Compiling GOAL Agent Programs into Jazzyk Behavioural State Machines

Koen Hindriks[1] and Peter Novák[2]

[1] EEMCS, Delft University of Technology, The Netherlands
k.v.hindriks@tudelft.nl
[2] Department of Informatics, Clausthal University of Technology, Germany
peter.novak@tu-clausthal.de

Abstract. A variety of agent-oriented programming languages based on concepts such as beliefs and goals has been proposed in the literature. Even though most of these languages now come with interpreters implemented in e.g. Java and can be used to write software agents, there is little work reporting how to implement such languages or to identify a core instruction set that would facilitate such implementation. In this paper we introduce a compiler for the language *GOAL* into the framework of *Jazzyk Behavioural State Machines*. The result is a translation of key agent concepts such as beliefs and goals into *Jazzyk* which lacks these notions, thus providing some evidence that it may provide a sufficient instruction set for implementing agent programs. Moreover, arguably, the implementation strategy used can be applied also to other agent programming languages.

1 Introduction

Relatively little has been reported in the literature on implementing high-level agent programming languages [1]. An exception is the work of Dennis et al. [6], which aims at providing a common basis for a variety of such languages. As of yet, however, there is no equivalent of the *Warren Abstract Machine* [13] available - which provides such a basis for *Prolog* - that would facilitate implementation of these agent languages. In part this is due to the diversity of the proposed languages, ranging from extensions of Java with high-level agent concepts to completely new proposals for high-level agent-oriented programming languages. The effort needed, however, to implement the latter class of agent languages from scratch, in for example Java, is large, non-trivial and error-prone. Moreover a disadvantage of such an effort is that it is difficult to ascertain that such an implementation is a faithful implementation of the semantics. It therefore would be useful to have an *intermediate language that provides a core instruction set* of more high-level programming constructs than e.g. Java provides, and that could be used to compile agent programs into. As we will show, it turns out that the *Jazzyk* agent programming framework [8, 9] provides an interesting option for compiling agent programs. *Jazzyk* agents are *Behavioural State Machines* (*Jazzyk BSM*) that exactly provide the behavioural layer on top of a knowledge representational layer that is needed to implement agent languages. The main contribution of the paper is a formal proof that shows it is relatively easy to compile *GOAL* agents [5, 7] into *Jazzyk BSM*, demonstrating the usefulness of *Jazzyk* as a target language of an agent program compiler.

Besides showing that *Jazzyk* can be used as a target language of a compiler for such agents, our result provides some additional insights. One of the more important corollaries of the proof given is that it shows that the *GOAL* agent language is not committed

to any particular knowledge representation (KR) technology. *GOAL* agents may use *Prolog* [12], but there is nothing specific about *GOAL* enforcing such a choice. One of the motivations behind the *Jazzyk* language has been to allow the use and combination of heterogeneous KR technologies in a single agent. A consequence of our result is that the choice of the KR technology used by *GOAL* agents can be seen as a parameter to be instantiated when these agents are written. In fact, our result shows in a formally precise sense that an agent language such as *GOAL* can be viewed as an action selection mechanism put on top of an arbitrary knowledge representation technology. Finally, by showing that *GOAL* agents can be compiled into *Jazzyk*, some evidence is provided that *Jazzyk* supports the core functionality needed for implementing agent-oriented programming.

Since a key ingredient of agent languages are the KR technology(ies) used, for our purpose, we need to clarify in detail what we mean by a KR technology.

Definition 1 (KR Technology). *A* KR technology *is a triple* $\langle \mathcal{L}, \mathcal{Q}, \mathcal{U} \rangle$*, where:*

- $\mathcal{L}$ *is some logical language, with a typical element* $\phi \in \mathcal{L}$*,*
- $\mathcal{Q}$ *is a set of query operators* $\models \in \mathcal{Q}$ *such that* $\models \subseteq 2^{\mathcal{L}} \times \mathcal{L}$*,*
- $\mathcal{U}$ *is a set of update operators* $\oslash \in \mathcal{U}$ *of type* $: 2^{\mathcal{L}} \times \mathcal{L} \rightarrow 2^{\mathcal{L}}$*.*

Our definition of a KR technology is quite abstract and only specifies the types of operators which are associated with a knowledge representation language. This makes our result general, since it allows for a wide range of KR technologies that fit the KR schema introduced, such as *Prolog*, *Answer Set Programming*, *SQL*, etc. The only assumption made is that a special symbol $\bot$ is part of the KR language $\mathcal{L}$, which is intuitively interpreted as *falsum*; when $\bot$ can be derived from a set of sentences this set is said to be *inconsistent*. Our definition is inspired by [4] and explained in more detail in [3]. Apart from minor differences, it corresponds to the notion of a KR module in [8].

2 GOAL

The agent programming language *GOAL*, for Goal-Oriented Agent Language, is a language that incorporates declarative notions of beliefs and goals, and a mechanism for action selection based on these notions. That is, *GOAL* agents derive their choice of action from their beliefs and goals. A *GOAL* agent consists of four sections: (1) a set of beliefs, collectively called the *belief base*, typically denoted by Σ, (2) a set of goals, called the *goal base*, typically denoted by Γ, (3) a *program section* which consists of a set of *action rules*, typically denoted by Π, and (4) an *action specification section* that consists of a specification of the pre- and postconditions of actions of the agent, typically denoted by A. A *GOAL* agent $\mathcal{A}$ thus can be represented as a tuple $\mathcal{A} = \langle \Sigma, \Gamma, \Pi, A \rangle$. See Figure 1 below for a simplified *GOAL* agent that manipulates blocks on a table; for other examples and a more extensive discussion of *GOAL* we refer the reader to [5, 7].

