I. Introduction

We develop within Higher Order Logic (HOL) a General and
Abstract and Reusable Framework for Abstracting
XP, Depart. of Computer Science, The University of Sheffield,
M. Fairtlough, A. M. Mendler, and X. Cheng

Abstraction and Reuse
The process of abstraction is identical to the process of refinement.

Abstraction in Artificial Intelligence: Abstraction techniques have been extensively used in the design and implementation of problem-solving systems.
to find such a set of decision trees, or ensure that a predefined function or policy is not applicable. The approach can be refined effectively by a deep understanding of the core model of behavior and the associated techniques for applying it. In this section, we develop a framework for applying the approach, which we term the 'abstraction refinement' process. A key aspect of the abstraction refinement process is the identification of 'sub-modules'.

We define two main approaches to the problem of 'abstraction refinement'.

1. The first approach is to identify 'sub-modules'. One way to identify 'sub-modules' is to look at the interactions between different components of the model. This approach is useful in identifying the components that are most relevant to the problem at hand. It can be applied to a wide range of problems, including those in software engineering, where it is important to identify the components that are most relevant to the problem at hand.

2. The second approach is to identify 'sub-modules' based on the interactions between different components of the model. This approach is useful in identifying the components that are most relevant to the problem at hand. It can be applied to a wide range of problems, including those in software engineering, where it is important to identify the components that are most relevant to the problem at hand.
Abstract properties of the system under consideration. It turns out that if
the parameter is in the model then the condition holds, where

\[ \phi \models \psi \],

and any expression of \( \phi \) t.e. the parameter \( \psi \) is a condition.

The system is defined in a model of HOL. Intuitively, the occurrence of
HOL abstraction in a model is defined within HOL and the model
expression needed in these expressions may be defined within
\( \lambda \text{ def} \) as the condition

\[ (((\phi f) g) h) \lambda A \models (((\phi f) g) h) \lambda A = A \]

Each new expression is of the form \( \lambda A \), where

\[ (\phi f) g \lambda A : \text{def} \cdot \phi \]

process to these formulas are then

The roles of the application

We observe that the lambda

\[ (\phi f) g \lambda A = A \doteq_0 \]

where

\[ (\phi f) g \lambda A : \text{def} \cdot \phi \]

formulas are then

\[ (\phi f) g \lambda A : \text{def} \cdot \phi \]

process to these formulas are then

The roles of the application

We observe that the lambda

\[ (\phi f) g \lambda A = A \doteq_0 \]

where

\[ (\phi f) g \lambda A : \text{def} \cdot \phi \]

formulas are then

The roles of the application

We observe that the lambda

\[ (\phi f) g \lambda A = A \doteq_0 \]

where

\[ (\phi f) g \lambda A : \text{def} \cdot \phi \]

formulas are then

The roles of the application

We observe that the lambda

\[ (\phi f) g \lambda A = A \doteq_0 \]

where

\[ (\phi f) g \lambda A : \text{def} \cdot \phi \]
2 Higher-order framework: technical details

This section is intended to be a more detailed discussion of the higher-order framework introduced in the previous section. We assume a familiarity with basic concepts of higher-order logic and abstract syntax.

The key idea is to define a higher-order framework that integrates the higher-order logic with the higher-order framework.

\[ \forall \alpha \in \alpha \cdot \exists \phi : \alpha \to \phi \]

The main result is that the definition of \( (\alpha \to \phi) \), which combines higher-order logic with the higher-order framework, gives rise to a higher-order framework that integrates the higher-order logic with the higher-order framework.

\[ \forall \alpha \in \alpha \cdot \exists \phi : \alpha \to \phi \]

For example, the definition of \( (\alpha \to \phi) \) includes the higher-order logic and the higher-order framework, allowing for a seamless integration of the two.

\[ \forall \alpha \in \alpha \cdot \exists \phi : \alpha \to \phi \]

This is achieved through a higher-order framework that combines higher-order logic with the higher-order framework, allowing for a seamless integration of the two.

\[ \forall \alpha \in \alpha \cdot \exists \phi : \alpha \to \phi \]

We assume a familiarity with basic concepts of higher-order logic and abstract syntax.

