A Verified Cyclicity Checker
For Theories with Overloaded Constants

Arve Gengelbach [1] Johannes Åman Pohjola [2]

[1]KTH Royal Institute of Technology, Stockholm, Sweden

[2]University of New South Wales, Sydney, Australia

ITP, August 7, 2022

In classical and intuitionistic logics, any permitted definitions, like

$$\mathsf{c}_{\mathsf{bool}} \equiv \mathit{term}$$

should be free from contradiction, i. e. *consistent*.

In classical and intuitionistic logics, any permitted definitions, like

$$\mathsf{c}_{\mathsf{bool}} \equiv \mathit{term}$$

should be free from contradiction, i. e. *consistent*.

Suffices: $\mathsf{c}_{\mathsf{bool}}$ is a new symbol, not contained in *term*.

In classical and intuitionistic logics, any permitted definitions, like

$$\mathsf{c}_{\mathsf{bool}} \equiv \neg \mathsf{c}_{\mathsf{bool}}$$

should be free from contradiction, i. e. *consistent*.

Suffices: $\mathsf{c}_{\mathsf{bool}}$ is a new symbol, not contained in $\neg \mathsf{c}_{\mathsf{bool}}$.

In classical and intuitionistic logics, any permitted definitions, like

$$\mathsf{c}_{\mathsf{bool}} \equiv \textit{term}$$
$$\mathsf{c}_{\alpha\ \mathsf{list}} \equiv \dots$$

should be free from contradiction, i. e. *consistent*.

In classical and intuitionistic logics, any permitted definitions, like

$$\mathsf{c}_{\mathsf{bool}} \equiv \mathit{term}$$
$$\mathsf{c}_{\alpha\ \mathsf{list}} \equiv \ldots$$

should be free from contradiction, i. e. *consistent*.

How can we detect whether *overloaded* definitions are without contradiction?

In classical and intuitionistic logics, any permitted definitions, like

$$\mathsf{c}_{\mathsf{bool}} \equiv \textit{term}$$
$$\mathsf{c}_{\alpha\ \mathsf{list}} \equiv \dots$$

should be free from contradiction, i. e. *consistent*.

How can we detect in a verified manner whether *overloaded* definitions are without contradiction?

In classical and intuitionistic logics, any permitted definitions, like

$$c_{\mathsf{bool}} \equiv \mathit{term}$$
$$c_{\alpha\ \mathsf{list}} \equiv \dots$$

should be free from contradiction, i. e. *consistent*.

How can we detect in a verified manner whether *overloaded* definitions are without contradiction?

Why? Soundness critical (not well understood) part of theorem provers; mistakes in earlier attempts

Contribution

- Formally proved theory:
  there is structure in overloaded definitions.
- Verified checker
- Demonstrate checker on Isabelle/HOL basis
- Verified kernel for HOL with overloading

Higher-order logic (HOL)

- Rank-1 polymorphic lambda-calculus
  with built-in types $\quad \mathsf{bool}, \alpha \to \beta, \mathsf{ind}$
  and built-in constants $=_{\alpha \to \alpha \to \mathsf{bool}}$, $\epsilon_{(\alpha \to \mathsf{bool}) \to \alpha}$
- Theorems: boolean terms derived by inference rules

Higher-order logic (HOL) with definitions

- Rank-1 polymorphic lambda-calculus
  with built-in types $\mathsf{bool}, \alpha \to \beta, \mathsf{ind}$
  and built-in constants $=_{\alpha\to\alpha\to\mathsf{bool}}$, $\epsilon_{(\alpha\to\mathsf{bool})\to\alpha}$
- Theorems: boolean terms derived by inference rules
- Consistency by definitional (non axiomatic) theory extension,[1] assuming...
  - New types are isomorphic to non-empty subsets of existing type
  - New constants abbreviate existing terms