Beliefs and Goals The beliefs and goals of a *GOAL* agent are drawn from a *KR language* such as *Prolog* [12]. As mentioned, one of the contributions will be to show that *GOAL* agents are not married to *Prolog*. To this end, we abstract here from particulars of a specific KR language (similar to the abstraction presented in e.g. [5]). Instead, we use the abstract definition of a KR technology provided in Definition 1. For the purpose

of introducing *GOAL* agents below and to simplify the technical presentation, without loss of generality, we introduce a slightly more specific instance of a KR Technology $\mathcal{K}_0 = \langle \mathcal{L}, \{\models\}, \{\oplus, \ominus\} \rangle$ where $\models$ is an entailment relation on $\mathcal{L}$, $\oplus$ is a revision operator and $\ominus$ is a contraction operator. In the remainder of this paper we will use the label $\mathcal{K}_0$ to refer to arbitrary KR technologies of this form *used by GOAL agents*. The notation used for the operators has been chosen to suggest the usual meaning associated with these symbols: $\models$ is used to verify that a sentence follows from a particular set of sentences; $\oplus$ is used to (consistently) add to a given set of sentences a new sentence; and $\ominus$ is used to remove (contracts) a sentence from a given set of sentences. Both $\oplus$ and $\ominus$ are assumed to yield consistent sets of sentences, i.e. $T \oplus \phi \not\models \bot$ and $T \ominus \phi \not\models \bot$.

The belief base Σ and the goal base Γ of a *GOAL* agent are defined as subsets of sentences from the KR language $\mathcal{L}$. Together the belief and the goal base make up a *mental state* m of a GOAL agent, i.e. $m = \langle \Sigma, \Gamma \rangle$. Belief bases Σ and individual goals $\gamma \in \Gamma$ are required to be consistent, i.e. $\Sigma \not\models \bot$ and $\{\gamma\} \not\models \bot$. Additionally, an agent does not believe it achieved its goals, i.e. for all $\gamma \in \Gamma$ we have $\Sigma \not\models \gamma$.

Action Selection and Specification A *GOAL* agent chooses an action by means of a rule-based action selection mechanism. A program section in a *GOAL* agent consists of *action rules* of the form $\textbf{if } \psi \textbf{ then } \mathtt{a}$. These action rules define a mapping from states to actions, together specifying a non-deterministic policy or course of action. The condition of an action rule, typically denoted by ψ, is called a *mental state condition*. It determines the states in which the action $\mathtt{a}$ may be executed. Mental state conditions are Boolean combinations of basic formulae $\mathbf{bel}(\phi)$ or $\mathbf{goal}(\phi)$ with $\phi \in \mathcal{L}$. For example, $\neg\mathbf{bel}(\phi_0) \wedge \mathbf{goal}(\phi_0 \wedge \phi_1)$ is a mental state condition.

Definition 2 (Mental State Condition Semantics). *The semantics of a mental state condition, given a mental state $m = \langle \Sigma, \Gamma \rangle$, is defined by the following four clauses:*

$$
\begin{array}{lll}
m \models_g \mathbf{bel}(\phi) & \textit{iff} & \Sigma \models \phi, \\
m \models_g \mathbf{goal}(\phi) & \textit{iff} & \textit{there is a } \gamma \in \Gamma \textit{ s.t. } \{\gamma\} \models \phi, \\
m \models_g \neg\psi & \textit{iff} & m \not\models_g \psi, \\
m \models_g \psi_1 \wedge \psi_2 & \textit{iff} & m \models_g \psi_1 \textit{ and } m \models_g \psi_2.
\end{array}
$$

Actions are specified in *GOAL* using a STRIPS-like specification. The action specification section in a *GOAL* agent consists of specifications of the form:

$$\mathbf{action}\,\{\ \textbf{:pre}\{\phi\}\ \ \textbf{:post}\{\phi'\}\ \}$$

Such a specification of action $\mathbf{action}$ consists of a precondition ϕ and a postcondition ϕ'. An action is *enabled* whenever the agent believes the precondition to be true. Upon its execution the agent updates its beliefs (and, indirectly, possibly also its goals) with the postcondition ϕ'. In line with STRIPS-style action specifications we assume that the postcondition ϕ' of an action consists of two parts $\phi' = \phi_d \wedge \phi_a$ with ϕ_d a list of negative literals (negated facts) also called the *delete list* and ϕ_a a conjunction of positive literals (facts) also called the *add list*.[3] It is assumed here that each action matches with exactly one corresponding action specification.

[3] We could also have used e.g. ADL specifications [10], but for reasons of simplicity we use a STRIPS-like specification, which also nicely matches the KR technology $\mathcal{K}_0$ with two update operators: the operator $\oplus$ to add facts, and the operator $\ominus$ to delete facts.

Semantics of a GOAL Agent To specify what it means to execute a *GOAL* agent we use a transition style semantics [11]. For our purposes, it is sufficient to present the semantics for executing a single action by a *GOAL* agent. In Section 4 we show how this semantics can be *implemented* by means of a *Jazzyk BSM*.

Definition 3 (Action Semantics). *Let* $m = \langle \Sigma, \Gamma \rangle$ *be a mental state,* **if** ψ **then** a *be an action rule, and* a $\{$**:pre**$\{\phi\}$ **:post**$\{\phi_a \wedge \phi_d\}$ $\}$ *be a corresponding action specification of a GOAL agent. The following semantic rule can be used to derive that action* a *can be executed:*

$$\frac{m \models \psi, \Sigma \models \phi}{m \xrightarrow{\texttt{a}} m'}$$

where $\Sigma' = (\Sigma \ominus \phi_d) \oplus \phi_a$ *and* $m' = \langle \Sigma', \Gamma \setminus \{\gamma \in \Gamma | \Sigma' \models \gamma\} \rangle$.

Besides user specified actions, *GOAL* has two built-in actions **adopt** and **drop** to modify an agent's goal base. The following axioms define the semantics of these actions:

$$\langle \Sigma, \Gamma \rangle \xrightarrow{\mathbf{adopt}(\phi)} \langle \Sigma, \Gamma \cup \{\phi\} \rangle$$

$$\langle \Sigma, \Gamma \rangle \xrightarrow{\mathbf{drop}(\phi)} \langle \Sigma, \Gamma \setminus \{\gamma \in \Gamma \mid \{\gamma\} \models \phi\} \rangle$$

3 Jazzyk Behavioural State Machines

The programming language *Jazzyk* introduced in [8, 9] elegantly combines concepts for programming *agent behaviour* with concepts for *knowledge representation*. *Jazzyk* agents can be seen as concrete instantiations of *Gurevich's Abstract State Machines* (*ASM*) [2] , named *Jazzyk Behavioural State Machines*, or alternatively *Jazzyk agents*. *Jazzyk* defines a new and unique agent-oriented programming language due to the clear distinction it makes between the *knowledge representation* and *behavioural* layers within an agent. It thus provides a programming framework that clearly separates the programming concerns of *how to represent an agent's knowledge* about, for example, its environment and *how to encode its behaviours*.

