are introduced, and appropriate relations must be carefully picked for given additive quantities depending on the dependencies on other additive quantities. Note that this process can be automated in many scenarios. That is, users may axiomatize domains using $partValue$ and $value$ relations which can be replaced by appropriate $partValue\_n$ and $value\_n$ relations in a preprocessing step.

Neither have we found a solution to the unexpected unsafe variable diagnosis of `DLVHEX` nor can we characterize a class of descriptions for which such a diagnosis may be expected or not expected. But once the problem is resolved, it would increase the scope of problems to which our implementation can be successfully applied. For instance we can then find models for the multiple blocks problem like other problems listed in Appendix C.

APPENDIX C
EXAMPLE DOMAIN DESCRIPTIONS

C.1 Inelastic Collision of Two Balls in 2D

Figure 5.1 depicts two balls placed at points A and B which are set in motion at different times. They eventually collide inelastically and move together. The first ball is placed at A with coordinates $(-10, -10)$ and the second is placed at B with coordinates $(20, 0)$. The first ball is set in motion with velocity $(10, 10)$ at time 1, and the second ball is set in motion with velocity $(-10, 10)$ at time 2. The balls collide when they are at the same coordinates and the collision breaks their velocities such that both move in the same direction at the average of their velocities right before the collision. The domain description in F2LP syntax is given below.

```
parameter_2(pos,b1,x,pid1). parameter_2(pos,b1,y,pid2). derivative(pid3,1,pid1).
derivative(pid4,1,pid2). derivative(pid5,2,pid1). derivative(pid6,2,pid2).
parameter_2(pos,b2,x,pid7). parameter_2(pos,b2,y,pid8).
derivative(pid9, 1, pid7). derivative(pid10, 1, pid8).
derivative(pid11, 2, pid7). derivative(pid12, 2, pid8).
action_3(setinmotion,b1,10,10,aid1). action_3(setinmotion,b2,"-10",10,aid2).
action_2(collission,b1,b2,aid3).
fluent_1(moving,b1,fid1). fluent_1(moving,b2,fid2). fluent_2(joined,b1,b2,fid3).
initializedFalse(fid1). initializedFalse(fid2). initializedFalse(fid3).
initialValue(pid1,"-10"). initialValue(pid2,"-10"). initialValue(pid3,0).
initialValue(pid4,0). initialValue(pid5,0). initialValue(pid6,0).
initialValue(pid7,20). initialValue(pid8,0). initialValue(pid9,0).
initialValue(pid10,0). initialValue(pid11,0). initialValue(pid12,0).
happens(aid1,1). happens(aid2,2).

action_3(setinmotion,B,V_X,V_Y,AID)& fluent_1(moving,B,FID)
-> initiates(AID,FID,T) .

action_3(setinmotion,B,V_X,V_Y,AID)&parameter_2(pos,B,x,PID)
&derivative(PID1,1,PID)
-> breaksTo(AID,PID1,T,V_X) .
```

```
action_3(setinmotion,B,V_X,V_Y,AID)& parameter_2(pos,B,y,PID)
&derivative(PID1,1,PID)
-> breaksTo(AID,PID1,T,V_Y) .

fluent_1(moving,B,FID)& holdsAt(FID,T)& parameter_2(pos,B,x,PID)
&derivative(PID2,2,PID)
-> value(PID2,T,0).

fluent_1(moving,B,FID)& holdsAt(FID,T)& parameter_2(pos,B,y,PID)
&derivative(PID2,2,PID)
-> value(PID2,T,0).

fluent_1(moving,B1,FID1)& fluent_1(moving,B2,FID2)& B1!=B2& holdsAt(FID1,T)
& holdsAt(FID2,T)&fluent_2(joined,B1,B2,FID3)&not holdsAt(FID3,T)
& parameter_2(pos,B1,x,PID1x)& parameter_2(pos,B2,x,PID2x)
& parameter_2(pos,B1,y,PID1y)& parameter_2(pos,B2,y,PID2y)& value(PID1x,T,V1x)
& value(PID2x,T,V2x)& value(PID1y,T,V1y)& value(PID2y,T,V2y)
& V1x==V2x & V1y==V2y& action_2(collission,B1,B2,AID)
-> happens(AID,T) .

action_2(collission,B1,B2,AID) & fluent_2(joined,B1,B2,FID)
-> initiates(AID,FID,T).

action_2(collission,B1,B2,AID)& happens(AID,T)& parameter_2(pos,B1,x,PID1)
& derivative(PID11,1,PID1)&parameter_2(pos,B2,x,PID2)&derivative(PID21,1,PID2)
&value(PID11,T,V11)& value(PID21,T,V21)& SUM=V11+V21& AVG=SUM/2
-> breaksTo(AID,PID11,T,AVG).

action_2(collission,B1,B2,AID)&happens(AID,T)& parameter_2(pos,B1,x,PID1)
&derivative(PID11,1,PID1)&parameter_2(pos,B2,x,PID2)&derivative(PID21,1,PID2)
&value(PID11,T,V11)& value(PID21,T,V21)& SUM=V11+V21& AVG=SUM/2
-> breaksTo(AID,PID21,T,AVG).

action_2(collission,B1,B2,AID) &happens(AID,T)& parameter_2(pos,B1,y,PID1)
&derivative(PID11,1,PID1)&parameter_2(pos,B2,y,PID2)&derivative(PID21,1,PID2)
&value(PID11,T,V11)& value(PID21,T,V21)& SUM=V11+V21& AVG=SUM/2
-> breaksTo(AID,PID11,T,AVG).
```

```
action_2(collission,B1,B2,AID) &happens(AID,T)& parameter_2(pos,B1,y,PID1)
&derivative(PID11,1,PID1)&parameter_2(pos,B2,y,PID2)&derivative(PID21,1,PID2)
&value(PID11,T,V11)&value(PID21,T,V21)&SUM=V11+V21&AVG=SUM/2
-> breaksTo(AID,PID21,T,AVG).
```