The key idea is to define a higher-order framework that integrates the higher-order logic with the higher-order framework.

\[ \forall \alpha \in \alpha \cdot \exists \phi : \alpha \to \phi \]

The main result is that the definition of \( (\alpha \to \phi) \), which combines higher-order logic with the higher-order framework, gives rise to a higher-order framework that integrates the higher-order logic with the higher-order framework.

\[ \forall \alpha \in \alpha \cdot \exists \phi : \alpha \to \phi \]
In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001


In this paper, we study the concept of abstraction and refinement in higher-order logic. We introduce a framework for developing a theory of abstraction and refinement that is based on the idea of abstraction, where a higher-level (abstract) formula is a consequence of a lower-level (concrete) formula.

<table>
<thead>
<tr>
<th>[\forall x \in A] \iff [\forall x \in A^\circ]</th>
<th>\neg [\forall x \in A] \iff \neg [\forall x \in A^\circ]</th>
<th>[\forall x \in A] \iff [\forall x \in A^\circ]</th>
<th>[\forall x \in A] \iff [\forall x \in A^\circ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\forall x \in A] \iff [\forall x \in A^\circ]) \iff [\forall x \in A^\circ]</td>
<td>([\forall x \in A] \iff [\forall x \in A^\circ]) \iff [\forall x \in A^\circ]</td>
<td>([\forall x \in A] \iff [\forall x \in A^\circ]) \iff [\forall x \in A^\circ]</td>
<td>([\forall x \in A] \iff [\forall x \in A^\circ]) \iff [\forall x \in A^\circ]</td>
</tr>
</tbody>
</table>

In order to define when a part of \( \forall x \in A \) is properly formed, we have a function \( \forall x \in A \subseteq \forall x \in A^\circ \) for each order of formula, which is a consequence of abstraction and refinement. We define a function \( \forall x \in A \subseteq \forall x \in A^\circ \) for each order of formula, which is a consequence of abstraction and refinement. We define a function \( \forall x \in A \subseteq \forall x \in A^\circ \) for each order of formula, which is a consequence of abstraction and refinement.
The process of generating the proof is done in a constructive manner and on abstract formulae. The process of refinement from higher order to lower order can be seen to completely collapse together to produce a proof. Theorem 1. (Theorem 1)
For the definition of $\nu_e$ as an abstraction in HOL.

The equational calculus used in the equational proofs in Sections 8 and 9 are intended to be special cases of those equational calculi for HOL, which are special to our complete proof calculus. The proof calculus for HOL provides a computational semantics for the proof terms constructed such that the equational proofs then can be generated from the equational full abstraction theorems that prove the consistency of the quotient term syntax. These proofs of the consistency theorems are based on the fundamental consistency arguments by equational proof methods.

**Fig. 6. Natural Deduction Rules for Abstraction Logic.**

\[
\begin{align*}
\text{\textit{Mutual Induction Rule}}: & \quad \vdash x \to (y \to z) \to \phi \\
\text{\textit{Existential Introduction}}: & \quad \vdash \exists x : \phi \\
\text{\textit{Universal Elimination}}: & \quad \vdash \phi \to \psi \\
\text{\textit{Universal Introduction}}: & \quad \vdash \exists x : \phi \\
\text{\textit{Existential Elimination}}: & \quad \vdash \phi \to \psi \\
\text{\textit{Universal Elimination}}: & \quad \vdash \exists x : \phi \\
\end{align*}
\]

**Fig. 7. Natural Deduction Rules for Abstraction Logic.**

\[
\begin{align*}
\text{\textit{Mutual Induction Rule}}: & \quad \vdash x \to (y \to z) \to \phi \\
\text{\textit{Existential Introduction}}: & \quad \vdash \exists x : \phi \\
\text{\textit{Universal Elimination}}: & \quad \vdash \phi \to \psi \\
\text{\textit{Universal Introduction}}: & \quad \vdash \exists x : \phi \\
\text{\textit{Existential Elimination}}: & \quad \vdash \phi \to \psi \\
\text{\textit{Universal Elimination}}: & \quad \vdash \exists x : \phi \\
\end{align*}
\]
The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]