[1] Formal consistency proof by Kumar et al. ITP '14

HOL with overloaded definitions

- Overloading allows constants different definitions at non-overlapping types, e. g. $\mathsf{size}_{\alpha \to \mathsf{num}}$

$$\mathsf{size}(x_{\alpha\,\mathsf{list}}) \equiv \mathsf{length}\, x$$
$$\mathsf{size}(x_{\mathsf{bool}}) \equiv 1$$

HOL WITH OVERLOADED DEFINITIONS

- Overloading allows constants different definitions at non-overlapping types, e. g. $\mathsf{size}_{\alpha \to \mathsf{num}}$

$$\mathsf{size}(x_{\alpha\,\mathsf{list}}) \equiv \mathsf{length}\, x$$
$$\mathsf{size}(x_{\mathsf{bool}}) \equiv 1$$

- Each defined symbol *depends* on symbols in definition's rhs, e. g. $\mathsf{size}_{\alpha\,\mathsf{list} \to \mathsf{num}} \rightsquigarrow \mathsf{length}_{\alpha\,\mathsf{list} \to \mathsf{num}}$, $\mathsf{size}_{\alpha\,\mathsf{list} \to \mathsf{num}} \rightsquigarrow \alpha\,\mathsf{list}$

HOL WITH OVERLOADED DEFINITIONS

- Overloading allows constants different definitions at non-overlapping types, e. g. $\mathsf{size}_{\alpha \to \mathsf{num}}$

$$\mathsf{size}(x_{\alpha\,\mathsf{list}}) \equiv \mathsf{length}\,x$$
$$\mathsf{size}(x_{\mathsf{bool}}) \equiv 1$$

- Each defined symbol *depends* on symbols in definition's rhs, e. g. $\mathsf{size}_{\alpha\,\mathsf{list} \to \mathsf{num}} \rightsquigarrow \mathsf{length}_{\alpha\,\mathsf{list} \to \mathsf{num}}$, $\mathsf{size}_{\alpha\,\mathsf{list} \to \mathsf{num}} \rightsquigarrow \alpha\,\mathsf{list}$
- $\rightsquigarrow^{\downarrow}$ extends $\rightsquigarrow$ to all type-instances: e. g. $\mathsf{size}_{\mathsf{num}\,\mathsf{list} \to \mathsf{num}} \rightsquigarrow^{\downarrow} \mathsf{length}_{\mathsf{num}\,\mathsf{list} \to \mathsf{num}}, \ldots$

HOL with overloaded definitions

- Overloading allows constants different definitions at non-overlapping types, e. g. $\mathsf{size}_{\alpha \to \mathsf{num}}$

$$\mathsf{size}(x_{\alpha\,\mathsf{list}}) \equiv \mathsf{length}\, x$$
$$\mathsf{size}(x_{\mathsf{bool}}) \equiv 1$$

- Each defined symbol *depends* on symbols in definition's rhs, e. g. $\mathsf{size}_{\alpha\,\mathsf{list} \to \mathsf{num}} \rightsquigarrow \mathsf{length}_{\alpha\,\mathsf{list} \to \mathsf{num}}$, $\mathsf{size}_{\alpha\,\mathsf{list} \to \mathsf{num}} \rightsquigarrow \alpha\,\mathsf{list}$
- $\rightsquigarrow^{\downarrow}$ extends $\rightsquigarrow$ to all type-instances: e. g. $\mathsf{size}_{\mathsf{num}\,\mathsf{list} \to \mathsf{num}} \rightsquigarrow^{\downarrow} \mathsf{length}_{\mathsf{num}\,\mathsf{list} \to \mathsf{num}}$, ...
- *Terminating dependencies* = no infinite descending chains in $\rightsquigarrow^{\downarrow *}$