Simulation run for maximum time 4. The collision ($aid3$) happens at time 3 after which the velocities of both are $(0, 10)$.

```
equals(next0,1).equals(next1,2).equals(next2,3).equals(next3,4).

happens(aid1,1).happens(aid2,2).happens(aid3,3).

value(pid1,0,"-10").value(pid2,0,"-10").value(pid3,0,0).value(pid4,0,0).
value(pid5,0,0).value(pid6,0,0).value(pid7,0,20).value(pid8,0,0).
value(pid9,0,0).value(pid10,0,0).value(pid11,0,0).value(pid12,0,0).

value(pid1,1,"-10").value(pid2,1,"-10").value(pid3,1,0).value(pid4,1,0).
value(pid5,1,0).value(pid6,1,0).value(pid7,1,20).value(pid8,1,0).
value(pid9,1,0).value(pid10,1,0).value(pid11,1,0).value(pid12,1,0).
holdsAt(fid1,next1).

value(pid1,2,0).value(pid2,2,0).value(pid3,2,10).value(pid4,2,10).
value(pid5,2,0).value(pid6,2,0).value(pid7,2,20).value(pid8,2,0).
value(pid9,2,0).value(pid10,2,0).value(pid11,2,0).value(pid12,2,0).
holdsAt(fid1,2).holdsAt(fid2,next2).holdsAt(fid1,next2).

value(pid1,3,10).value(pid2,3,10).value(pid3,3,10).value(pid4,3,10).
value(pid5,3,0).value(pid6,3,0).value(pid9,3,"-10").value(pid7,3,10).
value(pid8,3,10).value(pid10,3,10).value(pid11,3,0).value(pid12,3,0).
holdsAt(fid1,3).holdsAt(fid2,3).holdsAt(fid3,next3).holdsAt(fid2,next3).
holdsAt(fid1,next3).

value(pid1,4,10).value(pid2,4,20).value(pid3,4,0).value(pid4,4,10).
value(pid5,4,0).value(pid6,4,0).value(pid7,4,10).value(pid8,4,20).
value(pid9,4,0).value(pid10,4,10).value(pid11,4,0).value(pid12,4,0).
holdsAt(fid1,4).holdsAt(fid2,4).holdsAt(fid3,4).
```

C.2 Discrete, Additive Water Flow in Tanks with Spillage into Lower Tank

Figure 5.2 shows tanks B and C are layered above tank A. The capacities of tanks A, B, and C are 8, 3, and 3 respectively. When the tanks above are filled instantaneously (discretely that is) any extra amount of water spills over to the bottom tank. Initially all the tanks are empty. At time 1 three concurrent actions fill tank B by amounts 1, 1, and 2 respectively. (The ramification is that 1 unit is added to tank A.) At time 2 two actions fill tank C by amounts 2 each. (The ramification is that 1 unit is added to tank A.) The domain description in F2LP syntax is given below.

```
action_3(fill,s1,1,1,aid1).action_3(fill,s1,2,1,aid2).
action_3(fill,s1,3,2,aid3).action_3(fill,s2,1,2,aid4).
action_3(fill,s2,2,2,aid5).
parameter_1(capacity, b, qid1).parameter_1(amount, b, qid2).
parameter_1(capacity, s1, qid3).parameter_1(amount, s1, qid4).
parameter_1(capacity, s2, qid5).parameter_1(amount, s2, qid6).
parameter_1(fillTotal, s1, qid7).parameter_1(fillTotal, s2, qid8).
initialValue(qid1,8). initialValue(qid2,0). initialValue(qid3,3).
initialValue(qid4,0). initialValue(qid5,3). initialValue(qid6,0).
initialValue(qid7,0). initialValue(qid8,0).
happens(aid1, 1). happens(aid2, 1). happens(aid3, 1). happens(aid4, 2).
happens(aid5, 2).

action_3(fill,S,K,R,AID)& parameter_1(fillTotal,S,QID)
-> breaksPartBy_1(AID,QID,T,R).

parameter_1(fillTotal,S,QID1)& parameter_1(amount,S,QID2)
& parameter_1(capacity,S,QID3)& rightLimit_1(QID1,T,R)& value(QID2, T, V2)
& value(QID3, T, V3)& Z=V2+R& Z<= V3 & breaksPartBy_1(A, QID1, T, RA)
-> breaksPartBy_2(A, QID2, T, RA) .

parameter_1(fillTotal,S,QID1)& parameter_1(amount,S,QID2)
& parameter_1(capacity,S,QID3)& rightLimit_1(QID1,T,R)& value(QID2, T, V2)
& value(QID3, T, V3)& Z=V2+R& Z> V3 & breaksPartBy_1(A, QID1, T, RA)
-> breaksTo(A, QID2, T, V3) .
```

```
parameter_1(fillTotal,S,QID1)& parameter_1(amount,S,QID2)
& parameter_1(capacity,S,QID3)& parameter_1(amount, b, QID4)
& rightLimit_1(QID1,T,R)& value(QID2, T, V2)& value(QID3, T, V3)& Z=V2+R
& Z> V3 & breaksPartBy_1(A, QID1, T, RA)& V= Z-V3
& numAdditiveEffects_1(QID1,T,N)& VByN=V/N
-> breaksPartBy_2(A, QID4, T, VByN) .
```

Simulation run for maximum time 4. 1 unit each spills over to the lower, bigger tank at times 1 and 2.

```
happens(aid3,1).happens(aid1,1).happens(aid2,1).happens(aid4,2).happens(aid5,2).
equals(next0,1).equals(next1,2).equals(next2,4).
value(qid1,0,8).value(qid2,0,0).value(qid3,0,3).value(qid4,0,0).value(qid5,0,3).
value(qid6,0,0).value(qid7,0,0).value(qid8,0,0).
value(qid1,1,8).value(qid2,1,0).value(qid3,1,3).value(qid4,1,0).value(qid5,1,3).
value(qid6,1,0).value(qid7,1,0).value(qid8,1,0).
value(qid1,2,8).value(qid2,2,1).value(qid3,2,3).value(qid4,2,3).value(qid5,2,3).
value(qid6,2,0).value(qid7,2,4).value(qid8,2,0).
value(qid1,4,8).value(qid2,4,2).value(qid3,4,3).value(qid4,4,3).value(qid5,4,3).
value(qid6,4,3).value(qid7,4,4).value(qid8,4,4).
```

