A Type-Directed Negation Elimination

Etienne Lozes
LSV, ENS Cachan & CNRS
lozes@lsv.ens-cachan.fr

In the modal μ-calculus, a formula is well-formed if each recursive variable occurs underneath an even number of negations. By means of De Morgan's laws, it is easy to transform any well-formed formula φ into an equivalent formula without negations – the negation normal form of φ. Moreover, if φ is of size n, the negation normal form of φ is of the same size $\mathcal{O}(n)$. The full modal μ-calculus and the negation normal form fragment are thus equally expressive and concise.

In this paper we extend this result to the higher-order modal fixed point logic (HFL), an extension of the modal μ-calculus with higher-order recursive predicate transformers. We present a procedure that converts a formula of size n into an equivalent formula without negations of size $\mathcal{O}(n^2)$ in the worst case and $\mathcal{O}(n)$ when the number of variables of the formula is fixed.

1 Introduction

Negation normal forms are commonplace in many logical formalisms. To quote only two examples, in first-order logic, negation normal form is required by Skolemization, a procedure that distinguishes between existential and universal quantifiers; in the modal μ-calculus, the negation normal form ensures the existence of the fixed points. More generally, the negation normal form helps identifying the polarities [15] of the subformulas of a given formula; for instance, in the modal μ-calculus, a formula in negation normal form syntactically describes the schema of a parity game.

Converting a formula in a formula without negations – or with negations at the atoms only – is usually easy. By means of De Morgan's laws, negations can be "pushed to the leaves" of the formula. For the modal μ-calculus without propositional variables, this process completely eliminates negations, because well-formed formulas are formulas where recursive variables occur underneath an even number of negations. Moreover, in the modal μ-calculus, if φ is of size n, the negation normal form of φ is of the same size $\mathcal{O}(n)$.

The higher-order fixed point modal logic (HFL) [20] is the higher-order extension of the modal μ-calculus. In HFL, formulas denote either predicates, or (higher-order) predicate transformers, each being possibly defined recursively as (higher-order) fixed points. Since HFL was introduced, it was never suggested that negation could be eliminated from the logic. On the contrary, Viswananthan and Viswanathan [20] motivated HFL with an example expressing a form of rely guarantee that uses negation, and they strove to make sure that HFL formulas are correctly restricted so that fixed points always exist. Negation normal forms in HFL would however be interesting: they would simplify the design of two-player games for HFL model-checking [3], they could help defining a local model-checking algorithms for HFL, they might help to define the alternation depth of a HFL formula, etc.

We show that HFL actually admits negation elimination, and that like for the modal μ-calculus, every HFL formula can be converted into a formula in negation normal form. The negation elimination procedure is more involved due to higher-orderness. As a witness of this increased complexity, our negation elimination procedure has a worst-case quadratic blow-up in the size of the formula, whereas for the μ-calculus the negation normal form is of linear size in the original formula.

R. Matthes, M. Mio (Eds.): Fixed Points
in Computer Science 2015 (FICS 2015)
EPTCS 191, 2015, pp. 132–142, doi:10.4204/EPTCS.191.12

Related Work Other examples of higher-order recursive objects are the higher-order pushdown automata [17, 4], or the higher-order recursion schemes (HORS) [6, 12, 5, 18]. Whereas the decidability of HFL model-checking against finite transition systems is rather simple, it took more time to understand the decidability of HORS model-checking against the ordinary (order 0) modal μ-calculus. This situation actually benefited to HORS: the intense research on HORS produced several optimized algorithms and implementations of HORS model-checking [2, 9, 19], whereas HFL model-checking remains a rather theoretical and unexplored topic. HORS can be thought as recursive formulas with no boolean connectives and least fixed points everywhere. On the opposite, HFL allows any kinds of boolean connectives, and in particular a form of "higher-order alternation".

Outline We recall the definition of HFL and all useful background about it in Section 2. In Section 3, we sketch the ideas driving our negation elimination and introduce the notion of monotonization, a correspondence between arbitrary functions and monotone ones that is at the core of our negation elimination procedure. We formally define the negation elimination procedure in Section 4, and make some concluding remarks in Section 5.

2 The Higher-Order Modal Fixed Point Logic

We assume an infinite set $\mathsf{Var} = \{X, Y, Z, \dots\}$ of variables, and a finite set $\Sigma = \{a, b, \dots\}$ of labels. Formulas φ, ψ, of the Higher-Order Modal Fixed Point Logic (HFL) are defined by the following grammar

$$\varphi, \psi \quad ::= \top \mid \varphi \vee \psi \mid \neg\varphi \mid \langle a \rangle \varphi \mid X \mid \lambda X^{\tau,\nu}.\ \varphi \mid \varphi\ \psi \mid \mu X^{\tau}.\ \varphi$$

where a type τ is either the ground type Prop or an arrow type $\sigma^{\nu} \to \tau$, and the *variance* ν is either $+$ (monotone), or $-$ (antitone), or 0 (unrestricted). For instance, $\tau_1 = (\mathsf{Prop}^{-} \to \mathsf{Prop})^{+} \to (\mathsf{Prop}^{0} \to \mathsf{Prop})$ is a type, and $\varphi_1 = \lambda F^{\mathsf{Prop}^{-} \to \mathsf{Prop},+}.\ \lambda Y^{\mathsf{Prop},0}.\ \mu Z^{\mathsf{Prop}}.\ (F\ \neg Y) \vee \langle a \rangle (Z \vee \neg Y)$ is a formula. The sets $\mathsf{fv}(\varphi)$ and $\mathsf{bv}(\varphi)$ of free and bound variables of φ are defined as expected: $\mathsf{fv}(X) = \{X\}$, $\mathsf{bv}(X) = \emptyset$, $\mathsf{fv}(\lambda X.\ \varphi) = \mathsf{fv}(\mu X.\ \varphi) = \mathsf{fv}(\varphi) \setminus \{X\}$, $\mathsf{bv}(\lambda X.\ \varphi) = \mathsf{bv}(\mu X.\ \varphi) = \mathsf{bv}(\varphi) \cup \{X\}$, etc. A formula is *closed* if $\mathsf{fv}(\varphi) = \emptyset$. For simplicity, we restrict our attention to formulas φ *without variable masking*, i.e. such that for every subformula $\lambda X.\ \psi$ (resp. $\mu X.\ \psi$), it holds that $X \notin \mathsf{bv}(\psi)$.

