First-Order Logic as a Constraint Programming Language

K.R. Apt[1,2] and C.F.M. Vermeulen[1]

[1] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
[2] University of Amsterdam, The Netherlands

Abstract. We provide a denotational semantics for first-order logic that captures the two-level view of the computation process typical for constraint programming. At one level we have the usual program execution. At the other level an automatic maintenance of the constraint store takes place.
We prove that the resulting semantics is sound with respect to the truth definition. By instantiating it by specific forms of constraint management policies we obtain several sound evaluation policies of first-order formulas. This semantics can also be used a basis for sound implementation of constraint maintenance in presence of block declarations and conditionals.

1 Introduction

By the celebrated result of Tarski first-order logic is undecidable. In particular, the question of determining for an interpretation whether a first-order formula is satisfiable and finding a satisfying substitution if it is, is undecidable. Still, for many formulas this question can be answered in a straightforward way. Take for instance the following simple formula interpreted over the standard interpretation for arithmetics:

$$y < z \wedge y = 1 \wedge z = 2 \tag{1}$$

It is easy to see that it is satisfied by the substitution $\{y/\mathbf{1}, z/\mathbf{2}\}$. Similarly, it is easy to see that the formula

$$\neg(x = 1) \wedge x = 0 \tag{2}$$

is satisfied by the substitution $\{x/\mathbf{0}\}$.
The question is whether we can capture this concept of "straightforwardness" in a natural way. Our first attempt to answer this question was given in Apt and Bezem [3] by providing a natural operational semantics for first-order logic which is independent of the underlying interpretation for it. It captures the computation process as a search for a satisfying substitution for the formula in question. Because the problem of finding such a substitution is in general undecidable, we introduced in it the possibility of a partial answers in the form of a special **error** state indicating a run-time error. In Apt [2] we slightly extend this approach by

M. Baaz and A. Voronkov (Eds.): LPAR 2002, LNAI 2514, pp. 19–35, 2002.

explaining how more general equalities can be handled and formulate it in the form of a denotational semantics for first-order logic. Unfortunately, both semantics are too weak to deal properly with formulas (1) and (2): for both of them the **error** state is generated.

In this paper we try to overcome these limitations by providing a computational interpretation of first-order logic in the spirit of constraint programming. According to this view the computation process takes place on two levels. At one level we have the usual program execution. At the other level, in the "background" inaccessible to the user, an automatic maintenance of the constraint store takes place. The problem we tackle is undecidable, so we introduce the possibility of partial answers. They are modeled now by a non-empty constraint store or the **error** state.

The automatic maintenance of the constraint store is modeled by an parametric *infer* operation that acts on states. The idea of an abstract *infer* operation is due to Jaffar and Maher [9]. Here we consider it in presence of arbitrary first-order formulas. Because of this generality we can obtain various sound realizations of the constraint store management by appropriately instantiating *infer*. The correctness of this approach is formalized in the form of an appropriate soundness result. To establish it we need to assume some properties of the *infer* operation. They are formulated as five "healthiness" conditions.

To illustrate the benefits of this view of first-order logic and to show the scope of the soundness result, we discuss several ways of instantiating the *infer* parameter to specific constraint management policies. Examples include admission of a constraint store consisting of arbitrary first-order formulas, restriction to a constraint store consisting only of atomic constraints, and restriction to a constraint store consisting only of arbitrary first-order positive formulas. We can also discuss in this framework in a uniform way unification, an algorithm for solving equations and disequations over the Herbrand algebra, and Gaussian elimination in presence of arithmetic constraints.

On the more practical side, these considerations lead to specific implementation proposals of the constraint store in presence of block declarations and conditionals, here modeled, respectively, by means of existential quantification and of negation, conjunction and disjunction.

To clarify these issues we return to formula (1). If we do not admit constraint storage, as in the semantics of [3] and [2], its evaluation yields the **error** state, since we cannot evaluate $y < z$ without knowing the values for y and z. But if we do allow atomic constraints in the store, we can postpone the evaluation of $y < z$ and the evaluation yields the substitution $\{y/\mathbf{1}, z/\mathbf{2}\}$.

Next, let us reconsider formula (2). If only atomic formulas are allowed as constraints, the evaluation of this formula yields the **error** state, since we can neither evaluate $\neg(x = 1)$ nor add this formula to the constraint store. If, however, negated formulas are allowed in the constraint store the substitution $\{x/\mathbf{0}\}$ is an answer. The soundness theorem states that each computed substitution satisfies the evaluated formula.

The question of providing an appropriate semantics for first-order logic in the spirit of constraint programming could be approached by taking for a formula $\phi(\bar{x})$ a clause $p(\bar{x}) \leftarrow \phi(\bar{x})$, where p is a new relational symbol and by applying to it a transformation in the style of Lloyd and Topor [11]. The outcome would be a constraint logic program that uses negation. After clarifying how to deal properly with negation this could yield a rather indirect answer to the question we study. In contrast, our approach, expressed in the form of a denotational semantics, is much more direct and conceptually transparent: the meaning of each formula is expressed directly in terms of the meaning of its constituents and it is parametrized in a simple way by the *infer* operation.
The rest of the paper is organized as follows. In Section 2 we introduce the *infer* operation and discuss in detail the requirements we impose on it. The main difficulty has to do with the appropriate treatment of existential variables. In Section 3 we define our denotational semantics. Next, in Section 4 we show that the proposed semantics subsumes the denotational semantics provided in [2]. Then in Section 5 we discuss various increasingly powerful forms of constraint store management, each modeled by means of a particular *infer* operation. Finally, in Section 6 we discuss related work.

2 Towards the Denotational Semantics

Below we work our way towards our proposal for the denotational semantics in several steps, first introducing the basic semantic ingredients, then discussing the crucial conditions on the *infer* parameter and finally presenting the denotational semantics for first order logic with *infer* parameter. In the next section we will then state the soundness result for the semantics. The proof details are referred to the appendix. We discuss several ways of instantiating the *infer* parameter to show the scope of the soundness result. In the final section we review the goals and results and look ahead to further developments.

Preliminaries. Let's assume that an algebra $\mathcal{J}$ is given over which we want to perform computations. The basic ingredient of the semantic universe will be the set of states, STATES. States come in two kinds. First we have an ERROR state, which remains unanalyzed. All other states consist of two components: one component is a constraint store $\mathcal{C}$, the other a substitution θ. Such a state is then written $\langle \mathcal{C}; \theta \rangle$. As always, a substitution θ is a mapping from variables to terms. It assigns a term $x\theta$ to each variable x, but there are only finitely many variables for which $x \neq x\theta$. These variables form $dom(\theta)$, the domain of θ. The application of a substitution θ to a term t, written $t\theta$, is defined as usual. We denote the empty substitution by ϵ.
A constraint store $\mathcal{C}$, is simply a finite set of formulas of first order logic. In many applications there are extra requirements on the syntactic form of a constraint store, but for now we keep things as general as possible. $\perp$ is a special formula which is always false.

Throughout the paper we try to limit the number of brackets and braces as much as possible. In particular, for a finite set $\{A_1, \ldots, A_n\}$ we will often write $A_1, \ldots, A_n$. Also, we write $infer\langle \mathcal{C}; \theta \rangle$ instead of $infer(\langle \mathcal{C}; \theta \rangle)$, etc.