Mental states of *Jazzyk BSM* agents, different from those in *GOAL*, are collections of one or more so-called *knowledge representation modules*, typically denoted by $\mathcal{M}$, each of which represents part of the agent's knowledge base. Transitions between such states result from applying so-called *mental state transformers* (*mst*), typically denoted by τ. The various types of *mst* determine the behaviour that an agent can generate. A *Jazzyk BSM agent* $\mathcal{B}$ consists of a set of KR modules $\mathcal{M}_1, \ldots, \mathcal{M}_n$ and a mental state transformer τ, i.e. $\mathcal{B} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$; the *mst* τ is also called an *agent program*.

A KR module of a *Jazzyk BSM* can be seen as a database of statements drawn from a specific KR language. KR modules may be used to represent and maintain various attitudes of an agent such as its knowledge about its environment, or its goals, intentions, obligations, etc. *Jazzyk* allows agents to have any number of such KR modules and does not enforce any particular view on these modules. Unlike *GOAL*, *Jazzyk* abstracts from a particular purpose a KR module can be made to serve. Formally, a KR module $\langle D, \mathcal{L}, \mathcal{Q}, \mathcal{U} \rangle$ is a KR technology $\langle \mathcal{L}, \mathcal{Q}, \mathcal{U} \rangle$ (cf. Definition 1) extended with a

state (knowledge base) $D \subseteq \mathcal{L}$. A KR module is a self-encapsulated computational entity providing two sets of interfaces: *query* operators for querying the knowledge base and *update* operators to modify it. In a *Jazzyk BSM* $(\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$ we additionally require that the set of query and update operators of any two modules are disjoint, i.e. $\mathcal{Q}_i \cap \mathcal{Q}_j = \emptyset$ and $\mathcal{U}_i \cap \mathcal{U}_j = \emptyset$.

Syntax of Queries and Mental State Transformers Queries, typically denoted by φ, are operators constructed from the set of available query operators $\mathcal{Q}$ that are available in a KR technology. A primitive query $\varphi = (\models \phi)$ consists of a query operator $\models \in \mathcal{Q}$ and a formula $\phi \in \mathcal{L}$ of the same KR technology. Arbitrary queries can be composed again by means of conjunction $\wedge$, disjunction $\vee$ and negation $\neg$. Mental state transformers enable transitions from one state to another. A primitive *mst* $\oslash\phi$, typically denoted by ρ and constructed from an update operator $\oslash \in \mathcal{U}$ and a formula $\phi \in \mathcal{L}$, is an update on the state of the corresponding KR module of a mental state. Conditional *mst* are of the form $\varphi \longrightarrow \tau$, where φ is a query and τ is a *mst*. Such a conditional *mst* allows to make the application of *mst* τ conditional on the evaluation of query φ. *Mst*s can be combined by means of the choice $|$ and the sequence $\circ$ syntactic constructs.

Definition 4 (Jazzyk Mental State Transformer). *Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules of the form $\langle D_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i \rangle$. The set of* mental state transformers *is defined as:*

1. **skip** *is a* primitive *mst,*
2. *if $\oslash \in \mathcal{U}_i$ and $\phi \in \mathcal{L}_i$, then $\oslash\phi$ is a* primitive *mst,*
3. *if φ is a query, and τ is a mst, then $\varphi \longrightarrow \tau$ is a* conditional *mst,*
4. *if τ and τ' are mst's, then $\tau|\tau'$ is an mst (*choice*) and $\tau \circ \tau'$ is an mst (*sequence*).*

Figure 1 provides an example of a *Jazzyk BSM* agent. To improve readability, we use a mix of concrete *Jazzyk* syntax and the formal syntax introduced above. For a more extensive example of a *Jazzyk BSM* program see [9].

Jazzyk BSM Semantics The semantics of *Jazzyk BSM* is defined using a semantic calculus similar to that used for *ASM* [2]. This formalism provides a *functional* rather than an operational view on *Jazzyk* mental state transformers. The $yields$ calculus, introduced below, specifies an update associated with executing an *mst*. It formally defines the meaning of the state transformation induced by executing an *mst* in a state.

Formally, a mental state s of a *Jazzyk BSM* $(\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$ consists of the corresponding states $\langle D_1, \ldots, D_n \rangle$ of its KR modules. To specify the semantics of a *Jazzyk BSM*, first we need to define how queries are evaluated and how a state is modified by applying updates to it. A primitive query $\models \phi$ in a *Jazzyk BSM* state $s = \langle D_1, \ldots, D_n \rangle$ evaluates the formula $\phi \in \mathcal{L}_i$ using the query operator $\models \in \mathcal{Q}_i$ in the current state $D_i \subseteq \mathcal{L}_i$ of the corresponding KR module $\langle D_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i \rangle$. That is, $s \models_j (\models \phi)$ holds in a mental state s iff $D_i \models \phi$, otherwise we have $s \not\models_j (\models \phi)$. Given the usual meaning of Boolean operators, it is straightforward to extend the query evaluation to compound query formulae. Note that a query $\models \phi$ does not change the mental state s.

The semantics of a mental state transformer is a set of (possibly sequences of) *updates* (update set). The same notation $\oslash\phi$ is used to denote a simple update as well as the corresponding primitive *mst*. It should be clear from the context which of the two

is intended. Sequential application of updates is denoted by $\bullet$, i.e. $\rho_1 \bullet \rho_2$ is an update resulting from applying ρ_1 first and then applying ρ_2.