Another example is the formula $\varphi_2 = (\lambda F^{\mathsf{Prop}^{-} \to \mathsf{Prop},+}.\ \mu X^{\mathsf{Prop}}.\ F\ X)\ (\lambda Y^{\mathsf{Prop},-}.\ \neg Y)$. This formula can be β-reduced to the modal μ-calculus formula $\varphi_2' = \mu X^{\mathsf{Prop}}.\ \neg X$, which does not have a fixed point semantics. Avoiding ill-formed HFL formulas such as φ_2 cannot just rely on counting the number of negations between μX and the occurence of X, it should also take into account function applications and the context of a subformula.

A type judgement is a tuple $\Gamma \vdash \varphi : \tau$, where Γ is a set of assumptions of the form $X^{\nu} : \tau$. The typing environment $\neg\Gamma$ is the one in which every assumption $X^{\nu} : \tau$ is replaced with $X^{-\nu} : \tau$, where $-+ = -$, $-- = +$, and $-0 = 0$. A formula φ is well-typed and has type τ if the type judgement $\vdash \varphi : \tau$ is derivable from the rules defined in Fig. 1. Intuitively, the type judgement $X_1^{\nu_1} : \tau_1, \dots, X_n^{\nu_n} : \tau_n \vdash \varphi : \tau$ is derivable if asssuming that X_i has type τ_i, it may be infered that φ has type τ and that φ, viewed as a function of X_i, has variance ν_i. For instance, $\vdash \varphi_1 : \tau_1$, where φ_1 and τ_1 are the formula and the type we defined above, but φ_2 cannot be typed, even with different type annotations.

Proposition 1 *[20] If $\Gamma \vdash \varphi : \tau$ and $\Gamma \vdash \varphi : \tau'$ are derivable, then $\tau = \tau'$, and the two derivations coincide.*

$$\frac{}{\Gamma \vdash \top : \mathsf{Prop}} \qquad \frac{\Gamma \vdash \varphi : \tau \qquad \Gamma \vdash \psi : \tau}{\Gamma \vdash \varphi \vee \psi : \tau} \qquad \frac{\neg\Gamma \vdash \varphi : \tau}{\Gamma \vdash \neg\varphi : \tau} \qquad \frac{\Gamma \vdash \varphi : \mathsf{Prop}}{\Gamma \vdash \langle a \rangle \varphi : \mathsf{Prop}} \qquad \frac{\nu \in \{+, 0\}}{\Gamma,\, X^{\nu} : \tau \vdash X : \tau}$$

$$\frac{\Gamma, X^{\nu} : \sigma \vdash \varphi : \tau}{\Gamma \vdash \lambda X^{\nu,\sigma}.\, \varphi : \sigma^{\nu} \to \tau} \qquad \frac{\Gamma, X^{+} : \tau \vdash \varphi : \tau}{\Gamma \vdash \mu X^{\tau}.\, \varphi : \tau} \qquad \frac{\Gamma \vdash \varphi : \sigma^{+} \to \tau \qquad \Gamma \vdash \psi : \sigma}{\Gamma \vdash \varphi\ \psi : \tau}$$

$$\frac{\Gamma \vdash \varphi : \sigma^{-} \to \tau \qquad \neg\Gamma \vdash \psi : \sigma}{\Gamma \vdash \varphi\ \psi : \tau} \qquad \frac{\Gamma \vdash \varphi : \sigma^{0} \to \tau \qquad \Gamma \vdash \psi : \sigma \qquad \neg\Gamma \vdash \psi : \sigma}{\Gamma \vdash \varphi\ \psi : \tau}$$

Figure 1: The type system of HFL.

If φ is a well-typed closed formula and ψ is a subformula of φ, we write $\mathsf{type}(\psi/\varphi)$ for the type of ψ in (the type derivation of) φ.

A labeled transition system (LTS) is a tuple $\mathscr{T} = (S, \delta)$ where S is a set of states and $\delta \subseteq S \times \Sigma \times S$ is a transition relation. For every type τ and every LTS $\mathscr{T} = (S, \delta)$, the complete Boolean ring $\mathscr{T}[\![\tau]\!]$ of interpretations of closed formulas of type τ is defined by induction on τ: $\mathscr{T}[\![\mathsf{Prop}]\!] = 2^S$, and $\mathscr{T}[\![\sigma^{\nu} \to \tau]\!]$ is the complete Boolean ring of all total functions $f : \mathscr{T}[\![\sigma]\!] \to \mathscr{T}[\![\tau]\!]$ that have variance ν, where all Boolean operations on functions are understood pointwise. Note that since $\mathscr{T}[\![\tau]\!]$ is a complete Boolean ring, it is also a complete lattice, and any monotone function $f : \mathscr{T}[\![\tau]\!] \to \mathscr{T}[\![\tau]\!]$ admits a unique least fixed point.

A $\mathscr{T}$-valuation ρ is a function that sends every variable of type τ to some element of $\mathscr{T}[\![\tau]\!]$. More precisely, we say that ρ is well-typed according to some typing environment Γ, which we write $\rho \models \Gamma$, if $\rho(X) \in \mathscr{T}[\![\tau]\!]$ for every $X^{\nu} : \tau$ in Γ. The semantics $\mathscr{T}[\![\Gamma \vdash \varphi : \tau]\!]$ of a derivable typing judgement is a function that associates to every $\rho \models \Gamma$ an interpretation $\mathscr{T}[\![\Gamma \vdash \varphi : \tau]\!](\rho)$ in $\mathscr{T}[\![\tau]\!]$; this interpretation is defined as expected by induction on the derivation tree (see [20] for details). For a well-typed closed formula φ of type Prop, a LTS $\mathscr{T} = (S, \delta)$ and a state $s \in S$, We write $s \models_{\mathscr{T}} \varphi$ if $s \in \mathscr{T}[\![\vdash \varphi : \mathsf{Prop}]\!]$.