C.3 Outflow from Upper Tanks to Lower Tanks

Figure 5.2 shows tanks B and C have outlets, and they are above tank A. The capacities of tanks A, B and C are 20, 15 and 15 respectively. Tanks B and C are full initially and tank A is empty. The outlet of B is opened at time 1, and that of C is opened at time 2. The rate of outflows are proportional to the amount of water in the tank, and the constant of proportionality is 2. While outlets of the above tanks are open and the above tanks are non-empty they fill tank A. When tank A is full an overflow action is triggered which breaks the rate of change of amount of water in A to 0. The domain description in F2LP syntax is given below.

```
above(b,a). above(c,a).
action_1(openout,b,aid1). action_1(openout,c,aid2). action_1(overflow,a,aid3).
parameter_1(capacity,a,qid1). parameter_1(capacity,b,qid2).
parameter_1(capacity,c,qid3). parameter_1(amount,a,qid4).
```

```
parameter_1(amount,b,qid5). parameter_1(amount,c,qid6).
derivative(qid7,1,qid4). derivative(qid8,1,qid5). derivative(qid9,1,qid6).
fluent_1(open,a,fid1). fluent_1(outopen,b,fid2). fluent_1(outopen,c,fid3).
initializedTrue(fid1). initializedFalse(fid2). initializedFalse(fid3).
initialValue(qid1,20). initialValue(qid2,15). initialValue(qid3,15).
initialValue(qid4,0). initialValue(qid5,15). initialValue(qid6,15).
derivative(Q,N,B)-> initialValue(Q,0).
happens(aid1,1). happens(aid2,2).

action_1(openout, B, AID)& fluent_1(outopen, B, FID) -> initiates(AID, FID, T).

fluent_1(outopen,B,FID)& holdsAt(FID,T)& parameter_1(amount,B,QID)
& derivative(QID1,1,QID)& value(QID,T,V)& Z="-2"*V
-> value(QID1,T,Z).

above(B,A)& fluent_1(outopen,B,FID)& holdsAt(FID,T)& parameter_1(capacity,A,QCAP)
& parameter_1(amount,A,QAMT)& value(QCAP,T,VCAP)& value(QAMT,T,VAMT)& VAMT<VCAP
& derivative(QAMT1,1,QAMT)& parameter_1(amount,B,QID)& value(QID,T,V)& Z=2*V
-> partValue_1(QAMT1,T,B,Z).

above(B,A)& fluent_1(outopen,B,FID)& holdsAt(FID,T)& parameter_1(amount,B,QID)
& value(QID,T,V)& V>0& parameter_1(capacity,A,QCAP)& parameter_1(amount,A,QAMT)
& value(QCAP,T,VCAP)& value(QAMT,T,VAMT)& VAMT==VCAP& action_1(overflow,A,AID)
-> happens(AID, T).

action_1(openout, B, AID)& parameter_1(amount,B,QID)& derivative(QID1,1,QID)
-> breaks(AID,QID1,T).

above(B,A)& action_1(openout, B, AID)& parameter_1(amount,A,QID)
& derivative(QID1,1,QID)
-> breaks(AID,QID1,T).

action_1(overflow,A,AID)& parameter_1(amount,A,QID) & derivative(QID1,1,QID)
-> breaksTo(AID,QID1,T,0).
```

Simulation run for maximum time 4. The tank A overflows at time 2.27.

```
equals(next0,1).equals(next1,2).equals(next2,"2.2661965595755684").
```

```
equals(next3,4).
happens(aid1,1).happens(aid2,2).happens(aid3,"2.2661965595755684").

holdsAt(fid1,0).holdsAt(fid1,1).holdsAt(fid1,2).holdsAt(fid2,2).
holdsAt(fid1, "2.2661965595755684").holdsAt(fid2, "2.2661965595755684").
holdsAt(fid3, "2.2661965595755684").
holdsAt(fid1,4).holdsAt(fid2,4).holdsAt(fid3,4).

rightLimit_1(qid7,1,30).rightLimit_1(qid7,2,"34.060058497098382").

value(qid1,0,20).value(qid2,0,15).value(qid3,0,15).value(qid4,0,0).
value(qid5,0,15).value(qid6,0,15).value(qid7,0,0).value(qid8,0,0).
value(qid9,0,0).value(qid1,1,20).value(qid2,1,15).value(qid3,1,15).
value(qid4,1,0).value(qid5,1,15).value(qid6,1,15).value(qid7,1,0).
value(qid8,1,0).value(qid9,1,0).value(qid1,2,20).value(qid2,2,15).
value(qid3,2,15).value(qid4,2,"12.969970751450811").
value(qid5,2,"2.0300292485491904").value(qid6,2,15).
value(qid7,2,"4.0600584970983782").value(qid8,2,"-4.0600584970983808").
value(qid9,2,0).value(qid1,"2.2661965595755684",20).
value(qid2,"2.2661965595755684",15).value(qid3,"2.2661965595755684",15).
value(qid4,"2.2661965595755684",20).
value(qid5,"2.2661965595755684","1.1920292202211753").
value(qid6,"2.2661965595755684","8.8079707797788238").
value(qid7,"2.2661965595755684",20).
value(qid8,"2.2661965595755684","-2.3840584404423506").
value(qid9,"2.2661965595755684","-17.615941559557648").value(qid1,4,20).
value(qid2,4,15).value(qid3,4,15).value(qid4,4,20).
value(qid5,4,"0.037181282649995366").value(qid6,4,"0.27473458333101269").
value(qid7,4,0).value(qid8,4,"-0.074362565299990732").
value(qid9,4,"-0.54946916666202539").
```

C.4 Falling Object Bouncing off the Ground

A ball is at a height above ground. It is thrown at some point. Determine when it bounces off the ground. Collision is totally inelastic and ball comes to a standstill. We can modify this so that collision is partially inelastic. In that case, ball bounces back travels upward and then starts falling, finally bouncing off the ground.

A similar example of falling object is discussed by Mueller (2006), and implemented by Lee and Palla (2012)[23]. Lee and Palla (2009) assume that the height changes at a constant rate whereas we assume that the object falls under the influence of gravity by gravity. Besides, instead of using *Trajectory* and *AntiTrajectory*, we use differential equations to describe the continuous-change.

```
action_2(drop, nathan, apple, aid1) . action_1(hitGround, apple, aid2) .
fluent_1(falling, apple, fid1) .
parameter_1(height, apple, pid1) .
derivative(pid2, 1, pid1) . derivative(pid3, 2, pid1) .
initializedFalse(fid1) .
initialValue(pid1, 3) .
derivative(DPID, N, pid1) -> initialValue(DPID, 0) .
action_2(drop, nathan, apple, AID)
-> happens(aid1, 0) .

action_2(drop, Agent, Object, AID) & fluent_1(falling, Object, FID)
-> initiates(AID, FID, T) .
action_1(hitGround, Object, AID) & fluent_1(falling, Object, FID)
-> terminates(AID, FID, T) .
fluent_1(falling, Object, FID) & holdsAt(FID, T)
& parameter_1(height, Object, PID) & value(PID, T, 0)
& action_1(hitGround, Object, AID)
-> happens(AID, T) .
parameter_1(height, Object, PID) & derivative(DPID, 2, PID)
& action_2(drop, Agent, Object, AID)
-> breaks(AID, DPID, T) .
fluent_1(falling, Object, FID) & holdsAt(FID, T)
& parameter_1(height, Object, PID) & derivative(DPID, 2, PID)
-> value(DPID, T, "-9.8") .
parameter_1(height, Object, PID) & derivative(DPID, 2, PID)
& action_1(hitGround, Object, AID)
-> breaksTo(AID, DPID, T, 0) .
parameter_1(height, Object, PID) & derivative(DPID, 1, PID)
& action_1(hitGround, Object, AID)
```

