

General Translation

The general translation splits into three parts: the translation of the implementation ($\llbracket M \rrbracket_k$) and the translations of the specification, detailed below.

- $\llbracket x \rrbracket_k^i$ denotes the translation of x with bound k , starting from state i .
- ${}_l \llbracket x \rrbracket_k^i$ denotes the translation of x with bound k , starting from state i , with a loop in the transition relation from state k to state l .
- ${}_l L_k$ denotes the transition function from state k to state l (*i.e.*, the loop transition).
- L_k is an abbreviation for $\bigvee_{l=0}^k {}_l L_k$.

The specification holds without loops in the transition relation.

The specification holds if there is some loop ${}_l L_k$ in the transition relation.

$$\llbracket M, f \rrbracket_k := \llbracket M \rrbracket_k \wedge \left(\left(\neg L_k \wedge \llbracket f \rrbracket_k^0 \right) \vee \bigvee_{l=0}^k ({}_l L_k \wedge {}_l \llbracket f \rrbracket_k^0) \right)$$

f	$\llbracket f \rrbracket_k^i$	${}_l \llbracket f \rrbracket_k^i$
$\mathbf{G} f_1$	false	$\bigwedge_{j=\min(i,l)}^k {}_l \llbracket f_1 \rrbracket_k^j$
$\mathbf{F} f_1$	$\bigvee_{j=i}^k \llbracket f_1 \rrbracket_k^j$	$\bigvee_{j=\min(i,l)}^k {}_l \llbracket f_1 \rrbracket_k^j$
$\mathbf{X} f_1$	$i < k \wedge \llbracket f_1 \rrbracket_k^{i+1}$	$(i < k \wedge {}_i \llbracket f_1 \rrbracket_k^{i+1}) \vee (i = k \wedge {}_l \llbracket f_1 \rrbracket_k^l)$
$f_1 \mathbf{U} f_2$	$\bigvee_{j=i}^k (\llbracket f_2 \rrbracket_k^j \wedge \bigwedge_{n=i}^{j-1} \llbracket f_1 \rrbracket_k^n)$	$\bigvee_{j=i}^k ({}_l \llbracket f_2 \rrbracket_k^j \wedge \bigwedge_{n=i}^{j-1} {}_l \llbracket f_1 \rrbracket_k^n)$
$f_1 \mathbf{R} f_2$	$\bigvee_{j=i}^k (\llbracket f_1 \rrbracket_k^j \wedge \bigwedge_{n=i}^j \llbracket f_2 \rrbracket_k^n)$	$\bigwedge_{j=\min(i,l)}^k {}_l \llbracket f_2 \rrbracket_k^j \vee \bigvee_{j=i}^k ({}_l \llbracket f_1 \rrbracket_k^j \wedge \bigwedge_{n=i}^j {}_l \llbracket f_2 \rrbracket_k^n)$
		$\vee \bigvee_{j=l}^{i-1} ({}_l \llbracket f_1 \rrbracket_k^j \wedge \bigwedge_{n=i}^k {}_l \llbracket f_2 \rrbracket_k^n \wedge \bigwedge_{n=l}^j {}_l \llbracket f_2 \rrbracket_k^n)$

The Size of Clause Forms

[NRW98] describes renaming methods for reducing the size of a clause form conversion; in doing so, the authors define a function $p(\phi)$ which determines the size of the conversion of formula ϕ . The paper deals with first order logic; here we present the definition for propositional logic.

ϕ	$p(\phi)$	$\bar{p}(\phi)$
$\neg\phi_1$	$\bar{p}(\phi_1)$	$p(\phi_1)$
$\phi_1 \wedge \phi_2$	$p(\phi_1) + p(\phi_2)$	$\bar{p}(\phi_1)\bar{p}(\phi_2)$
$\bigwedge_{i=a}^b \phi_1$	$\sum_{i=a}^b p(\phi_1)$	$\prod_{i=a}^b \bar{p}(\phi_1)$
[1] $\bigwedge_{i=a}^b \phi_1$	$(b - a + 1) p(\phi_1)$	$\bar{p}(\phi_1)^{b-a+1}$
$\phi_1 \vee \phi_2$	$p(\phi_1)p(\phi_2)$	$\bar{p}(\phi_1) + \bar{p}(\phi_2)$
$\bigvee_{i=a}^b \phi_1$	$\prod_{i=a}^b p(\phi_1)$	$\sum_{i=a}^b \bar{p}(\phi_1)$
[1] $\bigvee_{i=a}^b \phi_1$	$p(\phi_1)^{b-a+1}$	$(b - a + 1) \bar{p}(\phi_1)$
$\phi_1 \Rightarrow \phi_2$	$\bar{p}(\phi_1)p(\phi_2)$	$p(\phi_1) + \bar{p}(\phi_2)$
$\phi_1 \Leftrightarrow \phi_2$	$p(\phi_1)\bar{p}(\phi_2) + \bar{p}(\phi_1)p(\phi_2)$	$p(\phi_1)p(\phi_2) + \bar{p}(\phi_1)\bar{p}(\phi_2)$

[1] where $i \notin \text{fv}(\phi_1)$

- The function $p(\phi)$ is exact, not a heuristic, if the conversion is based on the usual transformations (e.g. $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$).
- It does not deal with obvious simplifications (e.g. $a \vee a = a$) so it may be an upper bound if these are applied in the conversion algorithm.