Example 1 *Let* $\tau_3 = (\mathsf{Prop}^{+} \to \mathsf{Prop})^{+} \to \mathsf{Prop}^{+} \to \mathsf{Prop}$ *and* $\varphi_3 =$

$$\left(\mu F^{\tau_3}.\, \lambda G^{\mathsf{Prop}^{+} \to \mathsf{Prop}}, X^{\mathsf{Prop}}.\, (G\, X) \vee \left(F\ (\lambda Y^{\mathsf{Prop}}.G\, (G\, Y))\ X\right)\right) \quad (\lambda Z^{\mathsf{Prop}}.\, \langle a \rangle Z) \quad \langle b \rangle \top.$$

Then $s \models \varphi_3$ *iff there is* $n \geq 0$ *such that there is a path of the form* $a^{2^n} b$ *starting at* s. *Since* $\{a^{2^n} b \mid n \geq 0\}$ *is not a regular language, the property expressed by* φ_3 *cannot be expressed in the modal* μ*-calculus.*

Proposition 2 *[20] Let* $\mathscr{T} = (S, \delta)$ *be a LTS and let* $s, s' \in S$ *be two bisimilar states of* $\mathscr{T}$. *Then for any closed formula* φ *of type* Prop, $s \models_{\mathscr{T}} \varphi$ *iff* $s' \models_{\mathscr{T}} \varphi$.

We assume the standard notations $\wedge$, $[a]$ and $\nu X.\, (.)$ for the conjunction, the necessity modality, and the greatest fixed point, defined as the duals of $\vee$, $\langle a \rangle$ and $\mu X.\, (.)$ respectively.

Definition 1 (Negation Normal Form) *A HFL formula is in negation normal form if it is derivable from the grammar*

$$\varphi, \psi \ ::= \top \mid \bot \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi \mid [a]\varphi \mid X \mid \lambda X^{\sigma}.\varphi \mid \varphi\ \psi \mid \mu X^{\tau}.\varphi \mid \nu X^{\tau}.\varphi$$

where the τ *are* monotone *types, i.e. types where all variances are equal to* $+$.

Note that since all variances are $+$, we omit them when writting formulas in negation normal form.

We say that two formulas φ, ψ are equivalent, $\varphi \equiv \psi$, if for every type environment Γ, for every LTS $\mathscr{T}$, for all type τ, the judgement $\Gamma \vdash \varphi : \tau$ is derivable iff $\Gamma \vdash \psi : \tau$ is, and in that case $\mathscr{T}[\![\Gamma \vdash \varphi : \tau]\!] = \mathscr{T}[\![\Gamma \vdash \psi : \tau]\!]$.

Model-Checking We briefly recall the results known about the data complexity of HFL model-checking (see also the results of Lange *et al* on the combined complexity [1] or the descriptive complexity [14] of HFL and extensions).

Note that if $\mathscr{T} = (S, \delta)$ is a finite LTS, then for all type τ, the Boolean ring $\mathscr{T}[\![\tau]\!]$ is a finite set, and every element of $\mathscr{T}[\![\tau]\!]$ can be represented *in extension*. Moreover, the least fixed point of a monotone function $f : \mathscr{T}[\![\tau]\!] \to \mathscr{T}[\![\tau]\!]$ can be computed by iterating f at most n times, where n is the size of the finite boolean ring $\mathscr{T}[\![\tau]\!]$.

The order $\mathsf{ord}(\tau)$ of a type τ is defined as $\mathsf{ord}(\mathsf{Prop}) = 0$ and $\mathsf{ord}(\sigma^{\nu} \to \tau) = \max(\mathsf{ord}(\tau), 1 + \mathsf{ord}(\sigma))$. We write $\mathsf{HFL}(k)$ to denote the set of closed HFL formulas φ of type Prop such that all type annotations in φ are of order at most k. For every fixed $\varphi \in \mathsf{HFL}(k)$, we call $\mathsf{MC}(\varphi)$ the problem of deciding, given a LTS $\mathscr{T}$ and a state s of $\mathscr{T}$, wether $s \models_{\mathscr{T}} \varphi$.

Theorem 3 *[1] For every $k \geq 1$, for every $\varphi \in \mathsf{HFL}(k)$, the problem $\mathsf{MC}(\varphi)$ is in k-EXPTIME, and there is a $\psi_k \in \mathsf{HFL}(k)$ such that $\mathsf{MC}(\psi_k)$ is k-EXPTIME hard.*

3 Monotonization

In order to define a negation elimination procedure, the first idea is probably to reason like in the modal μ-calculus, and try to "push the negations to the leaves". Indeed, there are De Morgan laws for all logical connectives, including abstraction and application, since

$$\neg(\varphi\ \psi) \quad \equiv \quad (\neg\varphi)\ \psi \qquad\qquad \text{and} \qquad\qquad \neg(\lambda X^{\nu,\tau}.\psi) \quad \equiv \quad \lambda X^{-\nu,\tau}.\neg\psi.$$

In the modal μ-calculus, this idea is enough, because the "negation counting" criterion ensures that each pushed negation eventually reaches another negation and both anihilate. This does not happen for HFL. Consider for instance the formula $\varphi_4 =$

$$\left(\mu X^{\mathsf{Prop}^0 \to \mathsf{Prop}}.\ \lambda Y^{\mathsf{Prop},0}.\ (\neg Y) \vee \left(X\ (\langle a \rangle Y)\right)\right) \quad \top.$$

The negation already is at the leaf, but φ_4 is not in negation normal form. By fixed point unfolding, one can check that φ_4 is equivalent to the infinite disjunct $\bigvee_{n \geq 0} [a]^n \bot$, and thus could be expressed by $\mu X^{\mathsf{Prop}}.[a]X$. The generalization of this strategy for arbitrary formulas would be interesting, but it is unclear to us how it would be defined.

We follow another approach: we do not try to unfold fixed points nor to apply β-reductions during negation elimination, but we stick to the structure of the formula. In particular, in our approach a subformula denoting a function f is mapped to a subformula denoting a function f' in the negation normal form. Note that even if f is not monotone, f' must be monotone since it is a subformula of a formula in negation normal form. We call f' a *monotonization* of f.

Examples Before we formaly define monotonization, we illustrate its principles on some examples.