The treatment of local variables: dropping things. An important ingredient of the set up is the $DROP_u$ mapping on states. It is the way we deal with local variables. This works in two steps: first we define the substitution $DROP_u(\theta)$ for each variable u and substitution θ, as in [2]:

$$\begin{aligned} uDROP_u(\theta) &= u \\ xDROP_u(\theta) &= x\theta \text{ for all other variables } x \end{aligned}$$

So, $DROP_u$ makes the current value of u disappear, thus capturing the idea of a local variable to the substitutions. But we also have another component in states: the constraint store. Dropping u from such a set of formulas compares to existential quantification over u. There is one little extra point to take care of, however: in a state $\langle \mathcal{C}; \eta \rangle$ the information that η provides about the value of u is implicitly available to $\mathcal{C}$. Therefore, we perform the quantification $\exists u$ only after adding the information about the value of u explicitly to $\mathcal{C}$. Also the values $y\eta$ in which u appears have to be kept in mind. We take the conjunction of the equations $y = y\eta$ for all such variables y and write it as $\mathbf{y} = \mathbf{y}\eta$. This leads us to the following formula that takes care of the local variables in $\mathcal{C}$.

$$\exists u \, (u = u\eta \wedge \mathbf{y} = \mathbf{y}\eta \wedge \bigwedge \mathcal{C})$$

Note that this formula depends both on u and η. So, we cannot define a drop_u-mapping on constraint stores alone: we have to know η as well.
This formula expresses the information we are after in a uniform way, but in 'borderline cases' the syntactic format is awkward. For example, if $\mathcal{C} = \emptyset$, we get a trivial existential quantification over the first two conjuncts. This existential quantifier is semantically harmless, but specific constraint propagation formalisms simply do not work on existentially quantified formulas. Therefore we rather adopt a format in which the quantifier only appears if it is really necessary.
This is done in two steps: first, the quantification over u only matters for the formulas in $\mathcal{C}$ in which u actually occurs. We make this explicit in the definition by distinguishing $\mathcal{C}(u)$, the subset of $\mathcal{C}$ that contains exactly the formulas with the free variable u. In the formula we use for the drop_u-mapping we can then always take $\mathcal{C} - \mathcal{C}(u)$ outside the scope of the quantifier. This gives:

$$\mathcal{C} - \mathcal{C}(u) \ , \ \exists u \, (u = u\eta \wedge \mathbf{y} = \mathbf{y}\eta \wedge \bigwedge \mathcal{C}(u))$$

Finally, in the special case $\mathcal{C}(u) = \emptyset$, we leave out the existentially quantified formula altogether.
For the ERROR-state we simply set: $\mathrm{DROP}_u\mathrm{ERROR} = \mathrm{ERROR}$. To summarize, the mapping DROP_u is defined on states by the following cases:

$$
\begin{array}{lll}
\text{DROP}_u\langle \mathcal{C};\eta\rangle & = \langle \mathcal{C}; DROP_u(\eta)\rangle & \text{if } \mathcal{C}(u)=\emptyset \\
\text{DROP}_u\langle \mathcal{C};\eta\rangle & = \langle\ \exists u\ (u=u\eta\ \wedge\ \mathbf{y}=\mathbf{y}\eta\ \wedge\ \bigwedge \mathcal{C}(u)), & \\
 & \qquad\qquad \mathcal{C}-\mathcal{C}(u);\quad DROP_u(\eta)\rangle & \text{if } \mathcal{C}(u)\neq\emptyset \\
\text{DROP}_u\text{ERROR} & = \text{ERROR} &
\end{array}
$$

Conditions on infer. Another important ingredient of the framework is the *infer* mapping. *infer* maps a state to a set of states. The *infer* mapping is the basic notion of computation in the semantics: we do not specify what happens 'within' the *infer* mapping. This makes the set up extremely general: the *infer* steps can consist of calls to a constraint solver, like a unification algorithm or an algorithm for solving linear equations over reals, calls to a constraint propagation algorithm, or other atomic computation steps. Several instances of the *infer* mapping will be discussed in more detail later on.
We can almost get away with complete generality regarding *infer*. To make sure that the formalism respects first order logic, we have to make a few modest requirements. Let us write $\langle \mathcal{C};\theta\rangle \models_{\mathcal{J}} \phi$ for $\mathcal{C}\theta \models_{\mathcal{J}} \phi\theta$. In particular $\langle \emptyset;\theta\rangle \models_{\mathcal{J}} \phi$ iff $\models_{\mathcal{J}} \phi\theta$. Then the restrictions that we need in the soundness proof below, read as follows:

(1) Equivalence: if $\langle \mathcal{C}';\theta'\rangle \in \mathit{infer}\langle \mathcal{C};\theta\rangle$, then $\langle \mathcal{C};\theta\rangle \models_{\mathcal{J}} \phi$ iff $\langle \mathcal{C}';\theta'\rangle \models_{\mathcal{J}} \phi$
(2) Renaming: if $\langle \mathcal{C}';\theta'\rangle \in \mathit{infer}\langle \mathcal{C};\theta\rangle$, then also $\langle \mathcal{C}'_v;\theta'_v\rangle \in \mathit{infer}\langle \mathcal{C};\theta\rangle$, where $\langle \mathcal{C}'_v;\theta'_v\rangle$ is obtained from $\langle \mathcal{C}';\theta'\rangle$ by replacing all occurrences of u by v for a variable u that is fresh w.r.t. $\langle \mathcal{C};\theta\rangle$ and a variable v that is fresh w.r.t. both $\langle \mathcal{C};\theta\rangle$ and $\langle \mathcal{C}';\theta'\rangle$
(3) Inconsistency: if $\mathit{infer}\langle \mathcal{C};\theta\rangle = \emptyset$, then $\langle \mathcal{C};\theta\rangle \models_{\mathcal{J}} \bot$
(4) Error: $\mathit{infer}\ \text{ERROR} = \{\text{ERROR}\}$
(5) Identity: $\mathit{infer}\langle \emptyset;\theta\rangle = \{\langle \emptyset;\theta\rangle\}$

So, the *infer* mapping should respect logical equivalence, i.e., the state $\langle \mathcal{C}';\theta'\rangle$ that we reach starting from $\langle \mathcal{C};\theta\rangle$, should still make the same formulas true. Furthermore, the *infer* mapping should not be sensitive to the choice of fresh variables: if *infer* works for u, it should also work for an alternative fresh variable v. Finally, *infer* should respect falsity and the ERROR state.[1] When we talk about the consistency of states, we are dealing with a three way distinction. We say that a state σ is: $\mathcal{J}$-consistent, if $\sigma \neq \text{ERROR}$ and $\sigma \not\models_{\mathcal{J}} \bot$; $\mathcal{J}$-inconsistent, if $\sigma \neq \text{ERROR}$ and $\sigma \models_{\mathcal{J}} \bot$; error, if $\sigma = \text{ERROR}$. For a set of states $\Sigma \subseteq \text{STATES}$ we then distinguish: $\mathit{cons}_{\mathcal{J}}(\Sigma) = \{\sigma \in \Sigma :\ \sigma \text{ is } \mathcal{J}\text{-consistent}\}$ and $\mathit{cons}^{+}_{\mathcal{J}}(\Sigma) = \{\sigma \in \Sigma :\ \sigma \text{ is not } \mathcal{J}\text{-inconsistent}\}$. Usually it is clear to which $\mathcal{J}$ we refer and we omit $\mathcal{J}$ from the notation.

[1] The Identity requirement is not necessary for the proof of the soundness theorem, but it seems too natural to leave it out. Renaming is used only in the proof of the Preservation/Persistence Lemma in the case of the existential formula.