Definition 5 (Applying an Update). *The result of applying an update $\rho = \oslash\phi$ to a state $s = \langle D_1, \ldots, D_n\rangle$ of a BSM $\mathcal{B} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$, denoted by $s \bigoplus \rho$, is a new state $s' = \langle D_1, \ldots, D'_i, \ldots, D_n\rangle$ where $D'_i = D_i\rho = D_i \oslash \phi$ and D_i, $\oslash$, and ϕ correspond to one and the same $\mathcal{M}_i$ of $\mathcal{B}$. Applying the special update $\emptyset$ to a state s results in the same mental state $s = s \bigoplus \emptyset$.*

We write $D_i \bigoplus(\rho_1 \bullet \ldots \bullet \rho_k)$ for $(\ldots(D_i \bigoplus \rho_1) \bigoplus \ldots \bigoplus \rho_k)$ where all ρ_i correspond to D_i. The result of applying an update of the form $\rho_1 \bullet \rho_2$ to a state s, i.e. $s \bigoplus(\rho_1 \bullet \rho_2)$, is the new state $(s \bigoplus \rho_1) \bigoplus \rho_2$.

The meaning of a mental state transformer in state s, formally defined by the $yields$ predicate below, is the update it yields in that state. We introduce a version of the $yields$ calculus adapted from [9].

Definition 6 (Yields Calculus). *A mental state transformer τ yields an* update *ρ in a state s, iff $yields(\tau, s, \rho)$ is derivable in the following calculus:*

$$\frac{\top}{yields(\mathbf{skip}, s, \emptyset)} \qquad \frac{\top}{yields(\oslash\phi, s, \oslash\phi)} \qquad \textit{(yields of a primitive mst)}$$

$$\frac{yields(\tau, s, \rho),\ s \models_j \phi}{yields(\phi \longrightarrow \tau, s, \rho)} \qquad \frac{yields(\tau, s, \rho),\ s \not\models_j \phi}{yields(\phi \longrightarrow \tau, s, \emptyset)} \qquad \textit{(yields of a conditional mst)}$$

$$\frac{yields(\tau_1, s, \rho_1),\ yields(\tau_2, s, \rho_2)}{yields(\tau_1 | \tau_2, s, \rho_1)} \quad \frac{yields(\tau_1, s, \rho_1),\ yields(\tau_2, s, \rho_2)}{yields(\tau_1 | \tau_2, s, \rho_2)} \qquad \textit{(yields of a choice mst)}$$

$$\frac{yields(\tau_1, s, \rho_1),\ yields(\tau_2, s \bigoplus \rho_1, \rho_2)}{yields(\tau_1 \circ \tau_2, s, \rho_1 \bullet \rho_2)} \qquad \textit{(yields of a sequential mst)}$$

The *mst* $\mathbf{skip}$ yields the update $\emptyset$. Similarly, a primitive update *mst* yields the corresponding update. In case the condition of a conditional *mst* $\varphi \longrightarrow \tau$ is satisfied in the current mental state, the calculus yields one of the updates corresponding to the right hand side *mst* τ, otherwise the $\emptyset$ update is yielded. A non-deterministic choice *mst* yields an update corresponding to either of its members and finally a sequential *mst* yields a sequence of updates corresponding to the first *mst* of the sequence and an update yielded by the second member of the sequence in a state resulting from application of the first update to the current mental state.

4 Compiling a GOAL Agent into a Jazzyk BSM

In this Section we show that *GOAL* agents can be implemented as, or compiled into, *Jazzyk BSM*. The compiler is abstractly represented here by a function $\mathfrak{C}$ that translates (compiles) *GOAL* agents into *Jazzyk Behavioural State Machines*. The main result is a proof that for every *GOAL* agent $\mathcal{A} = \langle \Sigma, \Gamma, \Pi, \mathcal{A}\rangle$ there is a *Jazzyk BSM* $\mathfrak{C}(\mathcal{A}) = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$ that implements that *GOAL* agent. In fact, we will show that a *Jazzyk BSM* $\mathfrak{C}(\mathcal{A}) = (\mathcal{M}_\Sigma, \mathcal{M}_\Gamma, \tau)$ with precisely two KR modules is sufficient, where module $\mathcal{M}_\Sigma$ corresponds to the belief base Σ and module $\mathcal{M}_\Gamma$ corresponds to the goal base Γ. We proceed as follows. First, we define the KR modules $\mathcal{M}_\Sigma$ and $\mathcal{M}_\Gamma$

of the *Jazzyk BSM*, using the KR technology employed by *GOAL* agents as a starting point. Second, we show how to obtain a *Jazzyk BSM* agent program τ that implements the action rules in the program section Π and action specifications A of the *GOAL* agent. Finally, the equivalence of the *GOAL* agent with its *Jazzyk BSM* counterpart $\mathfrak{C}(\mathcal{A})$ is proven by showing that both are able to generate the same mental states.

Translation It is important to repeat that throughout this paper we have assumed that a *GOAL* agent uses a KR technology of the form $\mathcal{K}_0 = \langle \mathcal{L}, \{\models\}, \{\oplus, \ominus\}\rangle$ (see Section 2). Given this, it is straightforward to map a *GOAL* belief base onto a *Jazzyk BSM* KR module that is able to implement (i) the evaluation of a mental state condition $\mathbf{bel}(\phi)$ on a belief base as well as (ii) the execution of updates associated with performing an action. We simply map the *GOAL* belief base Σ onto the *Jazzyk BSM* module

$$\mathcal{M}_\Sigma = \mathfrak{C}_{\mathbf{bb}}(\Sigma) = \langle \Sigma, \mathcal{L}, \{\models\}, \{\oplus, \ominus\}\rangle \tag{1}$$

Whereas the underlying KR technology is implicitly assumed in a *GOAL* agent, this assumption is made explicit in the corresponding *Jazzyk BSM* KR module.