Example [2]: Inconsistency from non-terminating dependencies

A theory with three constants $c_{\alpha\ \mathsf{list} \to \mathsf{bool}}$, $d_{(\alpha \times \beta) \to \mathsf{bool}}$, undef_α and two definitions:

$$c\,(x_{\alpha\ \mathsf{list}}) \equiv d(\mathsf{undef}_{\alpha \times \alpha}) \qquad d\,(x_{\alpha \times \mathsf{num}}) \equiv \neg c(\mathsf{undef}_{\alpha\ \mathsf{list}})$$

[2] Adapted from Kunčar CPP '15

Example [2]: Inconsistency from non-terminating dependencies

A theory with three constants $c_{\alpha\ \mathsf{list} \to \mathsf{bool}}$, $d_{(\alpha \times \beta) \to \mathsf{bool}}$, undef_α and two definitions:

$$c\,(x_{\alpha\ \mathsf{list}}) \equiv d(\mathsf{undef}_{\alpha \times \alpha}) \qquad d\,(x_{\alpha \times \mathsf{num}}) \equiv \neg c(\mathsf{undef}_{\alpha\ \mathsf{list}})$$

Derive the contradiction:

$$c\,(\mathsf{undef}_{\mathsf{num\ list}}) = d\,(\mathsf{undef}_{\mathsf{num} \times \mathsf{num}}) = \neg c\,(\mathsf{undef}_{\mathsf{num\ list}})$$

[2] Adapted from Kunčar CPP '15

Example [2]: Inconsistency from non-terminating dependencies

A theory with three constants $c_{\alpha\ \mathsf{list} \to \mathsf{bool}}$, $d_{(\alpha \times \beta) \to \mathsf{bool}}$, undef_α and two definitions:

$$c\,(x_{\alpha\ \mathsf{list}}) \equiv d(\mathsf{undef}_{\alpha \times \alpha}) \qquad d\,(x_{\alpha \times \mathsf{num}}) \equiv \neg c(\mathsf{undef}_{\alpha\ \mathsf{list}})$$

Derive the contradiction:

$$c\,(\mathsf{undef}_{\mathsf{num\ list}}) = d\,(\mathsf{undef}_{\mathsf{num} \times \mathsf{num}}) = \neg c\,(\mathsf{undef}_{\mathsf{num\ list}})$$

Replace d by c to use overloading.

[2] Adapted from Kunčar CPP '15

Example [2]: Inconsistency from non-terminating dependencies

A theory with three constants $c_{\alpha\ \mathsf{list} \to \mathsf{bool}}$, $d_{(\alpha \times \beta) \to \mathsf{bool}}$, undef_α and two definitions:

$$c\,(x_{\alpha\ \mathsf{list}}) \equiv d(\mathsf{undef}_{\alpha \times \alpha}) \qquad d\,(x_{\alpha \times \mathsf{num}}) \equiv \neg c(\mathsf{undef}_{\alpha\ \mathsf{list}})$$

Dependencies:

$$c_{\alpha\ \mathsf{list} \to \mathsf{bool}} \rightsquigarrow d_{(\alpha \times \alpha) \to \mathsf{bool}} \quad \text{and} \quad d_{(\alpha \times \mathsf{num}) \to \mathsf{bool}} \rightsquigarrow c_{\alpha\ \mathsf{list} \to \mathsf{bool}}$$

[2] Adapted from Kunčar CPP '15

Example [2]: Inconsistency from non-terminating dependencies

A theory with three constants $c_{\alpha\ \mathsf{list}\to\mathsf{bool}}$, $d_{(\alpha\times\beta)\to\mathsf{bool}}$, undef_α and two definitions:

$$c\,(x_{\alpha\ \mathsf{list}}) \equiv d(\mathsf{undef}_{\alpha\times\alpha}) \qquad d\,(x_{\alpha\times\mathsf{num}}) \equiv \neg c(\mathsf{undef}_{\alpha\ \mathsf{list}})$$