3 Denotational Semantics

We now define a denotational semantics for first order logic in which the *infer* mapping is a parameter. The parameter can be set to give the semantics from Apt [2], for example, but many other settings are available, as we will see below. This way we obtain general results, that apply uniformly to various forms of constraint store management.
We define the mapping $[\![\phi]\!]$: STATES $\rightarrow$ STATES, using postfix notation.[2]

$$
\begin{array}{lcl}
\langle \mathcal{C};\theta\rangle[\![A]\!] & = & \mathit{infer}\langle \mathcal{C}, A;\theta\rangle \qquad \text{for an atomic formula } A \\
\langle \mathcal{C};\theta\rangle[\![\phi_1 \vee \phi_2]\!] & = & \langle \mathcal{C};\theta\rangle[\![\phi_1]\!] \cup \langle \mathcal{C};\theta\rangle[\![\phi_2]\!] \\
\langle \mathcal{C};\theta\rangle[\![\phi_1 \wedge \phi_2]\!] & = & (\langle \mathcal{C};\theta\rangle[\![\phi_1]\!])[\![\phi_2]\!] \\
\langle \mathcal{C};\theta\rangle[\![\neg\phi]\!] & = & \begin{cases} \mathit{infer}\langle \mathcal{C};\theta\rangle & \text{if } \mathit{cons}^+(\langle \mathcal{C};\theta\rangle[\![\phi]\!]) = \emptyset \\ \emptyset & \text{if } \langle \mathcal{C}';\theta'\rangle \in \mathit{cons}(\langle \mathcal{C};\theta\rangle[\![\phi]\!]) \text{ for} \\ & \text{some } \langle \mathcal{C}';\theta'\rangle \text{ equivalent to } \langle \mathcal{C};\theta\rangle \\ \mathit{infer}\langle \mathcal{C}, \neg\phi;\theta\rangle & \text{otherwise} \end{cases} \\
\langle \mathcal{C};\theta\rangle[\![\exists x\ \phi]\!] & = & \bigcup_\sigma \{\mathit{infer}\ \mathrm{DROP}_u(\sigma)\} \text{ where, for some fresh } u, \\ & & \qquad \sigma \text{ ranges over } \mathit{cons}^+(\langle \mathcal{C};\theta\rangle[\![\phi\{x/u\}]\!]) \\
\mathrm{ERROR}[\![\phi]\!] & = & \{\mathrm{ERROR}\} \text{ for all } \phi
\end{array}
$$

The definition relies heavily on the notation that was introduced before. But it is still quite easy to see what goes on. The atomic formulas are handled by means of the *infer* mapping. Then, disjunction is interpreted as nondeterministic choice, and conjunction as sequential composition. For existential quantification we use the DROP_u mapping (for a fresh variable u). The ERROR clause says that there is no recovery from ERROR. In the case for negation, three contingencies are present: first, the case where ϕ is inconsistent. Then we continue with the input state $\langle \mathcal{C};\theta\rangle$. Secondly, the case where ϕ is already true in (a state equivalent to) the input state. Then we conclude that $\neg\phi$ yields inconsistence, i.e., we get $\emptyset$. Finally, we add $\neg\phi$ to the constraint store $\mathcal{C}$ if it is impossible at this point to reach a decision about the status of $\neg\phi$.
Next we show that the denotational semantics with the *infer* parameter is sound. This amounts to two things: *1.* successful computations of ϕ result in states in which ϕ holds; *2.* if no successful computation of ϕ exists, ϕ is false in the initial state.

Theorem 1 (Soundness). *Let $\langle \mathcal{C};\theta\rangle$ and ϕ be given. Then we have:*

1. *If $\langle \mathcal{C}';\theta'\rangle \in \langle \mathcal{C};\theta\rangle[\![\phi]\!]$, then $\langle \mathcal{C}';\theta'\rangle \models_{\mathcal{J}} \phi$*
2. *If $\mathit{cons}^+(\langle \mathcal{C};\theta\rangle[\![\phi]\!]) = \emptyset$, then $\langle \mathcal{C};\theta\rangle \models_{\mathcal{J}} \neg\phi$.*

[2] We also sneak in the notation: $\Sigma[\![\phi]\!]$ for $\bigcup_{\sigma\in\Sigma}\{\sigma[\![\phi]\!]\}$.

The proof of the theorem is by a simultaneous induction on the structure of the formula ϕ. In the proof we need a preservation/persistence result, that we give as a separate lemma.

Lemma 1 (Preservation/Persistence).

1. *If* $\langle \mathcal{C}; \theta \rangle \models_{\mathcal{J}} \phi_1$ *and* $\langle \mathcal{C}'; \theta' \rangle \in \langle \mathcal{C}; \theta \rangle [\![\phi_2]\!]$, *then* $\langle \mathcal{C}'; \theta' \rangle \models_{\mathcal{J}} \phi_1$ *(validity)*
2. *If* $\mathcal{C}\theta$ *and* $(\phi_1 \wedge \phi_2)\theta$ *are mutually consistent (in* $\mathcal{J}$*) and* $\langle \mathcal{C}'; \theta' \rangle \in cons(\langle \mathcal{C}; \theta \rangle [\![\phi_2]\!])$, *then* $\mathcal{C}'\theta'$ *and* $(\phi_1 \wedge \phi_2)\theta'$ *are mutually consistent (in* $\mathcal{J}$*).* *(consistency)*

The lemma says that computations of $[\![\phi_2]\!]$ will not disturb the status of $[\![\phi_1]\!]$: the computation preserves validity and consistency. The proof of the lemma is by a simultaneous induction on the structure of ϕ_2. Some proof details are given in the appendix. Here we continue by considering several instantiations of the general format.

4 Modeling the Denotational Semantics of Apt [2]

We start our analysis by recalling the semantics provided in [2]. The idea of this semantics is to provide a uniform computational meaning for the first-order formulas independent of the underlying interpretation and without a constraint store. This yields a limited way of processing formulas in the sense that occasionally an ERROR may arise. After we have reintroduced this semantics we shall discuss a number of its extensions, all involving a specific constraint store management. So, let us recall the relevant definitions.

Definition 1. *Assume a language of terms* L *and an algebra* $\mathcal{J}$ *for it.*

- *Consider a term of* L *in which we replace some of the variables by the elements of the domain* D. *We call the resulting object a* generalized term.
- *Given a generalized term* t *we define its* $\mathcal{J}$-evaluation *as follows. Each ground term of* s *of* L *evaluates to a unique value in* $\mathcal{J}$. *Given a generalized term* t *replace each maximal ground subterm of* t *by its value in* $\mathcal{J}$. *We call the resulting generalized term a* $\mathcal{J}$-term *and denote it by* $[\![t]\!]_{\mathcal{J}}$.
- *By a* $\mathcal{J}$-substitution *we mean a finite mapping from variables to* $\mathcal{J}$*-terms which assigns to each variable* x *in its domain a* $\mathcal{J}$*-term different from* x. *We write it as* $\{x_1/h_1, \ldots, x_n/h_n\}$. *We define the notion of an application of a* $\mathcal{J}$*-substitution* θ *to a generalized term* t *in the standard way and denote it by* $t\theta$.
- *A* composition of two $\mathcal{J}$-substitutions θ and η, *written as* $\theta\eta$, *is defined as the unique* $\mathcal{J}$*-substitution* γ *such that for each variable* x

$$x\gamma = [\![(x\theta)\eta]\!]_{\mathcal{J}}.$$

The $\mathcal{J}$-substitutions generalize both the usual substitutions and the valuations, which assign domain values to variables. After these introductory definitions we recall the semantics $[\![\cdot]\!]$ of an equation between two generalized terms (so *a fortiori*, between two terms). Here and elsewhere we do not indicate the dependency of the semantics on the underlying interpretation or algebra.

$$[\![s = t]\!](\theta) := \begin{cases} \{\theta\{s\theta/[\![t\theta]\!]_{\mathcal{J}}\}\} & \text{if } s\theta \text{ is a variable that does not occur in } t\theta, \\ \{\theta\{t\theta/[\![s\theta]\!]_{\mathcal{J}}\}\} & \text{if } t\theta \text{ is a variable that does not occur in } s\theta \\ & \text{and } s\theta \text{ is not a variable,} \\ \{\theta\} & \text{if } [\![s\theta]\!]_{\mathcal{J}} \text{ and } [\![t\theta]\!]_{\mathcal{J}} \text{ are identical,} \\ \emptyset & \text{if } s\theta \text{ and } t\theta \text{ are ground and } [\![s\theta]\!]_{\mathcal{J}} \neq [\![t\theta]\!]_{\mathcal{J}}, \\ \{\text{ERROR}\} & \text{otherwise.} \end{cases}$$