First, consider again the above formula φ_4. This formula contains the function $\lambda Y^{\mathsf{Prop},0}.\ (\neg Y) \vee \left(X\ (\langle a \rangle Y)\right)$. This function is unrestricted (neither monotone nor antitone). The monotonization of this

function will be the function $\lambda Y^{\mathsf{Prop},+},\overline{Y}^{\mathsf{Prop},+}.\overline{Y} \vee \big(X\ (\langle a\rangle Y)\big)$. To obtain this function, a duplicate $\overline{Y}$ of Y is introduced, and is used in place of $\neg Y$. Finally, the formula $\varphi_4' =$

$$\big(\mu X^{\mathsf{Prop}\to\mathsf{Prop}\to\mathsf{Prop}}.\ \lambda Y^{\mathsf{Prop}},\overline{Y}^{\mathsf{Prop}}.\ \overline{Y} \vee \big(X\ (\langle a\rangle Y)\ ([a]\overline{Y})\big)\big) \quad \top \qquad \bot$$

can be used as a negation normal form of φ_4. Note that the parameter $\top$ that was passed to the recursive function in φ_4 is duplicated in φ_4', with one duplicate that has been negated (the $\bot$ formula).

More generally, whenever a function is of type $\sigma^0 \to \tau$, we transform it into a function of type $\sigma_t^+ \to \sigma_t^+ \to \tau_t$ that takes two arguments of type σ_t (the translation of σ). Later, when this function is applied, we make sure that its argument is duplicated, one time positively, the other negatively.

Duplicating arguments might cause an exponential blow-up. For instance, for the formula $\varphi_5 =$

$$(\lambda X^{\mathsf{Prop}}.\ X \vee \langle a\rangle\neg X) \quad \big((\lambda Y^{\mathsf{Prop},0}.\ Y \vee \langle b\rangle\neg Y)\ \top\big)$$

if we duplicated arguments naively, we could get the formula $\varphi_5' =$

$$(\lambda X^{\mathsf{Prop}},\overline{X}^{\mathsf{Prop}}.\ X \vee \langle a\rangle\overline{X}) \quad \big((\lambda Y^{\mathsf{Prop}},\overline{Y}^{\mathsf{Prop}}.\ Y \vee \langle b\rangle\overline{Y})\ \top\ \bot\big) \quad \big((\lambda Y^{\mathsf{Prop}},\overline{Y}^{\mathsf{Prop}}.\ \overline{Y} \wedge [b]Y)\ \top\ \bot\big)$$

where the original $\top$ formula has been duplicated. If it occurred underneath $n+2$ applications of an unrestricted function, we would have 2^n copies of $\top$. We will come back to this problem in Section 4.

Let us now observe how monotonization works for functions that are antitone. In general, if f is an antitone function, both the "negation at the caller" $f_1(x) = \neg f(x)$ and the "negation at the callee" $f_2(x) = f(\neg x)$ are two monotone functions that faithfully represent f. Actually, both of them might be needed by our negation elimination procedure.

Consider the formula $\varphi_6 =$

$$(\lambda F^{\mathsf{Prop}^-\to\mathsf{Prop},+}.\mu X^{\mathsf{Prop}}.F\ (\neg X)) \quad (\lambda Y^{\mathsf{Prop},-}.\neg\langle a\rangle Y).$$

In order to compute the negation normal form of φ_6, we may represent $\lambda Y^{\mathsf{Prop},-}.\neg\langle a\rangle Y$ by its "negation at the callee", yielding the formula $\varphi_6' =$

$$(\lambda F^{\mathsf{Prop}\to\mathsf{Prop}}.\mu X^{\mathsf{Prop}}.F\ X) \quad (\lambda\overline{Y}^{\mathsf{Prop}}.[a]\overline{Y}).$$

Conversely, consider the formula $\varphi_7 =$

$$(\lambda F^{\mathsf{Prop}^-\to\mathsf{Prop},-}.\ \mu X^{\mathsf{Prop}}.\ (\neg F)\ X) \quad (\lambda Y^{\mathsf{Prop},-}.\ \neg\langle a\rangle Y).$$

The only difference with φ_6 is that the negation is now in front of F instead of X. In that case, "negation at the callee" does not help eliminating negations. But "negation at the caller" does, and yields the negation normal form $\varphi_7' =$

$$(\lambda\overline{F}^{\mathsf{Prop}\to\mathsf{Prop}}.\ \mu X^{\mathsf{Prop}}.\ \overline{F}\ X) \quad (\lambda Y^{\mathsf{Prop}}.\ \langle a\rangle Y).$$

These examples suggest a negation elimination that proceeds along possibly different strategies in the case of an application $\varphi\ \psi$, depending on the semantics of φ and ψ. In the next section, we explain how the strategy is determined by the type of φ. For now, we focus on making more formal our notion of monotonization.

$$
\begin{array}{ll}
\exp(\mathsf{Prop}) & = \mathsf{Prop} \\
\exp(\tau^+ \to \sigma) & = \exp(\tau)^+ \to \exp(\sigma) \\
\exp(\tau^- \to \sigma) & = \exp(\tau)^+ \to \exp(\sigma) \\
\exp(\tau^0 \to \sigma) & = \exp(\tau)^+ \to \exp(\tau)^+ \to \exp(\sigma)
\end{array}
\qquad
\begin{array}{ll}
\exp(\Gamma_1, \Gamma_2) & = \exp(\Gamma_1), \exp(\Gamma_2) \\
\exp(X^+ : \tau) & = X^+ : \exp(\tau) \\
\exp(X^- : \tau) & = \overline{X}^+ : \exp(\tau) \\
\exp(X^0 : \tau) & = X^+ : \exp(\tau), \overline{X}^+ : \exp(\tau)
\end{array}
$$

Figure 2: Expansion of types and typing environments towards monotonization.

Monotonization Relations We saw that our negation elimination bases on the ability to faithfully represent a predicate transformer φ by a monotone predicate transformer ψ; in this case, we will say that ψ is a *monotonization* of φ. We now aim at defining formally this notion. More precisely, we aim at defining the relation $\triangleleft$ such that $\varphi \triangleleft \psi$ holds if ψ is a monotonization of φ.