[23]See falling object with events and falling object with antitrajectory examples by (Lee & Palla, 2009a).

```
-> breaksTo(AID, DPID, T, 0) .
parameter_1(height, Object, PID) & value(PID, T, V)
-> V >= 0 .
```

Simulation run for maximum time 3. The apple hits the ground at ∼0.78.

```
equals(next0,"0.7824607964359549").equals(next1,3).
value(pid1,0,3). value(pid3,0,0).value(pid2,0,0).
happens(aid1,0).happens(aid2,"0.7824607964359549").
holdsAt(fid1,next0).
value(pid1,"0.7824607964359549",0).value(pid2,"0.7824607964359549","-7.668").
value(pid3,"0.7824607964359549","-9.8000000000000007").
holdsAt(fid1,"0.7824607964359549").
value(pid1,3,0).value(pid2,3,0).value(pid3,3,0).
```

C.5 Hot Air Balloon

This example is discussed by Mueller (2006) and implemented by Lee and Palla (2012)[24]. When the heater is turned on it rises at a constant rate of 1 and when turned off it descends at a constant rate of −1. Lee and Palla (2012) describe continuous-change via *Trajectory* and *AntiTrajectory* relations whereas we use differential equations.

```
fluent_1(heaterOn, balloon, fid1) . fluent_1(moving, balloon, fid2) .
parameter_1(height, balloon, pid1) .
derivative(pid2, 1, pid1) .
action_2(turnOnHeater, nathan, balloon, aid1) .
action_2(turnOffHeater, nathan, balloon, aid2) .
initializedFalse(fid1) . initializedFalse(fid2) .
initialValue(pid1, 0) .initialValue(pid2, 0) .
happens(aid1, 0) . happens(aid2, 2) .

action_2(turnOnHeater, Agent, Balloon, AID)& fluent_1(heaterOn, Balloon, FID)
-> initiates(AID, FID, T) .
action_2(turnOffHeater, Agent, Balloon, AID)& fluent_1(heaterOn, Balloon, FID)
-> terminates(AID, FID, T) .
action_2(turnOnHeater, Agent, Balloon, AID)& fluent_1(moving, Balloon, FID)
```

[24]See hot air balloon example by (Lee & Palla, 2009a).

```
-> initiates(AID, FID, T) .
action_2(turnOffHeater, Agent, Balloon, AID)& fluent_1(moving, Balloon, FID)
-> initiates(AID, FID, T) .

action_2(turnOnHeater, Agent, Balloon, AID)& parameter_1(height, Balloon, PID)
& derivative(DPID, 1, PID)
-> breaks(AID, DPID, T) .
action_2(turnOffHeater, Agent, Balloon, AID)& parameter_1(height, Balloon, PID)
& derivative(DPID, 1, PID)
-> breaks(AID, DPID, T) .
fluent_1(heaterOn, Balloon, FID) & holdsAt(FID, T)
& parameter_1(height, Balloon, PID) & derivative(DPID, 1, PID)
-> value(DPID, T, 1) .
fluent_1(moving, Balloon, FID1) & holdsAt(FID1, T)
& fluent_1(heaterOn, Balloon, FID2) & -holdsAt(FID2, T)
& parameter_1(height, Balloon, PID) & derivative(DPID, 1, PID)
-> value(DPID, T, "-1") .
```

Simulation run for maximum time 4.

```
equals(next0,2).equals(next1,4).
happens(aid1,0).happens(aid2,2).
value(pid1,0,0).value(pid2,0,0).
holdsAt(fid2,next0).holdsAt(fid1,next0).
value(pid1,2,2).value(pid2,2,1).
holdsAt(fid1,2).holdsAt(fid2,2).holdsAt(fid2,next1).
value(pid1,4,0).value(pid2,4,"-1").holdsAt(fid2,4).
```