Consider now an interpretation $\mathcal{I}$ based on an algebra $\mathcal{J}$. Given an atomic formula $p(t_1, \ldots, t_n)$ different from $s = t$ and a $\mathcal{J}$-substitution θ we denote by $p_{\mathcal{I}}$ the interpretation of p in $\mathcal{I}$. We say that

- $p(t_1, \ldots, t_n)\theta$ is *true* if $p(t_1, \ldots, t_n)\theta$ is ground and $([\![t_1\theta]\!]_{\mathcal{J}}, \ldots, [\![t_n\theta]\!]_{\mathcal{J}}) \in p_{\mathcal{I}}$,
- $p(t_1, \ldots, t_n)\theta$ is *false* if $p(t_1, \ldots, t_n)\theta$ is ground and $([\![t_1\theta]\!]_{\mathcal{J}}, \ldots, [\![t_n\theta]\!]_{\mathcal{J}}) \notin p_{\mathcal{I}}$.

To deal with the existential quantification we use the $DROP_x$ operation defined in Section 2, extended in the standard way to the subsets of $Subs \cup \{error\}$. Now $[\![\cdot]\!]$ is defined by structural induction as follows. A is here an atomic formula different from $s = t$.

- $[\![A]\!](\theta) := \begin{cases} \{\theta\} & \text{if } A\theta \text{ is true,} \\ \emptyset & \text{if } A\theta \text{ is false,} \\ \{\text{ERROR}\} & \text{otherwise, that is if } A\theta \text{ is not ground,} \end{cases}$
- $[\![\phi_1 \wedge \phi_2]\!](\theta) := [\![\phi_2]\!]([\![\phi_1]\!](\theta))$,
- $[\![\phi_1 \vee \phi_2]\!](\theta) := [\![\phi_1]\!](\theta) \cup [\![\phi_2]\!](\theta)$,
- $[\![\neg\phi]\!](\theta) := \begin{cases} \{\theta\} & \text{if } [\![\phi]\!](\theta) = \emptyset, \\ \emptyset & \text{if } \theta \in [\![\phi]\!](\theta), \\ \{\text{ERROR}\} & \text{otherwise,} \end{cases}$
- $[\![\exists x\, \phi]\!](\theta) := DROP_u([\![\phi\{x/u\}]\!](\theta))$, where u is a fresh variable.

The following example clarifies the way we interpret atoms and conjunction.

Example 1. Assume the standard algebra for the language of arithmetic with the set of integers as domain. We denote its elements by $\ldots, -\mathbf{2}, -\mathbf{1}, \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots$. Each constant i evaluates to the element $\mathbf{i}$. We then have

1. $[\![y = z - 1 \wedge z = x + 2]\!](\{x/\mathbf{1}\}) = [\![z = x + 2]\!](\{x/\mathbf{1}, y/z - \mathbf{1}\}) = \{\{x/\mathbf{1}, y/\mathbf{2}, z/\mathbf{3}\}\}$,
2. $[\![y = 1 \wedge z = 1 \wedge y - 1 = z - 1]\!](\varepsilon) = \{\{y/\mathbf{1}, z/\mathbf{1}\}\}$,
3. $[\![y = 1 \wedge z = 2 \wedge y < z]\!](\varepsilon) = \{\{y/\mathbf{1}, z/\mathbf{2}\}\}$,
4. $[\![x = 0 \wedge \neg(x = 1)]\!](\varepsilon) = \{\{x/\mathbf{0}\}\}$,
5. $[\![y - 1 = z - 1 \wedge y = 1 \wedge z = 1]\!](\varepsilon) = \{\text{ERROR}\}$,

6. $[\![y < z \wedge y = 1 \wedge z = 2]\!](\varepsilon) = \{\text{ERROR}\}$,
7. $[\![\neg(x = 1) \wedge x = 0]\!](\varepsilon) = \{\text{ERROR}\}$.

So in this semantics the conjunction is not commutative and consequently it is important in which order the formulas are processed. This semantics is a special case of the semantics provided in Section 3. It is obtained by using the following *infer* relation:

- $\mathit{infer}\langle A \ ; \ \theta\rangle := \{\langle \emptyset \ ; \ \eta\rangle \ : \ \eta \in [\![A]\!](\theta)\}$ for an atomic formula A, where we identify $\langle \emptyset \ ; \ \text{ERROR}\rangle$ with ERROR,
- $\mathit{infer}\langle \mathcal{C} \ ; \ \theta\rangle := \{\text{ERROR}\}$ for all other states $\langle \mathcal{C} \ ; \ \theta\rangle$.

The relevant 'embedding' theorem is the following one.

Theorem 2 (Embedding).

- $\eta \in [\![\phi]\!](\theta)$ *iff* $\langle \emptyset \ ; \ \eta\rangle \in \langle \emptyset \ ; \ \theta\rangle[\![\phi]\!]$.
- $\text{ERROR} \in [\![\phi]\!](\theta)$ *iff* $\text{ERROR} \in \langle \emptyset \ ; \ \eta\rangle[\![\phi]\!]$.

5 Specific Constraint Store Managements

We now illustrate the generality of our approach by presenting various increasingly powerful forms of constraint store management. Each of them is obtained by a particular propagation *step* that works on *special states* and is executed whenever and as-long-as it can be applied. *aux* is our name for the maximal repetition of the *step*.[3] So, *aux* is a procedure on special states that is the least fixed point of $\mathit{aux} = \mathit{step} \circ \mathit{aux}$. Then we can define the *infer* mapping as follows:

$$\begin{array}{lcll} \mathit{infer}\ \mathbf{error} & = & \{\mathbf{error}\} & \\ \mathit{infer}\langle \emptyset; \theta\rangle & = & \{\langle \emptyset; \theta\rangle\} & \\ \mathit{infer}\langle \mathcal{C}; \theta\rangle & = & \mathit{aux}\ \langle \mathcal{C}; \theta\rangle & \text{for a special state } \langle \mathcal{C}; \theta\rangle \\ \mathit{infer}\langle \mathcal{C}; \theta\rangle & = & \{\mathbf{error}\} & \text{otherwise} \end{array}$$

Now the examples are obtained by a specification of the special states and the *step* procedure. In each case it is then straightforward to check that the adopted definition of *infer* satisfies the conditions we put on it in Section 2. Consequently, in each case the Soundness Theorem holds. Informally, in each case we provide a sound constraint store management.

[3] Note that maximal repetition of one *step* is just one strategy for constraint management. Already Jaffar and Maher [9] mention other options, distinguishing for example, *quick-checking*, *progressive* and *ideal CPL* systems. Of course, our set up can also accommodate such variations.

Equations as active constraints. Below, following Jaffar and Maher [9], we make a distinction between *active* and *passive* constraints. In our framework active constraints are the ones that are capable of changing the values of the variables, while the passive ones boil down to formulas that become tests after an appropriate instantiation.
As an example how active constraints can be modeled using the presented semantics consider unification as a way of solving equality constraints. To model it we choose as the underlying algebra the Herbrand algebra, the universe of which consists of the set of all ground terms of the language L.
The constraint stores of special states only contain equations. The equations are active, and each step consists of unification, whenever possible. So, we put:

$$step\langle \emptyset \ ; \ \theta \rangle := \emptyset$$

$$step\langle C, s = t \ ; \ \theta \rangle := \begin{cases} \{\langle C \ ; \ \theta\eta \rangle\} & \text{if } \eta \text{ is an mgu of } s\theta \text{ and } t\theta, \\ \emptyset & \text{if } s\theta \text{ and } t\theta \text{ are not unifiable.} \end{cases}$$

Other specific forms of active constraints can be modeled in our framework in an equally straightforward way.