First of all, $\triangleleft$ relates a formula of type τ to a formula of type $\exp(\tau)$ as defined in Fig. 2: the number of arguments of φ is duplicated if φ is unrestricted, otherwise it remains the same, and of course ψ is monotone in all of its arguments.

In Fig. 2, we also associate to every typing environment Γ the typing environment $\exp(\Gamma)$ with all variances set to $+$, obtained after renaming all variables with variance $-$ in their bared version, and duplicating all variables with variance 0. In the remainder, we always implicitly assume that we translate formulas and typing environments that do not initially contain bared variables.

The relation $\triangleleft$ is then defined coinductively, in a similar way as logical relations for the λ-calculus. Let R be a binary relation among typing judgements of the form $\Gamma \vdash \varphi : \tau$. The relation R is well-typed if $(\Gamma \vdash \varphi : \tau)\, R\, (\Gamma' \vdash \varphi' : \tau')$ implies $\Gamma' = \exp(\Gamma)$ and $\tau' = \exp(\tau)$. When R is well typed, we write $\varphi R_{\Gamma,\tau} \varphi'$ instead of $(\Gamma \vdash \varphi : \tau)\, R\, (\Gamma' \vdash \varphi' : \tau')$.

Definition 2 *A binary relation R among typing judgements is a* monotonization relation *if it is well-typed, and for all formulas φ, φ', for all Γ, τ such that $\varphi\, R_{\Gamma,\tau}\, \varphi'$,*

1. *if φ, φ' are closed and $\tau = \mathsf{Prop}$, then $\varphi \equiv \varphi'$;*
2. *if $\Gamma = \Gamma', X^+ : \sigma$, then $(\lambda X^{\sigma,+}.\ \varphi)\, R_{\Gamma', \sigma^+ \to \tau}\, (\lambda X^{\exp(\sigma),+}.\ \varphi')$;*
3. *if $\Gamma = \Gamma', X^- : \sigma$, then $(\lambda X^{\sigma,-}.\ \varphi)\, R_{\Gamma', \sigma^- \to \tau}\, (\lambda \overline{X}^{\exp(\sigma),+}.\ \varphi')$;*
4. *if $\Gamma = \Gamma', X^0 : \sigma$, then $(\lambda X^{\sigma,0}.\ \varphi)\, R_{\Gamma', \sigma^0 \to \tau}\, (\lambda X^{\exp(\sigma),+}, \overline{X}^{\exp(\sigma),+}.\ \varphi')$;*
5. *if $\tau = \sigma^+ \to \upsilon$, then for all ψ, ψ' such that $\psi\, R_{\Gamma,\sigma}\, \psi'$, $(\varphi\ \psi)\, R_{\Gamma,\upsilon}\, (\varphi'\ \psi')$;*
6. *if $\tau = \sigma^- \to \upsilon$, then for all ψ, ψ', ψ'' such that $\psi\, R_{\Gamma,\sigma}\, \psi'$ and $\psi' \equiv \neg\psi''$, $(\varphi\ \psi)\, R_{\Gamma,\upsilon}\, (\varphi'\ \psi'')$;*
7. *if $\tau = \sigma^0 \to \upsilon$, then for all ψ, ψ', ψ'' such that $\psi\, R_{\Gamma,\sigma}\, \psi'$ and $\psi' \equiv \neg\psi''$, $(\varphi\ \psi)\, R_{\Gamma,\upsilon}\, (\varphi'\ \psi'\ \psi'')$.*

If $(R_i)_{i \in I}$ is a family of monotonization relation, then so is $\bigcup_{i \in I} R_i$; we write $\triangleleft$ for the largest monotonization relation.

Example 2 *Consider $\varphi = (\lambda X^{\mathsf{Prop},-}.\ \neg X)$. Then $\varphi \triangleleft_{\mathsf{Prop}^- \to \mathsf{Prop}} (\lambda \overline{X}^{\mathsf{Prop},+}.\ \overline{X})$. Consider also $\psi = (\lambda X^{\mathsf{Prop},0}.\ X \wedge \neg X)$. Then $\psi \triangleleft (\lambda X^{\mathsf{Prop},+}, \overline{X}^{\mathsf{Prop},+}.\ \bot)$ and $\psi \triangleleft (\lambda X^{\mathsf{Prop},+}, \overline{X}^{\mathsf{Prop},+}.\ X \wedge \overline{X})$.*

$$
\begin{array}{rcl@{\qquad}rcl}
\mathsf{tr}_+(\top) & = & \top & \mathsf{tr}_+(\psi_1 \vee \psi_2) & = & \mathsf{tr}_+(\psi_1) \vee \mathsf{tr}_+(\psi_2) \\
\mathsf{tr}_-(\top) & = & \bot & \mathsf{tr}_-(\psi_1 \vee \psi_2) & = & \mathsf{tr}_-(\psi_1) \wedge \mathsf{tr}_-(\psi_2) \\
\mathsf{tr}_+(X) & = & X & \mathsf{tr}_v(\lambda X^{\tau,+}.\ \psi) & = & \lambda X^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_v(\psi) \\
\mathsf{tr}_-(X) & = & \overline{X} & \mathsf{tr}_v(\lambda X^{\tau,-}.\ \psi) & = & \lambda \overline{X}^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_v(\psi) \\
\mathsf{tr}_v(\neg\psi) & = & \mathsf{tr}_{-v}(\psi) & \mathsf{tr}_v(\lambda X^{\tau,0}.\ \psi) & = & \lambda X^{\mathsf{exp}(\tau)}, \overline{X}^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_v(\psi) \\
\mathsf{tr}_+(\langle a\rangle\psi) & = & \langle a\rangle \mathsf{tr}_+(\psi) & \mathsf{tr}_+(\mu X^{\tau}.\ \psi) & = & \mu X^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_+(\psi) \\
\mathsf{tr}_-(\langle a\rangle\psi) & = & [a]\mathsf{tr}_-(\psi) & \mathsf{tr}_-(\mu X^{\tau}.\ \psi) & = & \nu \overline{X}^{\mathsf{exp}(\tau)}.\ \mathsf{tr}_-(\psi)
\end{array}
$$

$$
\mathsf{tr}_v(\psi_1\ \psi_2) \quad = \quad \begin{cases} \mathsf{tr}_v(\psi_1)\ \mathsf{tr}_+(\psi_2) & \text{if } \mathsf{type}(\psi_1/\varphi) = \sigma^+ \to \eta \\ \mathsf{tr}_v(\psi_1)\ \mathsf{tr}_-(\psi_2) & \text{if } \mathsf{type}(\psi_1/\varphi) = \sigma^- \to \eta \\ \mathsf{tr}_v(\psi_1)\ \mathsf{tr}_+(\psi_2)\ \mathsf{tr}_-(\psi_2) & \text{if } \mathsf{type}(\psi_1/\varphi) = \sigma^0 \to \eta \end{cases}
$$