C.6 Multiple Blocks with Friction

Given below is the domain description for the multiple blocks problem discussed in Section 4.5 in F2LP syntax.

```
%two blocks b1,b2
%actions
action_3(applyhf,frc1,b1,30,aid1).action_3(applyhf,frc2,b2,"-20",aid2).
%action_3(applyhf,frc1,b1,r1,aid1).action_3(applyhf,frc2,b2,r2,aid2).
%action_3(applyhf,frc1,b1,r1,aid1).action_3(applyhf,frc2,b2,r2,aid2).
```

```
%fluents
fluent_3(ahf,frc1,b1,30,fid1).fluent_3(ahf,frc2,b2,"-20",fid2).
%fluent_3(ahf,frc1,b1,r1,fid1).fluent_3(ahf,frc2,b2,r2,fid2).
%fluent_3(ahf,frc1,b1,r1,fid1).fluent_3(ahf,frc2,b2,r2,fid2).
fluent_2(on,b2,b1,fid3).
fluent_2(sl,b2,b1,fid4).fluent_2(sr,b2,b1,fid5).
%fluent_2(conn,b2,b1,fid6).fluent_2(conn,b1,b2,fid7).
%quantities
%parameter_2(csf,b1,grnd,qid1).parameter_2(ckf,b1,grnd,qid2).
%parameter_2(csf,b2,b1,qid3).parameter_2(ckf,b2,b1,qid4).
parameter_2(fbba,grnd,b1,qid1).parameter_2(fbba,b1,b2,qid2).
parameter_2(fr,b1,grnd,qid3).
parameter_2(fr,b2,b1,qid4).parameter_2(fr,b1,b2,qid5).
parameter_1(mass,b1,qid6).parameter_1(mass,b2,qid7).
parameter_1(nahf,b1,qid8).parameter_1(nahf,b2,qid9).
parameter_1(navf,b1,qid10).parameter_1(navf,b2,qid11).
parameter_1(nhf,b1,qid12).parameter_1(nhf,b2,qid13).
parameter_1(pos,b1,qid14).parameter_1(pos,b2,qid15).
parameter_1(wt,b1,qid16).parameter_1(wt,b2,qid17).

%derivatives
derivative(qid18,1,qid14).derivative(qid19,1,qid15).
derivative(qid20,2,qid14).derivative(qid21,2,qid15).

%initialization
initializedFalse(fid1).initializedFalse(fid2).initializedTrue(fid3).
secondary(fid4).secondary(fid5).%secondary(fid6).secondary(fid7).
%initialValue(qid1,"0.9").initialValue(qid2,"0.9").
%initialValue(qid3,"0.7").initialValue(qid4,"0.7").
initialValue(qid1,147).initialValue(qid2,49).
initialValue(qid3,0).
initialValue(qid4,0).initialValue(qid5,0).
initialValue(qid6,10).initialValue(qid7,5).
initialValue(qid8,0).initialValue(qid9,0).
initialValue(qid10,0).initialValue(qid11,0).
initialValue(qid12,0).initialValue(qid13,0).
initialValue(qid14,0).initialValue(qid15,0).
initialValue(qid16,98).initialValue(qid17,49).%10*9.8, 5*9.8
```

```
initialValue(qid18,0).initialValue(qid19,0).
initialValue(qid20,0).initialValue(qid21,0).

%stratification of additive parameters
%{nahf,navf},{fbba},{nhf}

happens(aid1,1).
happens(aid2,2).
%indon
fluent_2(on,B2,B1,ONID)& holdsAt(ONID,T)-> indon(B2,B1,T).
fluent_2(on,B3,B2,ONID)& holdsAt(ONID,T)& indon(B2,B1,T)-> indon(B3,B1,T).

%Eff(D)
action_3(applyhf,F,B,R,AID)& fluent_3(ahf,F,B,R,FID)->initiates(AID,FID,T).
action_3(applyvf,F,B,R,AID)& fluent_3(avf,F,B,R,FID)->initiates(AID,FID,T).

%non-additive continuous-change and constraints
%moving right
fluent_2(on,B2,B1,ONID)& fluent_2(sr,B2,B1,SRID)& fluent_2(sl,B2,B1,SLID)
& parameter_1(pos,B1,POS1ID)&  parameter_1(pos,B2,POS2ID)
& parameter_2(fr,B2,B1,FRID)& parameter_2(fbba,B1,B2,FBBAID)
& derivative(VEL1ID,1,POS1ID)& derivative(VEL2ID,1,POS2ID)
& derivative(ACC1ID,2,POS1ID)& derivative(ACC2ID,2,POS2ID)
& value(VEL1ID,T,R11)& value(VEL2ID,T,R21)& value_2(FBBAID,T,RAB)
& holdsAt(ONID,T)& holdsAt(SRID, T)& R11==R21& X= "-0.7"*RAB
-> value_3(FRID,T,X).

fluent_2(on,B2,B1,ONID)& fluent_2(sr,B2,B1,SRID)& fluent_2(sl,B2,B1,SLID)
& parameter_1(pos,B1,POS1ID)&  parameter_1(pos,B2,POS2ID)
& parameter_2(fr,B2,B1,FRID)& parameter_2(fbba,B1,B2,FBBAID)
& derivative(VEL1ID,1,POS1ID)& derivative(VEL2ID,1,POS2ID)
& derivative(ACC1ID,2,POS1ID)& derivative(ACC2ID,2,POS2ID)
& value(VEL1ID,T,R11)& value(VEL2ID,T,R21)& value_5(ACC1ID,T,R12)
& value_5(ACC2ID,T,R22)& value_2(FBBAID,T,RAB)& holdsAt(ONID,T)
& holdsAt(SRID, T)& R11==R21& X= "-0.7"*RAB
-> R22> R12.

fluent_2(on,B2,B1,ONID)& fluent_2(sr,B2,B1,SRID)& fluent_2(sl,B2,B1,SLID)
```

```
& parameter_1(pos,B1,POS1ID)&  parameter_1(pos,B2,POS2ID)
& parameter_2(fr,B2,B1,FRID)& parameter_2(fbba,B1,B2,FBBAID)
& derivative(VEL1ID,1,POS1ID)& derivative(VEL2ID,1,POS2ID)
& derivative(ACC1ID,2,POS1ID)& derivative(ACC2ID,2,POS2ID)
& value(VEL1ID,T,R11)& value(VEL2ID,T,R21)& value_5(ACC1ID,T,R12)
& value_5(ACC2ID,T,R22)& value_2(FBBAID,T,RAB)& holdsAt(ONID,T)
& holdsAt(SRID, T)& R11==R21& X= "-0.7"*RAB
-> not holdsAt(SLID, T).

%moving left
fluent_2(on,B2,B1,ONID)& fluent_2(sr,B2,B1,SRID)& fluent_2(sl,B2,B1,SLID)
& parameter_1(pos,B1,POS1ID)&  parameter_1(pos,B2,POS2ID)
& parameter_2(fr,B2,B1,FRID)& parameter_2(fbba,B1,B2,FBBAID)
& derivative(VEL1ID,1,POS1ID)& derivative(VEL2ID,1,POS2ID)
& derivative(ACC1ID,2,POS1ID)& derivative(ACC2ID,2,POS2ID)
& value(VEL1ID,T,R11)& value(VEL2ID,T,R21)& value_2(FBBAID,T,RAB)
& holdsAt(ONID,T)& holdsAt(SLID, T)& R11==R21& X= "0.7"*RAB
-> value_3(FRID,T,X).

fluent_2(on,B2,B1,ONID)& fluent_2(sr,B2,B1,SRID)& fluent_2(sl,B2,B1,SLID)
& parameter_1(pos,B1,POS1ID)&  parameter_1(pos,B2,POS2ID)