Atoms as passive constraints. The drawback of the semantics defined in the previous section is that it yields **error** when a wrong order of conjuncts is accidentally chosen. A possible remedy is to use atoms as passive constraints, i.e., to move the atoms that currently evaluate to **error** to the constraint store instead.
For the handling of passive constraints we include a *split* procedure on special states to isolate the passive constraints: $split\langle C; \theta \rangle = \langle C_p, C_a; \theta \rangle$, where C_p is a list of the constraints that are passive when evaluated by θ and C_a is a list of the constraints that are active when evaluated by θ. When this is done, we perform a *step* on the active constraints. Next we re-group the constraints to reconsider the active-passive *split* in the new state. So, the step we perform in the auxiliary procedure is a composed action: $aux = splitstep \circ splitstep$ and $splitstep = split \circ step$.[4]
In the current example we set the *split* procedure as indicated: we regard the atoms that would evaluate to **error** passive. Then the *step* works as follows:

$$\begin{array}{lll} step\langle C_p; \theta \rangle & = \{\langle C_p; \theta \rangle\} & \text{if no active constraints occur} \\ step\langle C_p, C_a, s = t; \theta \rangle & = \{\langle C_p, C_a; \theta\eta \rangle\} & \text{if } \eta \text{ is an mgu of } s\theta, \ t\theta \\ step\langle C_p, C_a, s = t; \theta \rangle & = \emptyset & \text{if } s\theta, \ t\theta \text{ cannot be unified} \\ step\langle C_p, C_a, A; \theta \rangle & = \{\langle C_p, C_a; \theta \rangle\} & \text{if } A\theta \text{ is true} \\ step\langle C_p, C_a, A; \theta \rangle & = \emptyset & \text{if } A\theta \text{ is false} \end{array}$$

Then the $splitstep := split \circ step$ combines the two actions and aux repeats the *splitstep* until no more active constraints are left to remove. Reconsider now the formulas from items (5) and (6) of Example 1. We now have

[4] We ignore various implementation details regarding the particular choice of an active constraint and the distinction between lists and sets.

$\langle \emptyset \ ; \ \varepsilon \rangle [\![y - 1 = z - 1 \wedge y = 1 \wedge z = 1]\!] =$
$\langle y - 1 = z - 1 \ ; \ \varepsilon \rangle [\![y = 1 \wedge z = 1]\!] = \{ \langle \emptyset \ ; \ \{ y/\mathbf{1}, z/\mathbf{1} \} \rangle \}$

and

$\langle \emptyset \ ; \ \varepsilon \rangle [\![y < z \wedge y = 1 \wedge z = 2]\!] = \langle y < z \ ; \ \varepsilon \rangle [\![y = 1 \wedge z = 2]\!] =$
$\{ \langle \emptyset \ ; \ \{ y/\mathbf{1}, z/\mathbf{2} \} \rangle \}$.

This shows the difference brought in by this *infer* procedure. However, in the case of the formula from item (7), we still have

$$\langle \emptyset \ ; \ \varepsilon \rangle [\![\neg(x = 1) \wedge x = 0]\!] = \{ \mathbf{error} \}.$$

Equations as active and passive constraints. In general, equations can be both active and passive constraints. For example, linear equations over reals can be active and non-linear ones passive. To model computation in their presence we choose as the underlying algebra the standard algebra for the language of arithmetic with the set of real numbers as the domain. The special states are the ones that just have equations in the constraint store. Next, we use a *split* procedure that regards the linear equations as active and the non-linear ones as passive. Using standard arithmetic operations each linear equation can be rewritten into one of the following forms:

- $0 = 0$,
- $r = 0$, where r is a non-zero real, and
- $x = u$, where $x \in \mathit{Var}$ and u a linear expression not containing x.

This leads to the following definition of the propagation *step*:

$$\mathit{step}\langle \mathcal{C}_p, \emptyset \ ; \ \theta \rangle := \{ \langle \mathcal{C}_p \ ; \ \theta \rangle \}$$

$$\mathit{step}\langle \mathcal{C}_p, \mathcal{C}_a, s = t \ ; \ \theta \rangle := \begin{cases} \langle \mathcal{C}_p, \mathcal{C}_a \ ; \ \theta \rangle & s\theta = t\theta \text{ rewrites to } 0 = 0, \\ \emptyset & s\theta = t\theta \text{ rewrites to } r = 0, \\ & \text{where } r \text{ is a non-zero real}, \\ \langle \mathcal{C}_p, \mathcal{C}_a \ ; \ \theta\{x/u\} \rangle & s\theta = t\theta \text{ rewrites to } x = u, \end{cases}$$

The last clause models in effect the Gaussian elimination step, now in presence of linear and non-linear equations.

Negative literals as passive constraints. The *infer* methods introduced above allowed only atoms in the constraint store of special states, that is to say an occurrence of non-atomic formulas in the constraint store leads to an immediate error. Let us extend the *infer* method to allow for negative literals in the constraint store of special states. Now we can easily modify the definitions from "Atoms as passive constraints": we regard states with finite sets of literals as special states and regard the literals that would evaluate to **error** as passive. Then the definition of the *step* is obtained by having a literal L instead of an atom A. Now, in the case of the formula from item (7) of Example 1 we have

$\langle \emptyset \ ; \ \varepsilon \rangle [\![\neg(x = 1) \wedge x = 0]\!] = \langle \neg(x = 1) \ ; \ \varepsilon \rangle [\![x = 0]\!] =$
$\mathit{step}\langle \neg(x = 1) \ ; \ \{ x/\mathbf{0} \} \rangle = \{ \langle \emptyset \ ; \ \{ x/\mathbf{0} \} \rangle \}$.

Equality and disequality constraints. We continue the previous example for the case of an arbitrary language of terms together with equality and disequality constraints.[5] We adapt the definition by having as active constraints all equations as well as those disequations that are ground or of the form $t\theta \neq t\theta$. The *split* of $\langle \mathcal{C}; \theta \rangle$ now produces $\langle \mathcal{C}_p, \mathcal{C}_a; \theta \rangle$ with $\mathcal{C}_p = s_1 \neq t_1, \ldots, s_n \neq t_n$, a list of all the disequations $s_i \neq t_i \in \mathcal{C}$ for which $(s_i \neq t_i)\theta$ is not ground and not of the form $t \neq t$. The definition of the *step* then is:

$$
\begin{array}{lll}
step\langle \mathcal{C}_p; \theta \rangle & = \{\langle \mathcal{C}_p; \theta \rangle\} & \text{if no active constraints occur} \\
step\langle \mathcal{C}_p, \mathcal{C}_a, s = t; \theta \rangle & = \{\langle \mathcal{C}_p, \mathcal{C}_a; \theta\eta \rangle\} & \text{if } \eta \text{ is an mgu of } s\theta,\ t\theta \\
step\langle \mathcal{C}_p, \mathcal{C}_a, s = t; \theta \rangle & = \emptyset & \text{if } s\theta,\ t\theta \text{ cannot be unified} \\
step\langle \mathcal{C}_p, \mathcal{C}_a, s \neq t; \theta \rangle & = \{\langle \mathcal{C}_p, \mathcal{C}_a; \theta \rangle\} & \text{if } s\theta \neq t\theta \text{ is true} \\
step\langle \mathcal{C}_p, \mathcal{C}_a, s \neq t; \theta \rangle & = \emptyset & \text{if } s\theta \neq t\theta \text{ is false}
\end{array}
$$