The translation of the goal base of a *GOAL* agent into a *Jazzyk BSM* module is less straightforward. A *Jazzyk BSM* module that implements the goal base needs to be able to implement (i) the evaluation of a mental state condition $\mathbf{goal}(\phi)$ on a goal base as well as (ii) the execution of updates on a goal base as a result of performing $\mathbf{adopt}$ or $\mathbf{drop}$ actions and the removal of goals that have been achieved. Because the $\mathbf{goal}$ operator has a somewhat non-standard semantics (see Definition 2), we need to define a non-standard KR technology associated with the *Jazzyk BSM* module implementing the goal base. Mapping a goal base Γ onto the module $\mathcal{M}_\Gamma$ provides what we need:

$$\mathcal{M}_\Gamma = \mathfrak{C}_{\mathbf{gb}}(\Gamma) = \langle \Gamma, \mathcal{L}, \{\models_{\mathbf{goal}}\}, \{\oplus_{\mathbf{adopt}}, \ominus_{\mathbf{drop}}, \ominus_{\mathbf{achieved}}\}\rangle \tag{2}$$

where:

- $\Gamma \models_{\mathbf{goal}} \phi$ iff there is a $\gamma \in \Gamma$ such that $\{\gamma\} \models \phi$.
- $\Gamma \oplus_{\mathbf{adopt}} \phi = \Gamma \cup \{\phi\}$.
- $\Gamma \ominus_{\mathbf{drop}} \phi = \Gamma \setminus \{\gamma \in \Gamma \mid \{\gamma\} \models \phi\}$.
- $\Gamma \ominus_{\mathbf{achieved}} \phi = \Gamma \setminus \{\phi\}$.

$\models_{\mathbf{goal}}$ is used to implement $\mathbf{goal}(\phi)$, $\oplus_{\mathbf{adopt}}$ implements $\mathbf{adopt}$, $\ominus_{\mathbf{drop}}$ is used to implement $\mathbf{drop}$, and finally $\ominus_{\mathbf{achieved}}$ implements the goal update mechanism to remove achieved goals. Note that the goal update mechanism of GOAL (cf. Definition 2) requires a simple set operator to remove a formula from the goal base such as $\ominus_{\mathbf{achieved}}$ and we cannot use $\ominus_{\mathbf{drop}}$ for this purpose.

Using the translations defined above it is now possible to translate mental state conditions ψ used in *GOAL* action rules of the form $\mathbf{if}\ \psi\ \mathbf{then}\ \mathtt{a}$. As noted above, $\mathfrak{C}(\mathbf{bel}(\phi))$ can be mapped onto the *Jazzyk BSM* query $\models \phi$; similarly, we can define $\mathfrak{C}(\mathbf{goal}(\phi)) = (\models_{\mathbf{goal}} \phi)$. Boolean combinations of mental state conditions are translated into Boolean combinations of *Jazzyk BSM* queries.

The translation of an action $\mathtt{a}$, the second part of an action rule of a *GOAL* agent, into *Jazzyk BSM mst*s is straightforward when $\mathtt{a}$ is either $\mathbf{adopt}$ or $\mathbf{drop}$ action. Since

both $\mathbf{adopt}(\phi)$ and $\mathbf{drop}(\phi)$ are always enabled, we can map these actions simply onto their corresponding primitive update operators:

$$\mathfrak{C}(\mathbf{adopt}(\phi)) = \oplus_{\mathbf{adopt}}\phi \quad (3)$$

$$\mathfrak{C}(\mathbf{drop}(\phi)) = \ominus_{\mathbf{drop}}\phi \quad (4)$$

The compilation of user defined actions, i.e. actions specified in the action specification section A, into *Jazzyk BSM* depends on the action specification A of the compiled *GOAL* agent. Such actions are mapped onto conditional *mst*s of the form $\varphi \longrightarrow \tau$. The preconditions of an action are mapped onto the query part φ of the *mst*; the effects of that action, expressed by a postcondition in *GOAL*, are translated into a sequential *mst* τ. Assuming that a is a *GOAL* action with the corresponding action specification a $\{$**:pre**$\{\phi\}$ **:post**$\{\phi_d \wedge \phi_a\}$, we define:

$$\mathfrak{C}(\mathtt{a}) = (\models \phi \longrightarrow \ominus\phi_d \circ \oplus\phi_a) \quad (5)$$

Note that the *Jazzyk BSM* operators $\models$, $\oplus$, and $\ominus$ are associated with the KR module $\mathcal{M}_\Sigma$ that implements the belief base of the *GOAL* agent, which ensures that the precondition ϕ is evaluated on the belief base of the agent and in line with Definition 3, the postcondition $\phi_d \wedge \phi_a$ is used to update that belief base.

Combining the translations of mental state conditions and actions yields a translation of action rules in the program section of a *GOAL* agent. It is also convenient to introduce a translation of a complete program section, i.e. a set Π of such rules. Note that the order of translation is unimportant.

$$\mathfrak{C}(\mathbf{if}\ \psi\ \mathbf{then}\ \mathtt{a}) = \mathfrak{C}(\psi) \longrightarrow \mathfrak{C}(\mathtt{a}) \quad (6)$$

$$\mathfrak{C}(\emptyset) = \mathbf{skip} \quad (7)$$

$$\mathfrak{C}(\Pi) = \mathfrak{C}(r) \,|\, \mathfrak{C}(\Pi \setminus \{r\})\text{ , if } r \in \Pi \quad (8)$$

The definitions above already allow us to define a compilation of a *GOAL* agent into a *Jazzyk BSM*, but it is convenient to first introduce the notion of a *possibly adopted goal*. A goal ϕ is said to be a *possibly adopted goal* whenever it is possible that the agent may come to adopt ϕ as a goal, i.e. whenever it is already present in the goal base or there is an action rule of the form $\mathbf{if}\ \psi\ \mathbf{then}\ \mathbf{adopt}(\phi)$ in Π. The set of possibly adopted goals $\mathcal{P}_\mathcal{A}$ of a *GOAL* agent $\mathcal{A} = \langle \Sigma, \Gamma, \Pi, A\rangle$ thus can be defined by $\mathcal{P}_\mathcal{A} = \Gamma \cup \{\phi \mid \mathbf{if}\ \psi\ \mathbf{then}\ \mathbf{adopt}(\phi) \in \Pi\}$. The notion introduced is useful since in the *Jazzyk BSM* translation we need to also implement the blind commitment strategy of *GOAL*, i.e. the removal of goals whenever these are completely achieved. A *Jazzyk BSM mst* that consists of a sequence of conditional *mst*s is introduced to implement the goal update mechanism of *GOAL*. Each of these corresponds to a single possibly adopted goal. The corresponding query evaluates whether $\phi \in \mathcal{P}_\mathcal{A}$ is (believed to be) achieved, whereupon ϕ is removed from the goal base:

$$\mathfrak{C}_{\mathbf{bcs}}(\emptyset) = \mathbf{skip} \quad (9)$$

$$\mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_\mathcal{A}) = (\models \phi \longrightarrow \ominus_{\mathbf{achieved}}\phi) \circ \mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_\mathcal{A} \setminus \{\phi\})\text{ , if } \phi \in \mathcal{P}_\mathcal{A} \quad (10)$$

```
:main: blocksWorld
{
  //*** Initializations omitted ***/
  :beliefs{. . .}
  :goals{. . .}

  :program{
    if bel(on_table([B|S]), clear(B),
           block(C), clear(C)) ,
       goal(on_table([C,B|S]))
    then move(C,B).
    if goal(on(B,A)),
       bel(on_table([C|S]),
           clear(C), member(B,S))
    then move(C,table).
  }

  :actionspec{
    move(X,Y) {
      :pre{ clear(X), clear(Y), on(X,Z), not(on(X,Y)) }
      :post{ not(on(X,Z)), on(X,Y) }
    }
  }
}
```

```
/*** Modules initialization omitted ***/
{ // ******** 𝔈(Π) ********
  when ⊨ [{on_table([B|S]), clear(B), block(C), clear(C)}]
    and ⊨goal [{on_table([C,B|S])}]
  then {
    when ⊨ [{clear(C), clear(B), on(C,Z), not(on(C,B))}]
    then ⊕ [{not(on(C,Z)), on(C,B)}]
  } ;
  when ⊨goal [{on(B,A)}] and
          ⊨ [{on_table([C|S]), clear(C), member(B,S)}]
  then {
  when ⊨ [{clear(C), clear(table),
           on(C,Z), not(on(C,table))}]
  then ⊕ [{not(on(C,Z)), on(C,table)}]
  }
} ,
{ // ******** 𝔈drop(Gl(A)) ********
  when ⊨ [{on(b,a), on(a,table)}]
    then ⊖goal [{on(b,a), on(a, table)}] ,
  when ⊨ [{on_table([a,b])}]
    then ⊖goal [{on_table([a,b])}] ,
  when ⊨ [{on_table([b])}]
    then ⊖goal [{on_table([b])}]
}
```

Fig. 1. Example of a translation of a simple *GOAL* agent moving blocks on a table into *Jazzyk BSM* pseudocode. **when** ... **then** ... encodes a conditional *mst*, ; and , stand for | and ∘ respectively.

The compilation of a *GOAL* agent $\langle \Sigma, \Gamma, \Pi, A\rangle$ into a *Jazzyk BSM* is defined as:

$$\mathfrak{C}(\langle \Sigma, \Gamma, \Pi, A\rangle) = (\mathcal{M}_\Sigma, \mathcal{M}_\Gamma, \mathfrak{C}(\Pi) \circ \mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_\mathcal{A})) \tag{11}$$

Correctness of the Translation Function $\mathfrak{C}$ The main effort in proving that the compilation of a *GOAL* agent $\mathcal{A} = \langle \Sigma, \Gamma, \Pi, A\rangle$ into a *Jazzyk BSM* $\mathfrak{C}(\mathcal{A}) = (\mathcal{M}_\Sigma, \mathcal{M}_\Gamma, \mathfrak{C}(\Pi) \circ \mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_\mathcal{A}))$ is correct consists of showing that the action rules Π of the *GOAL* agent generate the same mental states as the mental state transformer $\mathfrak{C}(\Pi) \circ \mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_\mathcal{A})$. In order to prove this we first prove some useful properties of $\mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_\mathcal{A})$ that implements the goal update mechanism of *GOAL* (Lemma 1), the relation of *GOAL* mental states resulting from action execution to the application of updates to *Jazzyk BSM* mental states (Lemma 2), and the evaluation of mental state conditions in *GOAL* to the evaluation of their translations in *Jazzyk* (Lemma 3). Due to space limitations we omit the detailed proofs for these lemmas.

Lemma 1 shows that a *Jazzyk BSM* state, which does not need to be a *GOAL* state, nevertheless is a *GOAL* mental state after removing goals that are believed to be achieved, and that the mst $\mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_\mathcal{A})$ implements this goal update mechanism.

Lemma 1. *Let $m = \langle \Sigma, \Gamma\rangle$ be a* Jazzyk BSM *state such that $\Sigma \not\models \bot$ and $\Gamma \subseteq \mathcal{P}_\mathcal{A}$, and ρ be an update $\ominus_{\mathbf{achieved}}\gamma_1 \bullet \ldots \bullet \ominus_{\mathbf{achieved}}\gamma_n$. Then $yields(\mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_\mathcal{A}), m, \rho)$ iff*

(i) $\langle \Sigma, \Gamma \bigoplus \rho\rangle$ is a GOAL *mental state, and*
(ii) there is no Γ': $\Gamma \bigoplus \rho \subset \Gamma' \subseteq \Gamma$ such that $\langle \Sigma, \Gamma'\rangle$ is a GOAL *mental state.*

Lemma 2 proves that the *GOAL* states resulting from executing an action can also be obtained by applying updates of a particular structure, which is useful to relate *GOAL*

actions to *Jazzyk BSM* updates. The fact that the *Jazzyk BSM* mst τ that is the *Jazzyk BSM* translation of a *GOAL* agent also yields updates with the same structure is useful to relate *Jazzyk BSM* updates to *GOAL* actions again.

Lemma 2. *Let* $\mathcal{A} = \langle \Sigma, \Gamma, \Pi, A \rangle$ *be a* GOAL *agent and* $\mathfrak{C}(\mathcal{A}) = (\mathcal{M}_\Sigma, \mathcal{M}_\Gamma, \tau)$ *its* Jazzyk BSM *compilation. Also let* $\mathtt{a}$ *be a user defined action of* GOAL *agent* $\mathcal{A}$*, with action specification* $\mathtt{a}$ {**:pre**$\{\phi\}$ **:post**$\{\phi_a \wedge \phi_d\}$}*. Then*

(i) $m \xrightarrow{\mathtt{a}} m'$ *iff* $\exists n \geq 0 : m' = m \bigoplus (\ominus\phi_d \bullet \oplus\phi_a \bullet \ominus_{\mathbf{achieved}}\gamma_1 \bullet \ldots \bullet \ominus_{\mathbf{achieved}}\gamma_n)$.