& parameter_2(fr,B2,B1,FRID)& parameter_2(fbba,B1,B2,FBBAID)
& derivative(VEL1ID,1,POS1ID)& derivative(VEL2ID,1,POS2ID)
& derivative(ACC1ID,2,POS1ID)& derivative(ACC2ID,2,POS2ID)
& value(VEL1ID,T,R11)& value(VEL2ID,T,R21)& value_5(ACC1ID,T,R12)
& value_5(ACC2ID,T,R22)& value_2(FBBAID,T,RAB)& holdsAt(ONID,T)
& holdsAt(SLID, T)& R11==R21& X= "0.7"*RAB
-> R22< R12.

fluent_2(on,B2,B1,ONID)& fluent_2(sr,B2,B1,SRID)& fluent_2(sl,B2,B1,SLID)
& parameter_1(pos,B1,POS1ID)&  parameter_1(pos,B2,POS2ID)
& parameter_2(fr,B2,B1,FRID)& parameter_2(fbba,B1,B2,FBBAID)
& derivative(VEL1ID,1,POS1ID)& derivative(VEL2ID,1,POS2ID)
& derivative(ACC1ID,2,POS1ID)& derivative(ACC2ID,2,POS2ID)
& value(VEL1ID,T,R11)& value(VEL2ID,T,R21)& value_5(ACC1ID,T,R12)
& value_5(ACC2ID,T,R22)& value_2(FBBAID,T,RAB)& holdsAt(ONID,T)
& holdsAt(SLID, T)& R11==R21& X= "0.7"*RAB
-> not holdsAt(SRID, T).
```

```
%moving together
fluent_2(on,B2,B1,ONID)& fluent_2(sr,B2,B1,SRID)& fluent_2(sl,B2,B1,SLID)
& parameter_1(pos,B1,POS1ID)&  parameter_1(pos,B2,POS2ID)
& parameter_2(fr,B2,B1,FRID)& parameter_2(fbba,B1,B2,FBBAID)
& derivative(VEL1ID,1,POS1ID)& derivative(VEL2ID,1,POS2ID)
& derivative(ACC1ID,2,POS1ID)& derivative(ACC2ID,2,POS2ID)
& value(VEL1ID,T,R11)& value(VEL2ID,T,R21)& value_5(ACC1ID,T,R12)
& value_5(ACC2ID,T,R22)& value_2(FBBAID,T,RAB)& value_3(FRID,T,RFR)
& holdsAt(ONID,T)& not holdsAt(SLID,T)& not holdsAt(SRID,T)
& R11==R21& RABSQ= RAB*RAB& RFRSQ= RFR*RFR& X= "0.81"*RABSQ
-> R22== R12.

fluent_2(on,B2,B1,ONID)& fluent_2(sr,B2,B1,SRID)& fluent_2(sl,B2,B1,SLID)
& parameter_1(pos,B1,POS1ID)&  parameter_1(pos,B2,POS2ID)
& parameter_2(fr,B2,B1,FRID)& parameter_2(fbba,B1,B2,FBBAID)
& derivative(VEL1ID,1,POS1ID)& derivative(VEL2ID,1,POS2ID)
& derivative(ACC1ID,2,POS1ID)& derivative(ACC2ID,2,POS2ID)
& value(QID11,T,R11)& value(QID21,T,R21)& value(QID12,T,R12)
& value(QID22,T,R22)& value_2(FBBAID,T,RAB)& value_3(FRID,T,RFR)
& holdsAt(ONID,T)& not holdsAt(SLID,T)& not holdsAt(SRID,T)
& R11==R21& RABSQ= RAB*RAB& RFRSQ= RFR*RFR& X= "0.81"*RABSQ
-> RFRSQ<= X.

fluent_2(on,B2,B1,ONID)& holdsAt(ONID,T)& parameter_2(fr,B2,B1,FRID)
& parameter_2(fr,B1,B2,OPPID)& value_3(FRID,T,RFR)& X= "-1"*RFR
-> value(OPPID,T,X).

parameter_1(pos,B,POSID)& derivative(ACCID,2,POSID)
& parameter_1(mass,B,MASSID)& parameter_1(nhf,B,NHFID)
& partValue_4(NHFID,T,PX_ARB,R_ARB)& value_4(NHFID,T,RNHF)
& value(MASSID,T,RMASS)& X=RNHF/RMASS
-> value_5(ACCID,T,X).

%additive effects
fluent_3(ahf,FRC,B,V,FID)& holdsAt(FID,T)& parameter_1(nahf,B,QID)
-> partValue_1(QID,T,FRC,V).
fluent_3(avf,FRC,B,V,FID)& holdsAt(FID,T)& parameter_1(navf,B,QID)
```

```
-> partValue_1(QID,T,FRC,V).
fluent_2(on,B2,B1,FID)& holdsAt(FID,T)& parameter_2(fbba,B1,B2,QID)
& parameter_1(wt,B2,WID)& value(WID,T,RW)
-> partValue_2(QID,T,WID,RW).
fluent_2(on,B2,B1,FID)& holdsAt(FID,T)& parameter_2(fbba,B1,B2,QID)
& parameter_1(navf,B2,NAVFID)& value_1(NAVFID,T,RNAVF)
-> partValue_2(QID,T,NAVFID,RNAVF).
indon(B3,B2,T)& fluent_2(on,B2,B1,FID)& holdsAt(FID,T)
& parameter_2(fbba,B1,B2,QID)& parameter_1(wt,B3,WID)& value(WID,T,RW)
-> partValue_2(QID,T,WID,RW).
indon(B3,B2,T)& fluent_2(on,B2,B1,FID)& holdsAt(FID,T)
& parameter_2(fbba,B1,B2,QID)& parameter_1(navf,B3,WID)
& value_1(NAVFID,T,RNAVF)
-> partValue_2(QID,T,NAVFID,RNAVF).
fluent_3(ahf,FRC,B,V,FID)& holdsAt(FID,T)& parameter_1(nhf,B,QID)
& parameter_1(nahf,B,NAHFID)& value_1(NAHFID,T,RNAHF)
-> partValue_4(QID,T,B,RNAHF).
fluent_3(ahf,FRC_ARB,B_ARB,V_ARB,FID_ARB)& holdsAt(FID_ARB,T)
& fluent_2(on,B2,B1,FID)& holdsAt(FID,T)& parameter_1(nhf,B1,QID)
& parameter_2(fr,B1,B2,FRID)& value_3(FRID,T,RFR)
-> partValue_4(QID,T,B2,RFR).
fluent_3(ahf,FRC_ARB,B_ARB,V_ARB,FID_ARB)& holdsAt(FID_ARB,T)
& fluent_2(on,B2,B1,FID)& holdsAt(FID,T)& parameter_1(nhf,B2,QID)
& parameter_2(fr,B2,B1,FRID)& value_3(FRID,T,RFR)
-> partValue_4(QID,T,B1,RFR).

%discontinuities caused by actions
%initiates(a,f,t) -> initorterm(a,f,t).
%terminates(a,f,t) -> initorterm(a,f,t).
action_3(applyhf,FRC,B1,R1,AID)& parameter_1(nahf,B2,QID)
-> breaks(AID,QID,T).
action_3(applyvf,FRC,B1,R1,AID)& parameter_1(navf,B2,QID)
-> breaks(AID,QID,T).
action_3(applyvf,FRC,B1,R1,AID)& parameter_2(fbba,B2,B3,QID)
-> breaks(AID,QID,T).
action_3(applyhf,FRC,B1,R1,AID)& parameter_2(fr,B2,B3,QID)
-> breaks(AID,QID,T).
action_3(applyvf,FRC,B1,R1,AID)& parameter_2(fr,B2,B3,QID)
```

```
-> breaks(AID,QID,T).
action_3(applyhf,FRC,B1,R1,AID)& parameter_1(nhf,B2,QID)
-> breaks(AID,QID,T).
action_3(applyvf,FRC,B1,R1,AID)& parameter_1(nhf,B2,QID)
-> breaks(AID,QID,T).
action_3(applyhf,FRC,B1,R1,AID)& parameter_1(pos,B2,QID)
& derivative(QID2,2,QID)
-> breaks(AID,QID2,T).
action_3(applyvf,FRC,B1,R1,AID)& parameter_1(pos,B2,QID)
& derivative(QID2,2,QID)
-> breaks(AID,QID2,T).
```