Then we get, for example

$$
\begin{array}{l}
\langle \emptyset\ ;\ \varepsilon \rangle [\![f(x) \neq f(y) \wedge g(x,b) = g(a,y)]\!] = \\
step\langle f(x) \neq f(y)\ ;\ \{x/a, y/b\} \rangle = \{\langle \emptyset\ ;\ \{x/a, y/b\} \rangle\}.
\end{array}
$$

In general, if no **error** occurs, we can expect $\langle \emptyset,\ \epsilon \rangle [\![\phi]\!]$ to contain special states from which all active constraints are removed, i.e., states of the form $\langle \mathcal{C}; \theta \rangle$ where $\mathcal{C}$ is a list of inequations $s \neq t$ such that $s\theta \neq t\theta$ is passive. It follows from the independence of inequations of [6] that over an infinite Herbrand Universe such a constraint store is consistent, i.e., has a grounding solution η. For such an η we can then conclude: $\models s_i\theta \neq t_i\theta$ (for each $1 \leq i \leq n$) and $\models \phi\theta\eta$.
The grounding solution η can not be built up during the computation of $[\![\phi]\!]$. This is clear from the example $x \neq y \wedge x = c$. If we make the choice $\{x/d, x/c\}$ as a grounding solution for $x \neq y$ too soon, we are no longer able to deal with $x = c$ later on. Hence we can benefit from the independence of inequations only after the computation of $[\![x \neq y \wedge x = c]\!]$ has been completed.

Existential formulas as passive constraints. At this point only literals are allowed in the constraint store. We can easily extend the current store management to one in which also existential formulas are allowed in the constraint store. To this end we need some quantifier elimination procedure *elim* that is able to deal with at least some form of existential quantification. Then we can have $step := elim$.

Arbitrary formulas as passive constraints. The previous constraint store management can be extended by allowing arbitrary formulas in the constraint store. This makes sense as soon as we have some decision procedure *solve* that is able to deal with at least some type of negative formulas. Then we can have $step := solve$.

[5] We ignore the notational distinction between the disequation $s \neq t$ and the negation $\neg(s = t)$ for the moment.

6 Rationale and Related Work

As clarified in Section 4 the soundness result established here generalizes the appropriate result provided in [2]. The drawback of this semantics was that it yielded error as answer for several clearly satisfiable formulas, like the ones considered in the introduction.

Our interest in a semantics that models constraint management in a sound way stems from our attempts to add constraints to the programming language Alma-0 of Apt et al. [4]. Alma-0 extends imperative programming by features that support declarative programming. This language allows us to interpret the formulas of first-order logic (without universal quantification) as executable programs. In Apt and Schaerf [1] we proposed to extend Alma-0 by constraints but found that this led to situations in which the customary interpretation of the conditionals by means of the implication is unsound.

Using the above considerations we can provide a simple sound interpretation of the `IF B THEN S ELSE T END` statement. Namely, it is sufficient to interpret it in logic as $(B \wedge S) \vee (\neg B \wedge T)$, written in the Alma-0 syntax as `EITHER B; S ORELSE (NOT B); T END`. This interpretation requires that negative literals, here `NOT B`, are used as passive constraints. On the implementation level backtracking is then needed but the above interpretation can be reduced to the customary implementation of `IF B THEN S ELSE T END` if the condition `B` evaluates to **true** or **false** irrespectively of the constraint store.

As already mentioned in the introduction the modelling of the constraint store maintenance by means of an abstract *infer* mechanism is due to Jaffar and Maher [9]. In their framework the computation mechanism of constraint logic programming is modeled, so local variables (modeled by existential variables) and negation are absent but recursion is considered. Additionally, only conjunctions of atomic formulas are allowed as constraints.

In [10] several semantics for constraint logic programming are compared. In this paper a mapping *solv* is used that allows for inconsistency checks during the computation. *solv* can vary with the intended application, just like our *infer* parameter, but, unlike *infer*, it cannot model arbitrary constraint propagation steps. In fact, in [10] the constraint propagation steps take place only at the end of each the computation.

An alternative approach to model the essentials of constraint programming is provided by the concurrent constraint programming (ccp) approach pioneered by Saraswat [14] and Saraswat, Rinard and Panangaden [13]. In this scheme the programs can also be considered as formulas with the difference that the atomic **tell** and **ask** operations are present and that the parallel composition connective is present. The idea captured by this model is that the processes interact by means of a constraint system using the **tell** and **ask** operations. The constraint system is a set of constraints equipped with the entailment operation. The ccp programs can be written in a logical way by dropping the "**tell**" context around a constraint and by interpreting the **ask**(c) statement as the implication $c \rightarrow$. However, in spite of this logical view of ccp programs it is not clear how to interpret them as first-order formulas with the customary semantics. In Fages,

Ruet and Soliman [8] a logical semantics of ccp programs is given by interpreting them in intuitionistic linear logic. Both the denotational semantics for this language and the correctness (in the assertional style) of ccp programs were considered in a number of papers, see, e.g., de Boer et al. [7] and de Boer et al. [5]. How to add to this framework in a sound way negation was studied in Palamidessi, de Boer and Pierro [12]. By the nature of this approach the study of the constraint store management captured here by means of the *infer* mechanism is absent in this framework.

Acknowledgement. We thank Catuscia Palamidessi for helpful discussion on the subject of ccp programs.

References

1. K. R. Apt and A. Schaerf. The Alma project, or how first-order logic can help us in imperative programming. In E.-R. Olderog and B. Steffen, editors, *Correct System Design*, Lecture Notes in Computer Science 1710, pages 89–113, 1999.
2. K.R. Apt. A denotational semantics for first-order logic. In *Proc. of the computational logic conference (CL2000)*, Lecture Notes in Artificial Intelligence 1861, pages 53–69. Springer Verlag, 2000.
3. K.R. Apt and M.A. Bezem. Formulas as programs. In K.R. Apt, V.W. Marek, M. Truszcyński, and D.S. Warren, editors, *The Logic Programming Paradigm: A 25 Year Perspective*, pages 75–107, 1999.
4. K.R. Apt, J. Brunekreef, V. Partington, and A. Schaerf. Alma-0: An imperative language that supports declarative programming. *ACM Toplas*, 20(5):1014–1066, 1998.
5. F.S. De Boer, M. Gabbrielli, E. Marchiori, and C. Palamidessi. Proving concurrent constraint programs correct. In *ACM Transactions on Programming Languages and Systems*, volume 19(5), pages 685–725, 1997.
6. A. Colmerauer. Equations and inequations on finite and infinite trees. In John Lloyd, editor, *Proc. of International Conference of Fifth Generation Computer Systems (FGCS'84)*, pages 85–99. OHMSHA Ltd. Tokyo and North-Holland, 1984.
7. F.S. de Boer, A. Di Pierro, and C. Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theoretical Computer Science*, 151(1):37–78, 1995.
8. F. Fages, P. Ruet, and S. Soliman. Linear concurrent constraint programming: Operational and phase semantics. *Information and Computation*, 165(1):14–41, 2001.
9. J. Jaffar and J.M. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19/20, 1994.
10. J. Jaffar, J.M. Maher, K. Marriott, and P. Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1):1–46, 1998.
11. J.W. Lloyd and R.W. Topor. Making Prolog more expressive. *Journal of Logic Programming*, 1:225–240, 1984.
12. C. Palamidessi, F.S. de Boer, and A. Di Pierro. An algebraic perspective of constraint logic programming. *Journal of Logic and Computation*, 7, 1997.

13. V. A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 333–352, Orlando, Florida, 1991.
14. Vijay Saraswat. *Concurrent Constraint Programming.* MIT Press, 1993.