Dependencies:

$$c_{\alpha\ \mathsf{list}\to\mathsf{bool}} \rightsquigarrow d_{(\alpha\times\alpha)\to\mathsf{bool}} \quad \text{and} \quad d_{(\alpha\times\mathsf{num})\to\mathsf{bool}} \rightsquigarrow c_{\alpha\ \mathsf{list}\to\mathsf{bool}}$$

Non-terminating dependencies at instance $\alpha \mapsto \mathsf{num}$:

$$c_{\mathsf{num}\ \mathsf{list}\to\mathsf{bool}} \rightsquigarrow^{\downarrow} d_{(\mathsf{num}\times\mathsf{num})\to\mathsf{bool}} \rightsquigarrow^{\downarrow} c_{\mathsf{num}\ \mathsf{list}\to\mathsf{bool}}$$

[2] Adapted from Kunčar CPP '15

HOL with overloaded definitions

- Consistency by definitional (non axiomatic) theory extension, assuming definitions have terminating dependencies$^{\downarrow *}$.
  [Åman Pohjola et al. LPAR'20].

HOL WITH OVERLOADED DEFINITIONS

- Consistency by definitional (non axiomatic) theory extension, assuming definitions have terminating dependencies$^{\downarrow *}$.
  [Åman Pohjola et al. LPAR'20].
- Termination undecideable [Obua RTA'06]

HOL with overloaded definitions

- Consistency by definitional (non axiomatic) theory extension, assuming definitions have terminating dependencies$^{\downarrow *}$. [Åman Pohjola et al. LPAR'20].
- Termination undecideable [Obua RTA'06]
- Pen-and-paper proof [Kunčar CPP'15]:
  Composable dependencies have further structure.
  Orthogonal dependencies are decidable.

Theory for a Cyclicity Checker

- Pen-and-paper proof [Kunčar CPP'15]:

$$\begin{aligned}\vdash\ & \text{wellformed} \rightsquigarrow \ \wedge\ \text{monotone} \rightsquigarrow \\ & \wedge\ \text{finite} \rightsquigarrow\ \wedge\ \text{composable} \rightsquigarrow \\ & \Longrightarrow (\neg\text{terminating} \rightsquigarrow^{\downarrow *} \iff \text{cyclic} \rightsquigarrow)\end{aligned}$$

Theory for a Cyclicity Checker

- Pen-and-paper proof [Kunčar CPP'15]:

$$\begin{aligned} \vdash\ & \text{wellformed} \rightsquigarrow \ \wedge\ \text{monotone} \rightsquigarrow \\ & \wedge\ \text{finite} \rightsquigarrow \ \wedge\ \text{composable} \rightsquigarrow \\ & \Longrightarrow (\neg \text{terminating} \rightsquigarrow^{\downarrow *} \iff \text{cyclic} \rightsquigarrow) \end{aligned}$$

- Assuming *composable* $\rightsquigarrow$, checking non-termination of $\rightsquigarrow^{\downarrow *}$ equals finding cycles $\rightsquigarrow \rightsquigarrow^{\downarrow *}$.

Theory for a Cyclicity Checker

- Pen-and-paper proof [Kunčar CPP'15]:

$$\begin{aligned} \vdash\ & \text{wellformed} \rightsquigarrow\ \wedge\ \text{monotone} \rightsquigarrow \\ & \wedge\ \text{finite} \rightsquigarrow\ \wedge\ \text{composable} \rightsquigarrow \\ & \Longrightarrow (\neg \text{terminating} \rightsquigarrow^{\downarrow *} \iff \text{cyclic} \rightsquigarrow) \end{aligned}$$

- Assuming *composable* $\rightsquigarrow$, checking non-termination of $\rightsquigarrow^{\downarrow *}$ equals finding cycles $\rightsquigarrow\rightsquigarrow^{\downarrow *}$.
- We formalise, uncover bugs and fix proof. [Gengelbach et al., tech report '21]

Cyclicity Checker Algorithm

- Kunčar: *cyclic* is decideable for *orthogonal* dependencies

Cyclicity Checker Algorithm

- Kunčar: *cyclic* is decideable for *orthogonal* dependencies
- *Orthogonal*: each symbol depends on at most one symbol