(ii) $m \xrightarrow{\mathbf{drop}(\phi)} m'$ *iff* $m' = m \bigoplus (\ominus_{\mathbf{drop}}\phi)$.

(iii) $m \xrightarrow{\mathbf{adopt}(\phi)} m'$ *iff* $m' = m \bigoplus (\oplus_{\mathbf{adopt}}\phi)$.

(iv) If $yields(\tau, m, \rho)$*, then* ρ *is of the form* $\ominus\phi_d \bullet \oplus\phi_a \bullet \ominus_{\mathbf{achieved}}\gamma_1 \bullet \ldots \bullet \ominus_{\mathbf{achieved}}\gamma_n$ *for some* $n \geq 0$*, or of the form* $\ominus_{\mathbf{drop}}\phi$ *or* $\ominus_{\mathbf{adopt}}\phi$*.*

Lemma 3 relates the evaluation of *GOAL* mental state conditions to the evaluation of their *Jazzyk BSM* translation in the same state.

Lemma 3. *Let* ψ *be a mental state condition. It holds that*

$$m \models_g \psi \text{ iff } m \models_j \mathfrak{C}(\psi)$$

Finally, Theorem 1 shows that the updates generated by the *Jazzyk* translation of a *GOAL* agent produce the same mental states as the execution of actions by that *GOAL* agent, which shows that the *Jazzyk BSM* implements the *GOAL* agent.

Theorem 1 (Correctness of GOAL-2-BSM Compilation). *Let* $\mathcal{A} = \langle \Sigma, \Gamma, \Pi, A \rangle$ *be a* GOAL *agent with mental state* $m = \langle \Sigma, \Gamma \rangle$ *and* $\mathfrak{C}(\mathcal{A}) = (\mathcal{M}_\Sigma, \mathcal{M}_\Gamma, \tau)$ *its corresponding* Jazzyk BSM *translation. Then for all* ρ*:*

$$\exists \mathtt{a} : m \xrightarrow{\mathtt{a}} m \bigoplus \rho \quad \textit{iff} \quad yields(\tau, m, \rho).$$

Proof. Informally, to show the left to right direction ($\Longrightarrow$), we have to show that if a *GOAL* action $\mathtt{a}$ is enabled in a mental state m, there exists an update ρ such that (a) the state resulting from performing $\mathtt{a}$ is $m \bigoplus \rho$ and (b) ρ is yielded by τ in this state. Note that even though an update operator ρ occurs on the left hand side the expression on the left hand side denotes a *GOAL* transition. From Lemma 2 we know that such a ρ exists and is of the form (i) $\rho = \ominus\phi_d \bullet \oplus\phi_a \bullet \ominus_{\mathbf{achieved}}\gamma_1 \bullet \ldots \bullet \ominus_{\mathbf{achieved}}\gamma_n$ for user specified actions $\mathtt{a}$, (ii) $\rho = \ominus_{\mathbf{drop}}\phi$ if $\mathtt{a} = \mathbf{drop}(\phi)$ and (iii) $\rho = \oplus_{\mathbf{adopt}}\phi$ if $\mathtt{a} = \mathbf{adopt}(\phi)$.

So suppose that $m \xrightarrow{\mathtt{a}} m \bigoplus \rho$ and $\mathtt{a}$ is a user defined action (the other cases dealing with $\mathtt{a} = \mathbf{drop}(\phi)$ and $\mathtt{a} = \mathbf{adopt}(\phi)$ are similar). This means there is an action rule **if** ψ **then** $\mathtt{a}$, and precondition ϕ and postcondition $\phi_d \wedge \phi_a$ associated with action $\mathtt{a}$ such that $m \models_g \psi$ and $\Sigma \models \phi$. It remains to show that update ρ is also yielded by τ. By construction, we must have that

$$\tau = (...|(\mathfrak{C}(\psi) \longrightarrow (\models \phi \longrightarrow \ominus\phi_d \circ \oplus\phi_a))|...) \circ \mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_\mathcal{A})$$

Since we have $m \models_g \psi$ and $\Sigma \models \phi$, using Lemma 3 it is immediate that we have $yields(\mathfrak{C}(\psi) \longrightarrow (\models \phi \longrightarrow \ominus\phi_d \circ \oplus\phi_a), m, \ominus\phi_d \bullet \oplus\phi_a)$. Finally, from Lemma 1, we

have that $yields(\mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_{\mathcal{A}}), m \bigoplus(\ominus\phi_d \bullet \oplus\phi_a), \{\ominus_{\mathbf{achieved}}\gamma_1 \bullet \ldots \bullet \ominus_{\mathbf{achieved}}\gamma_n)$ and by applying sequential composition on the resulting updates we are done.

($\Longleftarrow$) In the other direction, we have to prove that the updates performed by $\mathfrak{C}(\mathcal{A})$ correspond to enabled actions of the *GOAL* agent $\mathcal{A}$. So suppose that $yields(\tau, m, \rho)$, and ρ is of the form $\ominus\phi_d \bullet \oplus\phi_a \bullet \ominus_{\mathbf{achieved}}\gamma_1 \bullet \ldots \bullet \ominus_{\mathbf{achieved}}\gamma_n$ (using Lemma 2(iv); the other cases with $\rho = \ominus_{\mathbf{drop}}\phi$ and $\rho = \oplus_{\mathbf{adopt}}\phi$ are again similar). From the construction of $\mathfrak{C}$ it follows that we must have $yields(\mathfrak{C}(\psi) \longrightarrow (\models \phi \longrightarrow \ominus\phi_d \circ \oplus\phi_a) \circ \mathfrak{C}_{\mathbf{bcs}}(\mathcal{P}_{\mathcal{A}}), m, \rho)$. From the rule for conditional *mst* in the yields calculus (Definition 6) follows that $m \models_j \mathfrak{C}(\psi)$ and $m \models_j (\models \phi)$. By Lemma 3 we then have $m \models_g \psi$ and $\Sigma \models \phi$. We must also have an action rule $\mathbf{if}\ \psi\ \mathbf{then}\ \mathtt{a}$ with action specification $\mathtt{a}$ $\{$**:pre**$\{\phi\}$ **:post**$\{\phi_a \wedge \phi_d\}$ such that $m \xrightarrow{\mathtt{a}} m \bigoplus(\ominus\phi_d \bullet \oplus\phi_a \bullet \ominus_{\mathbf{achieved}}\gamma'_1 \bullet \ldots \bullet \ominus_{\mathbf{achieved}}\gamma'_m)$ (cf. Lemma 2(i)). It remains to be shown that $\ominus_{\mathbf{achieved}}\gamma_1 \bullet \ldots \bullet \ominus_{\mathbf{achieved}}\gamma_n$ is equal to $\ominus_{\mathbf{achieved}}\gamma'_1 \bullet \ldots \bullet \ominus_{\mathbf{achieved}}\gamma'_m$; this follows immediately from Lemma 1.