The key difference shown in the theorem prover is a theorem proving in higher-order logic. The theorem proving in higher-order logic can be described by
\[ I + (1 + 1) \]
and
\[ I \]
We refer to the results of transformations in fundamental mode; thereby the function that

\[ f: \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \]

remains a bijection. This is because the function \( f \) is a bijection, and therefore the operation \( f \circ \circ \) is a bijection. In particular, this guarantees that the bijection is a bijection. Consequently, the bijection is a bijection. Hence, the function that remains a bijection is a bijection.
Theorem-proving for Higher-Order Logic

II

In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

...
\[\begin{align*}
&\forall x \in \mathbb{N} \cup \{0\}, \exists y \in \mathbb{N} : x = y + 1 \\
&\forall x, y \in \mathbb{N}, \exists z \in \mathbb{N} : x + y = z \\
&\forall x, y, z \in \mathbb{N}, \exists t \in \mathbb{N} : x + y + z = t \\
&\forall x, y, z \in \mathbb{N}, \exists t \in \mathbb{N} : x + y + z = t \\
&\forall x, y, z \in \mathbb{N}, \exists t \in \mathbb{N} : x + y + z = t
\end{align*}\]
We have defined the proof term \( \phi \) for the induction step. Now we can take

\[
(1) \quad (\forall \sigma \cdot \exists \xi \cdot \left( (\mathcal{A} + 1) \left( \mathcal{B} + (1) \right) \right) = \xi \quad \forall \xi \quad \forall \xi
\]

\[
\exists \xi \quad \left( (\mathcal{A} + 1) \left( \mathcal{B} + (1) \right) \right) = \xi \quad \forall \xi \quad \forall \xi
\]

\[
(2) \quad \mathcal{A} + \mathcal{B} + (1) \left( \mathcal{C} + (1) \right) = \xi \quad \forall \xi \quad \forall \xi
\]

\[
\exists \xi \quad (\mathcal{A} + 1) \left( \mathcal{B} + (1) \right) = \xi \quad \forall \xi \quad \forall \xi
\]

\[
(3) \quad \left( \mathcal{A} + (1) \right) = \xi \quad \forall \xi \quad \forall \xi
\]

\[
\exists \xi \quad (\mathcal{A} + 1) = \xi \quad \forall \xi \quad \forall \xi
\]
M. Fairtlough, M. Mendler, X. Cheng: Abstraction and refinement in higher-order logic.

In R. J. Boulton, P. B. Jackson (eds), Theorem-proving in Higher-order Logic (TPHOLs'2001), pp. 201-216, Springer LNCS 2152, 2001

PRELIMINARY VERSION of a paper under copyright with Springer Verlag
\[ a \to b \quad \equiv \quad (a \land b) \lor (a \land \neg b) \lor (\neg a \land b) \lor (\neg a \land \neg b) \]

\[ (a \lor b) \land (c \lor d) \quad \equiv \quad (a \land c) \lor (a \land d) \lor (b \land c) \lor (b \land d) \]

We are currently evaluating an implementation of our method in the PROBE system.

We have presented a constructive extension to higher order logic reasoning.

4 Conclusions

We describe a version of constructive extension of HOL, where we define the extension of the Hoare logic. The extension is very straightforward in, e.g., we define the extension of the Hoare logic to be:

\[ \begin{align*}
\text{hoare} & : d \to d' \iff d, d' \in \text{HOL} \\
\text{hoareprove} & : (\exists d. d' \text{ and } d, d' \text{ Hoare-provable}) \\
\text{hoareprove} & : (\exists d. d' \text{ and } d, d' \text{ Hoare-provable})
\end{align*} \]
References

design, and implementation to expose the application to formal verification. We are now applying our methods to the older domain of thermodynamics and improve the timing constraints for the logic and the implementation example now be simply defined as $P$. We have used one implementation in practice.
Abstraction and refinement in higher-order logic.