Appendix: The Proofs

In this appendix we give proof details of the Soundness Theorem and Preservation/Persistence Lemma. Both proofs are by simultaneous inductions on the structure of the formula. We focus on the existential quantification cases, which are the most subtle. We use the notation $\models_{\mathcal{J}} \phi[\overline{a}]$ to indicate the assignment of values $\overline{a}$ to (at least) the free variables in ϕ. In the case of the lemma it will be convenient to standardize this as follows: we are concerned with $\mathcal{C}\theta$ and $\phi_i\theta$ (for $i = 1, 2$) and denote the values for the free variables shared by the $\phi_i\theta$ by $\overline{d}$. Then we use the values $\overline{c}$ for the remaining free variables in $\phi_1\theta$ and $\overline{e}$ for the remaining free variables in $\phi_2\theta$. Finally, we denote the values of the remaining free variables in $\mathcal{C}\theta$ by $\overline{b}$. So, we will mostly use blocks of the form $[\overline{b}, \overline{c}, \overline{d}, \overline{e}]$.

Proof of Preservation/Persistence Lemma 1:
∘ **atoms**: In the atomic case $\phi_2 = A$ for some atom A and $\langle \mathcal{C}''; \theta'' \rangle \in \mathit{infer}\langle \mathcal{C}, A; \theta \rangle$. Straightforward application of property (1) does the trick.
∘ **disjunction**: In this case $\phi_2 = (\psi_1 \vee \psi_2)$ and $\langle \mathcal{C}''; \theta'' \rangle \in \langle \mathcal{C}; \theta \rangle [\![\psi_i]\!]$ for some $i = 1, 2$. In this situation the inductive hypotheses apply straightforwardly.
∘ **conjunction**: In this case $\phi_2 = (\psi_1 \wedge \psi_2)$ and $\langle \mathcal{C}''; \theta'' \rangle \in \langle \mathcal{C}'; \theta' \rangle [\![\psi_2]\!]$ for some $\langle \mathcal{C}'; \theta' \rangle \in \langle \mathcal{C}; \theta \rangle [\![\psi_1]\!]$. Now two applications of the inductive hypothesis are required. For preservation of validity this is straightforward. For preservation of consistency it works as follows: by assumption $\models_{\mathcal{J}} (\mathcal{C}\theta, (\phi_1 \wedge (\psi_1 \wedge \psi_2))\theta)[\overline{b}, \overline{c}, \overline{d}, \overline{e}]$. So, $\models_{\mathcal{J}} (\mathcal{C}\theta, (\phi_1 \wedge \psi_1)\theta)[\overline{b}, \overline{c_1}, \overline{d_1}, \overline{e_1}]$, restricting the $\overline{d}$ and $\overline{e}$ to the relevant variables and moving some of the $\overline{d}$ values to $\overline{c_1}$. By induction hypothesis we get $\models_{\mathcal{J}} (\mathcal{C}'\theta', (\phi_1 \wedge \psi_1)\theta')[\overline{b_1'}, \overline{c_1'}, \overline{d_1'}, \overline{e_1'}]$. Next the induction hypothesis (for $\phi_1 \wedge \psi_1$ and ψ_2) provides $\models_{\mathcal{J}} (\mathcal{C}''\theta'', ((\phi_1 \wedge \psi_1) \wedge \psi_2)\theta'')[\overline{b''}, \overline{c''}, \overline{d''}, \overline{e''}]$, as required.
∘ **negation**: In this case $\phi_2 = \neg\psi$. We have to distinguish cases

- ϕ_2 is 'true': $\langle \mathcal{C}''; \theta'' \rangle \in \mathit{infer}\langle \mathcal{C}; \theta \rangle$ (and $\mathit{cons}^+(\langle \mathcal{C}; \theta \rangle [\![\psi]\!]) = \emptyset$.) Now property (1) gives the results.
- ϕ_2 is 'false': in this case $\langle \mathcal{C}; \theta \rangle [\![\phi_2]\!] = \emptyset$ and both (i) and (ii) are void.
- 'otherwise': we get $\langle \mathcal{C}''; \theta'' \rangle \in \mathit{infer}\langle \mathcal{C}, \neg\psi; \theta \rangle$. The results follow from first order logic and property (1).

∘ **existential quantification**: Now $\phi_2 = \exists x\ \psi$ for some x, ψ. So, consider *infer* $\mathrm{DROP}_u\langle \mathcal{C}'; \eta \rangle$ for some consistent $\langle \mathcal{C}'; \eta \rangle \in \langle \mathcal{C}; \theta \rangle [\![\psi\{x/u\}]\!]$ (u fresh). By property (1) it suffices to consider $\mathrm{DROP}_u\langle \mathcal{C}'; \eta \rangle = \langle (\mathcal{C}' - \mathcal{C}'(u, \mathbf{y})),\ \exists u\ (u = u\eta \wedge \mathbf{y} = \mathbf{y}\eta \wedge \mathcal{C}'(u, \mathbf{y}));\ \mathit{DROP}_u(\eta) \rangle$. We call $\mathit{DROP}_{u,\mathbf{y}}(\eta) = \theta''$.

1. By assumption $\langle \mathcal{C}; \theta \rangle \models_{\mathcal{J}} \phi_1$. By induction hypothesis $\langle \mathcal{C}'; \eta \rangle \models_{\mathcal{J}} \phi_1$. From this $(u = u\eta \wedge \mathcal{C}')\mathit{DROP}_u(\eta) \models_{\mathcal{J}} (u = u\eta \wedge \phi_1)\mathit{DROP}_u(\eta)$. So, $(u = u\eta \wedge$

$C')DROP_u(\eta) \models_{\mathcal{J}} \phi_1 DROP_u(\eta)$. Repeating this for the $\mathbf{y}$, we get $(u = u\eta \wedge \mathbf{y} = \mathbf{y}\eta \wedge C')\theta'' \models_{\mathcal{J}} \phi_1\theta''$. By (2) we may assume that this holds for some u that does not occur in $\phi_1\theta''$. Hence the implicit overall universal quantification over u can be replaced by an existential quantification over u on the left hand side of the sequent. This gives

$((C' - C'(u,\mathbf{y})) \wedge \exists u\,(u = u\eta \wedge \mathbf{y} = \mathbf{y}\eta \wedge C'(u,\mathbf{y})))\theta'' \models_{\mathcal{J}} \phi_1\theta''$.

Now we can safely re-instantiate the values of the variables in $\mathbf{y}$ to obtain the same for $DROP_u(\eta) = \theta'' \cup \{\mathbf{y}/\mathbf{y}\eta\}$

$((C' - C'(u,\mathbf{y})) \wedge \exists u\,(u = u\eta \wedge \mathbf{y} = \mathbf{y}\eta \wedge C'(u,\mathbf{y})))DROP_u(\eta) \models_{\mathcal{J}} \phi_1 DROP_u(\eta)$.