This is one of the examples where DLVHEX failed at logical reasoning returning the following error:

```
Exception: unsafe variables in:
program:88:1: aux_r_10000000_12("-0x1.6666666666666p-1",RAB,X)|aux_n_10000000_12
("-0x1.6666666666666p-1",RAB,X):-value_2(FBBAID,T,RAB),aux_i_6000000_19(RAB).
        program:88:47: X
        program:88:98: X
```

aux_r_* and aux_n_* are auxiliary predicates, corresponding with external atoms $\&times$ external atom in this example, generated by DLVHEX at the time of reasoning. So DLVHEX unexpectedly detects variable X to be unsafe, even though it is not unsafe in the original program listed above.

C.7 Drinking Bird

The following introduction to the drinking problem (Figure C.1) is borrowed from (Güémez et al., 2003). Figure 1 shows a schematic and a photo of this intriguing thermodynamic device. The bird consists of two spherical glass bulbs connected by a glass tube that enters well inside the lower bulb. The bottom bulb (the birds body, hereafter simply referred to as the body) is almost filled with a highly volatile liquid, normally methylene chloride (CH_2Cl_2), whose normal boiling point is close to room temperature. There is no air inside the bird, but only this internal liquid in thermal equilibrium with its vapor. The top bulb (the birds head, hereafter simply

Figure C.1: Drinking bird about to dip its beak in the water. Courtesy (Güémez et al., 2003).

called the head) is covered with a porous tissue. It has a small plastic hat and a long beak which is covered with the same tissue as the head. The bird can oscillate around a horizontal metallic bar attached to the tube at the middle. When the bird leans completely forward, it "drinks" water from a glass, although other external liquids may be used as well. This motion is called a dip.

The drinking bird undergoes a cycle. At the beginning of the cycle, the bird is upright, with all the internal liquid in the lower sphere. The water on the head is in contact with its vapor at a given (room) temperature. If the vapor pressure is smaller than its saturation (or equilibrium) value, evaporation occurs spontaneously. The evaporation cools the head outside, so that the CH_2Cl_2 vapor inside also has to cool. The vapor in the head condenses in very small drops, remaining in equilibrium with the internal liquid as the temperature decreases. The CH_2Cl_2 vapor pressure inside the head becomes smaller than that in the body according to the ClausiusClapeyron equation, and this pressure gradient forces the internal liquid to rise up in the tube. As the liquid rises, the center of mass of the system also rises, and the momentum produced by the weight eventually forces the bird to tip forward and to dip its beak in the glass, keeping its head wet. When the bird is almost horizontal, the lower end of the tube emerges above the internal liquid surface and some vapor passes from the body to the head. While drinking, the bird remains horizontal for a short time. Then, part of the liquid drains back into the body and the bird returns to its upright position. As water evaporation continues from the head, the internal liquid comes up again, starting a new cycle.

Here we give the axiomatic description of the drinking bird problem in the

Event Calculus. We use the following symbols:

Constants: $Head$ (head of the bird), $Body$ (body of the bird), $Tube$ (tube starting inside the body and connected to the head), $Beak$ (beak of the bird), $InternalLiquid$ or just $Liquid$ (liquid inside the bird, e.g. CH_2Cl_2), $Water$ (water in the glass), $Length$ (length of the tube), $OneTimeDrinkingAmount$ (amount water sucked from the glass in one dip)

Fluents: $Moist$ (beak of the bird is moist), $Heavier(Body, Head)$ (head is heavier than the body), $MoreVaporPressure(Head, Body)$ (head has a higher vapor pressure than the body), $MovingForward$ (tube is in an angular motion with the head moving towards the glass of water), $MovingBackward$ (tube is in an angular motion with the head moving away from the glass of water), $RisingLiquid$ (liquid is rising in the tube).

Parameters: $VaporPressure(Body)$ (vapor pressure of the liquid vapor in the body. Likewise for the head.), $Amount(Water)$ (amount of water in the glass), $Level$ (the level, from the bottom of the tube that liquid has risen in the tube), $RateOfRisingLiquid$ or $RateRL$ (rate at which liquid is rising), $Angle$ (angle that the tube makes with respect to the horizontal), $RateOfForwardAngular-Motion$ or $RateFAM$ (the rate of angular motion of the tube as the head moves towards the the glass), $RateOfBackwardAngularMotion$ or $RateBAM$ (the rate of angular motion of the tube as the head moves away from the the glass).

Actions: Dip (act marking beak of the bird dipping in water), $FlowLiquid(Head, Body)$ (act triggering the flow of water from the head to the body), $FlowVapor(Body, Head)$ (act triggering flow of vapor from body to head), $MoveForward$ (act triggering forward angular motion of the head), $MoveBackward$ (act triggering backward motion of the head), $StopMovingForward$ (act triggering stopping any further forward movement), $StopMovingBackward$ (act triggering stopping any further backward movement) $Evaporate$ (act triggers evaporation of water from the beak), $PushLiquid$ (act triggers pushed towards the head).