Figure 3: Type-Directed Negation Elimination

4 Negation Elimination

Our negation elimination procedure proceeds in two steps: first, a formula φ is translated into a formula $\mathsf{tr}_+(\varphi)$ that denotes the monotonization of φ; then, $\mathsf{tr}_+(\varphi)$ is concisely represented in order to avoid an exponential blow-up.

The transformation $\mathsf{tr}_+(.)$ is presented in Figure 3. The transformation proceeds by structural induction on the formula, and is defined as a mutual induction with the companion transformation $\mathsf{tr}_-(.)$. Whenever a negation is encountered, it is eliminated and the dual transformation is used. As a consequence, wether $\mathsf{tr}_+(.)$ or $\mathsf{tr}_-(.)$ should be used for a given subformula depends on the polarity [15] of this subformula.

Lemma 4 *Let φ be a fixed closed formula of type* Prop. *For every subformula ψ of φ, let $\mathsf{tr}_+(\psi)$ and $\mathsf{tr}_-(\psi)$ be defined as in Figure 3, and let $\Gamma \vdash \psi : \tau$ be the type judgement associated to ψ in the type derivation of φ. Then the following statements hold.*

1. $\mathsf{exp}(\Gamma) \vdash \mathsf{tr}_+(\psi) : \mathsf{exp}(\tau)$ *and* $\mathsf{exp}(\neg\Gamma) \vdash \mathsf{tr}_-(\psi) : \mathsf{exp}(\tau)$.
2. $\psi \lhd_{\Gamma,\tau} \mathsf{tr}_+(\psi)$ *and* $\psi \lhd_{\Gamma,\tau} \neg\mathsf{tr}_-(\psi)$.

Proof: By induction on ψ. We only detail the point 1 in the case of $\psi = \psi_1\ \psi_2$ with $\mathsf{type}(\psi_1/\varphi) = \sigma^- \to \tau$. Let us assume the two statements hold for ψ_1 and ψ_2 by induction hypothesis. Let Γ be such that $\Gamma \vdash \psi : \tau$, $\Gamma \vdash \psi_1 : \sigma^- \to \tau$, and $\neg\Gamma \vdash \psi_2 : \sigma$. By induction hypothesis, the judgements $\mathsf{exp}(\Gamma) \vdash \mathsf{tr}_+(\psi_1) : \mathsf{exp}(\sigma^- \to \tau)$ and $\mathsf{exp}(\neg\neg\Gamma) \vdash \mathsf{tr}_-(\psi_2) : \mathsf{exp}(\sigma)$ are derivable. Since $\mathsf{exp}(\sigma^- \to \tau) = \mathsf{exp}(\sigma)^+ \to \mathsf{exp}(\tau)$ and $\neg\neg\Gamma = \Gamma$, the typing rule for function application in the monotone case of Fig. 1 yields $\mathsf{exp}(\Gamma) \vdash \mathsf{tr}_+(\psi_1)\ \mathsf{tr}_-(\psi_2) : \mathsf{exp}(\tau)$, which shows statement 1 for $\mathsf{tr}_+(.)$. The case for $\mathsf{tr}_-(.)$ is similar. □

Corollary 5 *If φ is a closed formula of type* Prop, *then $\varphi \equiv \mathsf{tr}_+(\varphi)$ and $\mathsf{tr}_+(\varphi)$ is in negation normal form.*

As observed in Section 3, the duplication of the arguments in the case $v = 0$ of the monotonization of $\varphi\psi$ may cause an exponential blow-up in the size of the formula. However, this blow-up does not happen if we allow some sharing of identical subformulas.

Let φ be a fixed closed formula. We say that two subformulas ψ_1 and ψ_2 of φ are identical if they are syntactically equivalent and if moreover they have the same type and are in a same typing context,

i.e. if the type derivation of φ goes through the judgements $\Gamma_i \vdash \psi_i : \tau_i$ for syntactically equivalent Γ_i and τ_i. For instance, in the formula

$$(\lambda X^{\mathsf{Prop}\to\mathsf{Prop}}.\, X) \quad \big((\lambda X^{(\mathsf{Prop}\to\mathsf{Prop})\to(\mathsf{Prop}\to\mathsf{Prop})}.\, X) \quad \big((\lambda Y^{\mathsf{Prop}\to\mathsf{Prop}}.Y)\ \top\big)\big)$$

any two distinct subformulas are not identical (including the subformulas restricted to X). We call *dag size* of φ the number of non-identical subformulas of φ.

Lemma 6 *There is a logspace computable function* $\mathsf{share}(.)$ *that associates to every closed formula* φ *of dag size n a closed formula* $\mathsf{share}(\varphi)$ *of tree size* $\mathscr{O}(n\cdot|\mathsf{vars}(\varphi)|)$ *such that* $\varphi \equiv \mathsf{share}(\varphi)$.