2. The assumption gives $\models_{\mathcal{J}} (C\theta,\ (\phi_1 \wedge \exists x\ \psi))\theta)[\overline{b}, \overline{c}, \overline{d}, \overline{e}]$. From the induction hypothesis we obtain $\models_{\mathcal{J}} (C'\eta,\ (\phi_1 \wedge \psi\{x/u\})\eta)[\overline{b'}, \overline{c'}, \overline{d'}, \overline{e'}, f]$ where f is the value of u. From this we conclude $\models_{\mathcal{J}} (C'\eta,\ (\phi_1 \wedge \exists x\ \psi)\eta)[\overline{b'}, \overline{c'}, \overline{d'}, \overline{e'}, f]$. So, we can be sure that suitable values for all the variables in C' and $(\phi_1 \wedge \exists x\ \psi)$ are available, if we use the values in the block $[\overline{b'}, \overline{c'}, \overline{d'}, \overline{e'}, f]$ and the substitution η as a middle man. But then we can also assign these values directly to the variable u, eliminating the middle man η. This way we get values $[\overline{b''}, \overline{c''}, \overline{d''}, \overline{e''}]$ such that $\models_{\mathcal{J}} (((C' - C'(u,\mathbf{y})) \wedge u = u\eta \wedge \mathbf{y} = \mathbf{y}\eta \wedge C'(u,\mathbf{y}))DROP_u(\eta),\quad (\phi_1 \wedge \exists x\ \psi)DROP_u(\eta))[\overline{b''}, \overline{c''}, \overline{d''}, \overline{e''}]$. From this the consistency of $\mathrm{DROP}_u\langle C';\eta\rangle$ and $(\phi_1 \wedge \exists x\ \psi)DROP_u(\eta)$ is clear. □

Proof of Soundness Theorem 1:

∘ **atoms:** In case ϕ is an atomic formula A, $\langle C;\theta\rangle[\![\phi]\!] = \mathit{infer}\ \langle C, A;\theta\rangle$. Now straightforward applications of property (1) and (3) give the result.

∘ **disjunction:** In case ϕ is a disjunction, $\phi_1 \vee \phi_2$ say, $\langle C;\theta\rangle[\![\phi]\!] = \langle C;\theta\rangle[\![\phi_1]\!] \cup \langle C;\theta\rangle[\![\phi_2]\!]$. The induction hypotheses apply immediately.

∘ **conjunction:** In case ϕ is a conjunction, $\phi_1 \wedge \phi_2$ say, $\langle C'';\theta''\rangle \in \langle C;\theta\rangle[\![\phi]\!]$ iff $\langle C'';\theta''\rangle \in \langle C';\theta'\rangle[\![\phi_2]\!]$ for some $\langle C';\theta'\rangle \in \langle C;\theta\rangle[\![\phi_1]\!]$. Part 1. of the theorem is a straightforward consequence of the induction hypothesis and persistence. For part 2 we add some details. We have: if $\langle C';\theta'\rangle \in \langle C;\theta\rangle[\![\phi_1]\!]$ is consistent, then $\langle C';\theta'\rangle[\![\phi_2]\!]$ only contains inconsistent states. From this we may conclude by induction hypothesis that for each $\langle C';\theta'\rangle \in \mathit{cons}(\langle C;\theta\rangle[\![\phi_1]\!])$

$$\langle C';\theta'\rangle \models_{\mathcal{J}} \neg\phi_2. \tag{3}$$

Now assume that for some[6] $[\overline{b}, \overline{c}, \overline{d}, \overline{e}]$, $\models_{\mathcal{J}} (C\theta \wedge \phi_1\theta \wedge \phi_2\theta)[b, c, d, e]$ and that we have a $\langle C';\theta'\rangle \in \mathit{cons}(\langle C;\theta\rangle[\![\phi_1]\!])$. Then persistence (2) tells us that the consistency is preserved, i.e., there are $[\overline{b'}, \overline{c'}, \overline{d'}, \overline{e'}]$ such that $\models_{\mathcal{J}} (C'\theta' \wedge \phi_1\theta' \wedge \phi_2\theta')[\overline{b'}, \overline{c'}, \overline{d'}, \overline{e'}]$. But this contradicts the statement (3). So, for no $[\overline{b}, \overline{c}, \overline{d}, \overline{e}]$, $\models_{\mathcal{J}} (C\theta \wedge \phi_1\theta \wedge \phi_2\theta)[\overline{b}, \overline{c}, \overline{d}, \overline{e}]$, which is as required.

∘ **negation:** In case of a negation $\neg\phi$, there are three situations to consider

- $\langle C';\theta'\rangle \in \mathit{infer}\langle C;\theta\rangle$ and $\mathit{cons}^+(\langle C;\theta\rangle[\![\phi]\!]) = \emptyset$. Now case (1) of the theorem follows from the induction hypothesis for case (2) and equivalence condition (1). Case (2) follows from conditions (1) and (3).

[6] Notation for assignment of values as in the lemma.

- $\langle \mathcal{C}; \theta \rangle [\![\neg\phi]\!] = \emptyset$ and there is some $\langle \mathcal{C}'; \theta' \rangle \in \mathit{cons}(\langle \mathcal{C}; \theta \rangle [\![\phi]\!])$, which is equivalent to $\langle \mathcal{C}; \theta \rangle$. Now case (1) is satisfied trivially and case (2) follows from the induction hypothesis for (1) and condition (1) on *infer*.
- $\langle \mathcal{C}; \theta \rangle [\![\neg\phi]\!] = \mathit{infer}\langle \mathcal{C}, \neg\phi; \theta \rangle$. Let $\langle \mathcal{C}'; \theta' \rangle \in \mathit{infer}\langle \mathcal{C}, \neg\phi; \theta \rangle$ be given. Now case (1) follows from condition (1) and case two relies on conditions (1) and (3).

∘ **existential quantification**: In case of an existential quantification $\exists x\ \phi$, we have to consider $\langle \mathcal{C}''; \theta'' \rangle \in \mathit{infer}\ \mathrm{DROP}_u \langle \mathcal{C}'; \eta \rangle$, for $\langle \mathcal{C}'; \eta \rangle \in \mathit{cons}(\langle \mathcal{C}; \theta \rangle [\![\phi\{x/u\}]\!])$ (some fresh u). Call $\mathit{DROP}_{u,\mathbf{y}}(\eta) = \theta'$. Below we use a crucial fact about first order logic: if x is not free in χ, then $\models \forall x\ (\psi \to \chi) \leftrightarrow ((\exists x\ \psi) \to \chi)$.

1. By induction hypothesis $\langle \mathcal{C}'; \eta \rangle \models_{\mathcal{J}} \phi\{x/u\}$. By first order logic $(\mathcal{C}' - \mathcal{C}'(u, \mathbf{y}))\eta \models_{\mathcal{J}} (\mathcal{C}'(u, \mathbf{y}) \to \phi\{x/u\})\eta$. From this we conclude that $(\mathcal{C}' - \mathcal{C}'(u, \mathbf{y}))\eta \models_{\mathcal{J}} (\mathcal{C}'(u, \mathbf{y}) \to \exists x\ \phi)\eta$. $(\mathcal{C}' - \mathcal{C}'(u, \mathbf{y}))$ does not contain u or $\mathbf{y}$, so: $(\mathcal{C}' - \mathcal{C}'(u, \mathbf{y}))\theta' \models_{\mathcal{J}} (\mathcal{C}'(u, \mathbf{y}) \wedge u = u\eta \wedge \mathbf{y} = \mathbf{y}\eta \to \exists x\ \phi)\theta'$. As u does not occur in $(\exists x\ \phi)\theta'$, we can apply the crucial fact to get $(\mathcal{C}' - \mathcal{C}'(u, \mathbf{y}))\theta' \wedge \exists u\ (\mathcal{C}'(u, \mathbf{y}) \wedge u = u\eta \wedge \mathbf{y} = \mathbf{y}\eta)\theta' \models_{\mathcal{J}} (\exists x\ \phi)\theta'$. Now we can make θ' more specific by re-instantiating the values $\mathbf{y}\eta$ for the variables in $\mathbf{y}$. This suffices (by (1)).
2. In this case there is no fresh u which produces a $\langle \mathcal{C}'; \eta \rangle \in \mathit{cons}(\langle \mathcal{C}; \theta \rangle [\![\phi\{x/u\}]\!])$. The induction hypothesis then gives $\langle \mathcal{C}; \theta \rangle \models_{\mathcal{J}} (\neg\phi\{x/u\})$ (for all fresh u), from which $\langle \mathcal{C}; \theta \rangle \models_{\mathcal{J}} (\exists x\ \phi)$ (as u is fresh w.r.t. θ). □