Cyclicity Checker Algorithm

- Kunčar: *cyclic* is decideable for *orthogonal* dependencies
- *Orthogonal*: each symbol depends on at most one symbol
- *Orthogonal* is too restrictive:
  $\mathsf{size}_{\alpha\,\mathsf{list}\to\mathsf{num}} \rightsquigarrow \mathsf{length}_{\alpha\,\mathsf{list}\to\mathsf{num}}, \quad \mathsf{size}_{\alpha\,\mathsf{list}\to\mathsf{num}} \rightsquigarrow \alpha\,\mathsf{list}$

Cyclicity Checker Algorithm

- Kunčar: *cyclic* is decideable for *orthogonal* dependencies
- *Orthogonal*: each symbol depends on at most one symbol
- *Orthogonal* is too restrictive:
  $\mathsf{size}_{\alpha\,\mathsf{list}\to\mathsf{num}} \rightsquigarrow \mathsf{length}_{\alpha\,\mathsf{list}\to\mathsf{num}}, \quad \mathsf{size}_{\alpha\,\mathsf{list}\to\mathsf{num}} \rightsquigarrow \alpha\,\mathsf{list}$
- Thus: depth limited, breadth first search of cycles in $\rightsquigarrow\rightsquigarrow^{\downarrow *}$

Cyclicity Checker Algorithm (2)

- For $p \rightsquigarrow ({\rightsquigarrow^{\downarrow}}^{n})q$ (dependency chain of length $n+1$)
  Composable: limited ways extending $q \rightsquigarrow^{\downarrow} y$
  Acyclic: p is no type-instance of y

Cyclicity Checker Algorithm (2)

- For $p \rightsquigarrow (\rightsquigarrow^{\downarrow^n})q$ (dependency chain of length $n+1$)
  Composable: limited ways extending $q \rightsquigarrow^{\downarrow} y$
  Acyclic: p is no type-instance of y
- Soundness

$$\begin{aligned} &\vdash \text{wellformed} \rightsquigarrow \ \wedge \ \text{monotone} \rightsquigarrow \ \wedge \text{finite} \rightsquigarrow \\ &\quad \wedge \ (\forall n.\ \text{composable\_len} \rightsquigarrow n) \ \wedge \ (\forall n.\ \neg\text{cyclic\_len} \rightsquigarrow n) \\ &\quad \Longrightarrow \text{terminating} \rightsquigarrow^{\downarrow *} \end{aligned}$$

Cyclicity Checker Algorithm (2)

- For $p \rightsquigarrow (\rightsquigarrow^{\downarrow})^{n} q$ (dependency chain of length $n+1$)
  Composable: limited ways extending $q \rightsquigarrow^{\downarrow} y$
  Acyclic: p is no type-instance of y
- Soundness

$$\begin{aligned}\vdash\ & \text{wellformed} \rightsquigarrow\ \wedge\ \text{monotone} \rightsquigarrow\ \wedge \text{finite} \rightsquigarrow \\ & \wedge\ (\forall n.\ \text{composable\_len} \rightsquigarrow n)\ \wedge\ (\forall n.\ \neg\text{cyclic\_len} \rightsquigarrow n) \\ & \Longrightarrow \text{terminating} \rightsquigarrow^{\downarrow *}\end{aligned}$$

- Verified executable implementation in CakeML

Check Cycles in Isabelle/HOL

Theory	#Deps	Output	Runtime	Longest path
HOL	165	acyclic	0.01s	7
Orderings	764	acyclic	0.4s	13
Set	2657	acyclic	13s	14
Fun	2773	acyclic	13s	14
Transitive_Closure*	2195	acyclic	8s	20
Transitive_Closure	7159	acyclic	14h	35
Main*	12913	acyclic	12h	37
Main	45738	-	-	-