Proof: Let φ be fixed, and let $\varphi_1\ldots,\varphi_n$ be an enumeration of all subformulas of φ such that if φ_i is a strict subformula of φ_j, then $i<j$. In particular, we must have $\varphi=\varphi_n$. Pick some fresh variables $X_1,X_2,\ldots,X_n \in \mathsf{Var}$ and let $\upsilon_i = \mathsf{type}(\varphi_i/\varphi)$. For every $i=1,\ldots n$, let $Y_1,\sigma_1,\nu_1,\ldots Y_k,\sigma_k,\nu_k$ be a fixed enumeration of the free variables of φ_i, their types and their variances, and let $\lambda_i(\psi) = \lambda Y_1^{\sigma_1,\nu_1},\ldots,Y_k^{\sigma_k,\nu_k}.\ \psi$ and $@_i(\psi) = \psi\ Y_1\ \ldots\ Y_k$. Finally, let $\tau_i = \sigma_1^{\nu_1}\to\ldots\sigma_k^{\nu_k}\to\upsilon_i$. For every subformula ψ of φ, let $\|\psi\|$ be defined by case analysis on the first logical connective of ψ:

- if $\psi=\varphi_i=\eta Y^\sigma.\ \varphi_j$, where $\eta\in\{\lambda,\mu,\nu\}$, then $\|\psi\| = \lambda_i\big(\eta Y^\sigma.\ @_j(X_j)\big)$;
- if $\psi=\varphi_i=\varphi_j\oplus\varphi_k$, where $\oplus\in\{\vee,\wedge,\text{application}\}$, then $\|\psi\| = \lambda_i\big(@_j(X_j)\oplus @_k(X_k)\big)$;
- if $\psi=\varphi_i=\spadesuit\varphi_j$, where $\spadesuit\in\{\neg,\langle a\rangle,[a]\}$, then $\|\psi\| = \lambda_i\big(\spadesuit(@_j(X_j))\big)$;
- otherwise $\|\varphi_i\| = \lambda_i(\varphi_i)$.

Finally, let $\mathsf{share}(\varphi) = \textbf{let}\ X_1^{\tau_1} = \|\varphi_1\|\ \textbf{in let}\ X_2^{\tau_2} = \|\varphi_2\|\ \textbf{in}\ \ldots\ \textbf{let}\ X_{n-1}^{\tau_{n-1}} = \|\varphi_{n-1}\|\ \textbf{in}\ \|\varphi_n\|$ where $\textbf{let}\ X^\tau=\psi\ \textbf{in}\ \psi'$ is a macro for $(\lambda X^\tau.\ \psi')\ \psi$. Then $\mathsf{share}(\varphi)$ has the desired properties. □

Theorem 7 *There is a logspace-computable function* $\mathsf{nnf}(.)$ *that associates to every closed HFL formula* φ *(without variable masking) of type* Prop *a closed formula* $\mathsf{nnf}(\varphi)$ *such that*

1. $\varphi\equiv\mathsf{nnf}(\varphi)$,
2. $\mathsf{nnf}(\varphi)$ *is in negation normal form, and*
3. $|\mathsf{nnf}(\varphi)| = \mathscr{O}(|\varphi|\cdot|\mathsf{vars}(\varphi)|)$,

where $|\psi|$ *denotes the size of the tree representation of* ψ *(i.e. the number of symbols in* ψ*), and* $\mathsf{vars}(\varphi) = \mathsf{fv}(\varphi)\cup\mathsf{bv}(\varphi)$ *is the set of variables that occur in* φ.

Proof: Let $\mathsf{nnf}(\varphi)=\mathsf{share}(\mathsf{tr}_+(\varphi))$. This function is logspace computable ($\mathsf{tr}_+(\varphi)$ can be computed "on-the-fly") and $\mathsf{nnf}(\varphi)$ is of size $\mathscr{O}(|\varphi|\cdot|\mathsf{vars}(\varphi)|)$ by Figure 3 and Lemma 6. The formula $\mathsf{tr}_+(\varphi)$ is in negation normal form, and $\mathsf{share}(.)$ does not introduce new negations, so $\mathsf{nnf}(\varphi)$ is in negation normal form. Looking back at Figure 3, it can be checked that its dag size is linear in the dag size of φ, so the tree size of $\mathsf{nnf}(\varphi)$ is linear in the tree size of φ. Moreover, $\mathsf{nnf}(\varphi)\equiv\mathsf{tr}_+(\varphi)$ by Lemma 6, and $\mathsf{tr}_+(\varphi)\equiv\varphi$ by Corollary 5. □

5 Conclusion

We have considered the higher-order modal fixed point logic [20] (HFL) and its fragment without negations, and we have shown that both formalisms are equally expressive. More precisely, we have defined a procedure for transforming any closed HFL formula φ denoting a state predicate into an equivalent formula $\mathsf{nnf}(\varphi)$ without negations of size $\mathscr{O}(|\varphi|\cdot|\mathsf{vars}(\varphi)|)$. The procedure works in two phases: in a

first phase, a transformation we called *monotonization* eliminates all negations and represents arbitrary functions of type $\tau \to \sigma$ by functions of type $\tau \to \tau \to \sigma$ by distinguishing positive and negative usage of the function parameter. The price to pay for this transformation is an exponential blow-up in the size of the formula. If the formula is represented as a circuit, however, the blow-up is only linear. The second phase of our negation elimination procedure thus consists in implementing the sharing of common subformulas using higher-orderness. Thanks to this second phase, our procedure yields a negation-free formula $\mathsf{nnf}(\varphi)$ of size $\mathscr{O}(\mathsf{size}(\varphi) \cdot |\mathsf{vars}(\varphi)|)$, hence quadratic in the worst case in the size of the original formula φ.

Typed versus Untyped Negation Elimination Our monotonization procedure is *type-directed*: the monotonization of $\varphi\ \psi$ depends on the variance of φ, that is statically determined by looking at the type of φ. One might wonder if we could give a negation elimination that would not be type-directed. A way to approach this question is to consider an untyped conservative extension of the logic where we do not have to care about the existence of the fixed points – for instance, one might want to interprete $\mu X.\varphi(X)$ as the inflationary "fixed point" [7]. We believe that we could adapt our monotonization procedure to this setting, and it would indeed become a bit simpler: we could always monotonize $\varphi\ \psi$ "pessimistically", as if φ were neither a monotone nor an antitone function. For instance, the formula $\mu X.(\lambda Y.Y)\ X$ would be translated into $\mu X.(\lambda Y,\overline{Y}.Y)\ X\ \neg X$.

In our typed setting, it is crucial to use the type-directed monotonization we developed, because monotizing pessimistically might yield ill-typed formulas. In an untyped setting, a pessimistic monotonization is possible, but it yields less concise formulas, and it looses the desirable property that $\mathsf{nnf}(\mathsf{nnf}(\varphi)) = \mathsf{nnf}(\varphi)$.