Initially, the body is upright, neither there is any liquid in the tube nor is the liquid starting to flow into the tube, the tube is moving neither forward nor backward, and the beak is not moist but there is a positive vapor pressure difference between the head and the body.

$$Value(Angle, 0) = \pi/2 \wedge Value(Level, 0) = 0 \wedge \neg HoldsAt(RisingLiquid, 0)$$
$$\wedge \neg HoldsAt(MovingForward, 0) \wedge \neg HoldsAt(MovingBackward, 0) \quad \text{(DB1)}$$
$$\wedge \neg HoldsAt(Moist, 0) \wedge HoldsAt(MoreVaporPressure(Head, Body), 0)$$

If there is a positive pressure difference between the head and the body and the tube is upright, then the liquid is triggered to rise (Axiom DB2)). Act $PushLiquid$ initiates the $RisingLiquid$ condition (Axiom (DB3)) and breaks continuity of rate of change of $Level$ (Axiom (DB4)), as liquid begins to rise at the rate $RateRL$ (Axiom (DB5)).

$$[HoldsAt(MoreVaporPressure(Head, Body), t) \wedge Value(Angle, t) = \pi/2] \quad \text{(DB2)}$$
$$\longrightarrow Happens(PushLiquid, t)$$
$$Initiates(PushLiquid, RisingLiquid, t) \quad \text{(DB3)}$$
$$Breaks(PushLiquid, \delta(Level), t) \quad \text{(DB4)}$$
$$[HoldsAt(RisingLiquid, t) \wedge Value(Level, t) < Length] \quad \text{(DB5)}$$
$$\longrightarrow Value(\delta(Level), t) = RateRL$$

When the liquid is not rising, the rate of change in level is zero.

$$\neg[HoldsAt(RisingLiquid, t) \wedge Value(Level, t) < Length] \quad \text{(DB6)}$$
$$\longrightarrow Value(\delta(Level), t) = 0$$

When the liquid in the tube rises beyond the hinge, which is assumed to be at $Length/2$, a torque is triggered (Axiom (DB7)), that sets the head to move forward

(Axiom (DB8)), at the rate of $RateFAM$ (Axiom (DB9)).

$$[Value(Level, t) > Lenght/2 \wedge HoldsAt(RisingLiquid, t)] \longrightarrow Happens(MoveForward, t) \quad \text{(DB7)}$$

$$Initiates(MoveForward, MovingForward, t) \quad \text{(DB8)}$$

$$HoldsAt(MovingForward, t) \longrightarrow Value(\delta(Angle), t) = -1 \times Value(RateFAM, t) \quad \text{(DB9)}$$

We simplify the problem and assume that as the head begins to $MoveForward$, the body is no longer heavier than the head (which it becomes after some liquid flows from the head to the body when the tube is horizontal), the liquid doesn't rise any further, and the vapor pressure in the head is no more greater in the head than in the body (which becomes the case after evaporation of water from the head, in the upright position) (Axiom (DB10)). Also, the liquid suddenly stops to rise, that is $MoveForward$ also breaks the continuity rate of change of $Level$ (Axiom (DB11)). These simplifications could be adjusted by quantitative descriptions of change in vapor pressure, etc.

$$[f = Heavier(Body, Head) \vee f = RisingLiquid \vee f = MoreVaporPressure] \longrightarrow Terminates(MoveForward, f, t) \quad \text{(DB10)}$$

$$Breaks(MoveForward, \delta(Level), t) \quad \text{(DB11)}$$

The act of moving forward, and likewise stoppage of forward movement, backward movement and its stoppage, break the continuity of rate of change of Level.

$$[a = MoveForward \vee a = StopMovingForward \vee a = MoveBackward \vee a = StopMovingBackward] \longrightarrow Breaks(a, \delta(Angle), t) \quad \text{(DB12)}$$

The forward motion is stopped by the act $StopMovingForward$.

$$Terminates(StopMovingForward, MovingForward, t) \tag{DB13}$$

Axioms (DB14), (DB15), (DB16) are counterparts of Axioms (DB8), (DB9), (DB13), when the head is moving backwards instead of forward.

$$Initiates(MoveBackward, MovingBackward, t) \tag{DB14}$$

$$HoldsAt(MovingBackward, t) \longrightarrow Value(\delta(Angle), t) = Value(RateBAM, t) \tag{DB15}$$

$$Terminates(StopMovingBackward, MovingBackward, t) \tag{DB16}$$

When the tube becomes horizontal then it dips (Axiom (DB17)) and moists its beak (Axiom (DB18)), while drinking out some water from the glass (Axiom (DB19)). The water will be able to moist its beak only if there is some water in the glass (Axiom (DB18)). When bird becomes horizontal it also stops moving forward (Axiom (DB17)).

$$Value(Angle, t) = 0 \longrightarrow [Happens(Dip, t) \wedge Happens(StopMovingForward, t)] \tag{DB17}$$

$$Value(Amount(Water), t) > 0 \longrightarrow Initiates(Dip, Moist, t) \tag{DB18}$$

$$Value(Amount(Water), t) > 0 \tag{DB19}$$
$$\longrightarrow BreaksTo(Dip, Amount(Water), -1 \times OneTimeDrinkingAmount)$$

When the tube becomes horizontal, there is also flow of vapor from the body to the head and flow of liquid from the head to the body (Axiom (DB20)), and the latter makes the head heavier than the body again (Axiom (DB22)). We do not account for any direct effect of flow of liquid vapor on change in vapor pressures, and instead we simplify the system such that its direct effect is to flow any liquid in the tube out of the tube such that the level of liquid in the tube becomes zero

(Axiom (DB21)).

$$Value(Angle, t) = 0 \longrightarrow [Happens(FlowVapor(Body, Head), t) \wedge Happens(FlowLiquid(Head, Body), t)] \quad \text{(DB20)}$$

$$BreaksTo(FlowVapor(Body, Head), Level, t, 0) \quad \text{(DB21)}$$

$$Initiates(FlowLiquid(Head, Body), Heavier(Body, Head), t) \quad \text{(DB22)}$$

When the tube is horizontal and the body becomes heavier than the head then a torque is triggered that moves the head backwards (Axiom (DB23)), which stops once the tube becomes vertical (Axiom (DB24)).

$$[Value(Angle, t) = 0 \wedge HoldsAt(Heavier(Body, Head), t) \longrightarrow Happens(MoveBackward, t)] \quad \text{(DB23)}$$

$$Value(Angle, t) = \pi/2 \longrightarrow Happens(StopMovingBackward, t) \quad \text{(DB24)}$$

When the tube is upright and its beak is moist, evaporation happens (Axiom (DB25)) and the beak is not moist anymore (Axiom (DB26)). Due to the evaporation, the vapor pressure in the head becomes more than that in the body (Axiom (DB27)).

$$[Value(Angle, t) = \pi/2 \wedge HoldsAt(Moist, t)] \longrightarrow Happens(Evaporate, t) \quad \text{(DB25)}$$

$$Terminates(Evaporate, Moist, t) \quad \text{(DB26)}$$

$$Initiates(Evaporate, MoreVaporPressure(Head, Body), t) \quad \text{(DB27)}$$

The rate of angular motion is zero when the tube is moving neither forward nor backward.

$$[\neg HoldsAt(MovingForward, t) \wedge \neg HoldsAt(MovingBackward, t)] \longrightarrow Value(\delta(Angle), t) = 0 \quad \text{(DB28)}$$