- Extracted dependencies from subtheories of Isabelle/HOL Main
- *: Only constant $\rightsquigarrow$ constant

Check Cycles in Isabelle/HOL

Theory	#Deps	Output	Runtime	Longest path
HOL	165	acyclic	0.01s	7
Orderings	764	acyclic	0.4s	13
Set	2657	acyclic	13s	14
Fun	2773	acyclic	13s	14
Transitive_Closure*	2195	acyclic	8s	20
Transitive_Closure	7159	acyclic	14h	35
Main*	12913	acyclic	12h	37
Main	45738	-	-	-

- Extracted dependencies from subtheories of Isabelle/HOL Main
- *: Only constant $\rightsquigarrow$ constant
- Naive algorithm

Check Cycles in Isabelle/HOL

Theory	#Deps	Output	Runtime	Longest path
HOL	165	acyclic	0.01s	7
Orderings	764	acyclic	0.4s	13
Set	2657	acyclic	13s	14
Fun	2773	acyclic	13s	14
Transitive_Closure*	2195	acyclic	8s	20
Transitive_Closure	7159	acyclic	14h	35
Main*	12913	acyclic	12h	37
Main	45738	-	-	-

- Extracted dependencies from subtheories of Isabelle/HOL Main
- *: Only constant $\rightsquigarrow$ constant
- Naive algorithm
- Overloading mainly through Haskell-like type classes

Verified kernel for HOL with overloading

- Implemented verified kernel for HOL in [Åman Pohjola et al. LPAR'20], assuming terminating dependencies
- Verified cyclicity checker discharges termination assumption

Verified kernel for HOL with overloading

- Implemented verified kernel for HOL in [Åman Pohjola et al. LPAR'20], assuming terminating dependencies
- Verified cyclicity checker discharges termination assumption
- From earlier work: kernel is formally proven model-theoretic conservative, Gengelbach et al. LFMTP'20.

Demo: Verified kernel for HOL with overloading

- Acyclic example defining $\mathsf{e} : A$ such that:

$$\mathsf{e}_{\mathsf{bool}} \equiv \mathsf{True} \qquad\qquad \mathsf{e}_{A \to A} \equiv \lambda x.\, x$$

Demo: Verified kernel for HOL with overloading

- Acyclic example defining $\mathtt{e} : A$ such that:

$$\mathtt{e}_{\mathsf{bool}} \equiv \mathsf{True} \qquad\qquad \mathtt{e}_{A \to A} \equiv \lambda x.\ x$$

- Cyclic example defining $\mathtt{c} : A$, $\mathtt{d} : A$ such that:

$$\mathtt{c}_{(A \to A) \to \mathsf{bool}} \equiv \lambda x.\ d(\mathsf{undef}_{A \times A})$$
$$\mathtt{d}_{(A \times A) \to \mathsf{bool}} \equiv \lambda x.\ \neg c(\mathsf{undef}_{A \to A})$$

Demo: Verified kernel for HOL with overloading

- Acyclic example defining $\mathtt{e} : A$ such that:

$$\mathtt{e}_{\mathsf{bool}} \equiv \mathsf{True} \qquad\qquad \mathtt{e}_{A \to A} \equiv \lambda x.\, x$$

- Cyclic example defining $\mathtt{c} : A$, $\mathtt{d} : A$ such that:

$$\mathtt{c}_{(A \to A) \to \mathsf{bool}} \equiv \lambda x.\, d(\mathsf{undef}_{A \times A})$$

$$\mathtt{d}_{(A \times A) \to \mathsf{bool}} \equiv \lambda x.\, \neg c(\mathsf{undef}_{A \to A})$$

- Thanks to Oskar Abrahamsson

Contribution

- Formally proved theory:
  there is structure in overloaded definitions.
- Verified checker
- Demonstrate checker on Isabelle/HOL basis
- Verified kernel for HOL with overloading

Code available online: https://code.cakeml.org/