So types, and more precisely variances, seem quite unavoidable. However, strictly speaking, the monotonization we introduced is *variance-directed*, and not really type-directed. In particular, our monotonization might be extended to the untyped setting, relying on some other static analysis than types to determine the variances of all functional subformulas.

Sharing and Quadratic Blow-Up The idea of sharing subterms of a λ-term is reminiscent to implementations of λ-terms based on hash-consing [8, 11] and to compilations of the λ calculus into interaction nets [13, 16, 10]. We showed how sharing can be represented directly in the λ-calculus, whereas hash-consing and interaction nets are concerned with representing sharing either in memory or as a circuit. We compile typed λ-terms into typed λ-terms; a consequence is that we do not manage to share subterms that are syntactically identical but have either different types or are typed using different type assumptions for their free variables. This is another difference with hash consing and interaction nets, where syntactic equality is enough to allow sharing subterms. It might be the case that we could allow more sharing if we did not compile into a simply typed λ-calculus but in a ML-like language with polymorphic types.

An interesting issue is the quadratic blow-up of our implementation of "λ-circuits". One might wonder wether a more succinct negation elimination is possible, in particular a negation elimination with linear blow-up. To answer this problem, it would help to answer the following simpler problem: *given a λ-term t with n syntactically distinct subterms, is there an effectively computable λ-term t' of size $\mathscr{O}(n)$ such that $t =_{\beta\eta} t'$*? We leave that problem for future work.

References

[1] Roland Axelsson, Martin Lange & Rafal Somla (2007): *The Complexity of Model Checking Higher-Order Fixpoint Logic*. *Logical Methods in Computer Science* 3(2), doi:10.2168/LMCS-3(2:7)2007.

[2] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague & Olivier Serre (2013): *C-SHORe: a collapsible approach to higher-order verification*. In: *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pp. 13–24, doi:10.1145/2500365.2500589.

[3] Florian Bruse (2014): *Alternating Parity Krivine Automata*. In Erzsbet Csuhaj-Varj, Martin Dietzfelbinger & Zoltn sik, editors: *Mathematical Foundations of Computer Science 2014, Lecture Notes in Computer Science* 8634, Springer Berlin Heidelberg, pp. 111–122, doi:10.1007/978-3-662-44522-8_10.

[4] Thierry Cachat (2003): *Higher Order Pushdown Automata, the Caucal Hierarchy of Graphs and Parity Games*. In: *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, pp. 556–569, doi:10.1007/3-540-45061-0_45.

[5] Arnaud Carayol & Olivier Serre (2012): *Collapsible Pushdown Automata and Labeled Recursion Schemes: Equivalence, Safety and Effective Selection*. In: *LICS*, pp. 165–174, doi:10.1109/LICS.2012.73.

[6] Werner Damm (1982): *The IO- and OI-hierarchies*. *Theoretical Computer Science* 20(2), pp. 95 – 207, doi:10.1016/0304-3975(82)90009-3.

[7] Anuj Dawar, Erich Grädel & Stephan Kreutzer (2004): *Inflationary fixed points in modal logic*. *ACM Trans. Comput. Log.* 5(2), pp. 282–315, doi:10.1145/976706.976710.

[8] Jean-Christophe Filliâtre & Sylvain Conchon (2006): *Type-safe Modular Hash-consing*. In: *Proceedings of the 2006 Workshop on ML*, ML '06, ACM, New York, NY, USA, pp. 12–19, doi:10.1145/1159876.1159880.

[9] Koichi Fujima, Sohei Ito & Naoki Kobayashi (2013): *Practical Alternating Parity Tree Automata Model Checking of Higher-Order Recursion Schemes*. In: *Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings*, pp. 17–32, doi:10.1007/978-3-319-03542-0_2.

[10] Georges Gonthier, Martín Abadi & Jean-Jacques Lévy (1992): *The Geometry of Optimal Lambda Reduction*. In: *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92, ACM, New York, NY, USA, pp. 15–26, doi:10.1145/143165.143172.

[11] Jean Goubault (1993): *Implementing Functional Languages with Fast Equality Sets and Maps: an Exercise in Hash Cons*. In: *Journées Francophones des Langages Applicatifs (JFLA'93)*, Annecy, pp. 222–238.

[12] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong & Olivier Serre (2008): *Collapsible Pushdown Automata and Recursion Schemes*. In: *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pp. 452–461, doi:10.1109/LICS.2008.34.

[13] John Lamping (1990): *An Algorithm for Optimal Lambda Calculus Reduction*. In: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, ACM, New York, NY, USA, pp. 16–30, doi:10.1145/96709.96711.

[14] Martin Lange & Étienne Lozes (2014): *Capturing Bisimulation-Invariant Complexity Classes with Higher-Order Modal Fixpoint Logic*. In: *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, pp. 90–103, doi:10.1007/978-3-662-44602-7_8.

[15] Olivier Laurent (2002): *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II.

[16] Ian Mackie (1998): *YALE: Yet Another Lambda Evaluator Based on Interaction Nets*. In: *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, ICFP '98, ACM, New York, NY, USA, pp. 117–128, doi:10.1145/289423.289434.

[17] A. N. Maslov. (1976): *Multilevel stack automata*. *Problems of Information Transmission* 12, pp. 38–43.

[18] Sylvain Salvati & Igor Walukiewicz (2014): *Krivine machines and higher-order schemes*. *Inf. Comput.* 239, pp. 340–355, doi:10.1016/j.ic.2014.07.012.

[19] Taku Terao & Naoki Kobayashi (2014): *A ZDD-Based Efficient Higher-Order Model Checking Algorithm*. In: *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, pp. 354–371, doi:10.1007/978-3-319-12736-1_19.

[20] Mahesh Viswanathan & Ramesh Viswanathan (2004): *A Higher Order Modal Fixed Point Logic*. In Ph. Gardner & N. Yoshida, editors: *CONCUR, Lecture Notes in Computer Science* 3170, Springer, pp. 512–528, doi:10.1007/978-3-540-28644-8_33.