5 Discussion & Conclusion

We showed that any *GOAL* agent can be compiled into a *Jazzyk Behavioural State Machine*. More precisely, it was shown that every possible computation step of a *GOAL* agent can be emulated by the *Jazzyk BSM* that is the result of compiling the *GOAL* agent into *Jazzyk BSM*. The compilation procedure is *compositional* in the sense that any modifications or extensions of the belief base, goal base or program and action specification sections of the *GOAL* agent only *locally* affect, respectively, the compiled belief base module, the compiled goal base module, or the mental state transformer that is the result of compiling the program and action specification sections.

The compilation function introduced provides a means to translate *GOAL* agents into *Jazzyk BSM*, but not vice versa. Abstracting from a number of details a *Jazzyk BSM* could be viewed as a *GOAL* agent that does not use its goal base and associated goal update mechanism. As mentioned above, *Jazzyk* does not commit to any particular view on the KR modules of a *Jazzyk BSM*. This flexibility allowed us to implement the goal base of a *GOAL* agent by means of explicit emulation of the goal update mechanism.

As already noted in the introduction, there is not much related work aimed at providing an effective strategy or tools for implementing a variety of rule-based agent programming languages such as those described in [1]. To the best of our knowledge, only [6] has presented a framework to this end. The resulting framework, however, is based on the idea to incorporate each and every semantic feature of a variety of available high-level agent languages in order to be able to cover every type of agent. It thus does not provide an implementation strategy as the one promoted and illustrated in this paper, which is based on the idea to provide a concise set of simple high-level concepts (a common core) facilitating compilation of a variety of agent programs into this core instruction set. This strategy is explicitly aimed at *reducing* a set of high-level agent programming concepts to a *simpler*, more basic set of concepts.

The implementation strategy used to identify specific semantic features of the *GOAL* language and to emulate these explicitly in *Jazzyk* also raises the question whether features of other agent programming languages can be compiled in a similar way. Although

we do not have room to extensively argue for this, we believe that a similar approach can also be applied to other rule-based agent programming languages. In particular, the following implementation strategy could be applied to compile agent programs into *Jazzyk BSM*: (i) compile the underlying knowledge base(s) into equivalent *Jazzyk BSM* KR module(s), (ii) compile the (action, planning, ...) rules of the agent program into *Jazzyk BSM* mental state transformers using the operators of the KR module(s), and finally (iii) implement any specific semantic features of the language by a *Jazzyk BSM mst* and "append" it to the one constructed in the previous step. Moreover, since *Jazzyk BSM* also features a much simpler conceptual scheme than higher level agent languages, we believe that it provides a promising basis for an intermediate language into which agent programs can be compiled and interpreted.

Our result shows that *GOAL* does not commit to any particular KR technology such as Prolog. Another issue that remains is whether it would be possible to allow *GOAL* agents to use multiple KR technologies. The compilation into *Jazzyk BSM* provides some evidence that this is possible since *Jazzyk BSM* enables the use of many different KR technologies. However, the use of multiple KR technologies within a single agent will add expressive power only when certain key issues related to the "interoperability" of different KRs have been solved (for a discussion see also [3]).

References

1. R.H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming Languages, Platforms and Applications*. Kluwer, 2005.
2. E. Börger and R.F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.
3. Mehdi Dastani, Koen V. Hindriks, Peter Novák, and Nick A.M. Tinnemeier. Combining multiple knowledge representation technologies into agent programming languages. In *Proc. of the Intl. Workshop on Declarative Agent Languages and Technologies, (DALT'08)*, 2008.
4. R. Davis, H.E. Shrobe, and P. Szolovits. What Is a Knowledge Representation? *AI*, 14(1):17–33, 1993.
5. F. de Boer, K. Hindriks, W. van der Hoek, and J.-J.Ch. Meyer. A Verification Framework for Agent Programming with Declarative Goals. *Journal of Applied Logic*, 5(2):277–302, 2007.
6. L.A. Dennis, R.H. Bordini, B. Farwer, M. Fisher, and M. Wooldridge. A common semantic basis for BDI languages. In *Proceedings of the International Workshop on Programming Multi-Agent Systems (ProMAS'07)*, LNAI 4908. Springer, 2008.
7. K. Hindriks. Modules as Policy-Based Intentions. In *Proceedings of the International Workshop on Programming Multi-Agent Systems (ProMAS'07)*, LNAI 4908. Springer, 2008.
8. P. Novák. Behavioural State Machines: programming modular agents. In *AAAI 2008 Spring Symposium: Architectures for Intelligent Theory-Based Agents (AITA'08)*, 2008.
9. Peter Novák. Jazzyk: A programming language for hybrid agents with heterogeneous knowledge representations. In *Proc. of the 6th Intl. Workshop on Programming Multi-Agent Systems, (ProMAS'08)*, 2008.
10. E. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proc. of the Int. Conf. on Principles of Knowledge Representation and Reasoning*, 1989.
11. Gordon D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.
12. Leon Sterling and Ehud Shapiro. *The Art of Prolog*. MIT press, 1986.
13. David H. D. Warren. An Abstract Prolog Instruction Set. Technical Report 309, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, 1983.