General Size

The expected number of clauses produced by the general translation can be determined.

- The size of the translation of a transition from one state to another is independant of which states are involved.
- Define $t = p(T(x, y))$ for any x, y .
- Given that $\llbracket M \rrbracket_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(i, i+1)$, deduce that $p(\llbracket M \rrbracket_k) = p(I) + t^k$.
- Notice that ${}_i L_k = T(k, i)$, hence $p({}_i L_k) = t$.
- $p(L_k) = \prod_{i=0}^k p({}_i L_k) = t^{k+1}$
- $p(\neg L_k) = \sum_{i=0}^k \bar{p}({}_i L_k) = (k+1)\bar{t}$ (writing \bar{t} for $\bar{p}(T(x, y))$)

$$\llbracket M, f \rrbracket_k := \llbracket M \rrbracket_k \wedge \left(\boxed{(\neg L_k \wedge \llbracket f \rrbracket_k^0)} \vee \boxed{\bigvee_{l=0}^k ({}_l L_k \wedge {}_l \llbracket f \rrbracket_k^0)} \right)$$

$$p(\llbracket M, f \rrbracket_k) = p(I) + t^k + \left(\boxed{((k+1)\bar{t} + p(\llbracket f \rrbracket_k^0))} \times \boxed{\prod_{l=0}^k (t + p({}_l \llbracket f \rrbracket_k^0))} \right)$$

Observations:

- The overall size of the translation is ruled by the product of the loop and non-loop sizes.
- The size of the loop translation is itself a product, with terms containing an exponential in k .

Restricted CTL

The translation of the specification is defined recursively on its operators; if the specification is restricted to a single CTL operator applied to a propositional formula then the translations can be simplified.

For specification $f = \mathbf{O}_1 f_1$ where \mathbf{O}_1 is some unary operator, or $f = f_1 \mathbf{O}_2 f_2$ where \mathbf{O}_2 is some binary operator,

- The zero starting point can be substituted into equations, simplifying, and also eliminating blocks with ranges $l-0$.
- The translation of f_1 and f_2 with and without loops is $f_1(s_i)$ and $f_2(s_i)$ at state i respectively.
- For $\mathbf{O}_1 = \mathbf{X}$, we know that $i < k$, since $i = 0$.

f	$\llbracket f \rrbracket_k^0$	${}_l \llbracket f \rrbracket_k^0$
$\mathbf{G} f_1$	false	$\bigwedge_{j=0}^k f_1(s_j)$
$\mathbf{F} f_1$	$\bigvee_{j=0}^k f_1(s_j)$	$\bigvee_{j=0}^k f_1(s_j)$
$\mathbf{X} f_1$	$f_1(s_1)$	$f_1(s_1)$
$f_1 \mathbf{U} f_2$	$\bigvee_{j=0}^k (f_2(s_j) \wedge \bigwedge_{n=0}^{j-1} (f_1(s_n)))$	$\bigvee_{j=0}^k (f_2(s_j) \wedge \bigwedge_{n=0}^{j-1} (f_1(s_n)))$
$f_1 \mathbf{R} f_2$	$\bigvee_{j=0}^k (f_1(s_j) \wedge \bigwedge_{n=0}^j (f_2(s_n)))$	$\bigwedge_{j=0}^k f_2(s_j) \vee \bigvee_{j=0}^k (f_1(s_j) \wedge \bigwedge_{n=0}^j (f_2(s_n)))$

Observations:

- For any $f, l \notin \text{fv}({}_l \llbracket f \rrbracket_k^0)$, so $\bigvee_{l=0}^k ({}_l L_k \wedge {}_l \llbracket f \rrbracket_k^0)$ becomes $L_k \wedge {}_0 \llbracket f \rrbracket_k^0$.
- There are three types of operator:
 1. No non-loop part — $\mathbf{O}_1 = \mathbf{G}$
 2. Identical loop and non-loop parts — $\mathbf{O}_1 \in \{\mathbf{X}, \mathbf{F}\}, \mathbf{O}_2 = \mathbf{U}$
 3. Differing loop and non-loop parts — $\mathbf{O}_2 = \mathbf{R}$

Sizes with Restricted CTL

By substituting the simplified translations of operators into the corresponding general translations, we can predict the sizes of translations for each operator.

f	$p(\llbracket M, f \rrbracket_k)$
$\mathbf{G} f_1$	$p(I) + t^k + t^{k+1} + (k+1)p(f_1)$
$\mathbf{F} f_1$	$p(I) + t^k + p(f_1)^{k+1}$
$\mathbf{X} f_1$	$p(I) + t^k + p(f_1)$
$f_1 \mathbf{U} f_2$	$p(I) + t^k + \prod_{j=0}^k (p(f_2) + j p(f_1))$
$f_1 \mathbf{R} f_2$	$p(I) + t^k + t^{k+1} \prod_{j=0}^k (p(f_1) + (j+1)(p(f_2))) + (k+1)p(f_2) \prod_{j=0}^k (p(f_1) + (j+1)(p(f_2)))$

- It is a useful guide to determine the number of clauses produced by an example.
- In this case, we take $p(f_1) = p(f_2) = 1$, $p(I) = t = k = 2$. That is, a very small, simple circuit, with a bound of 2 and an atomic parameter to the operator in the specification.
- Observe that for a propositional formula, the size of the encoding is independent to the state. For any i , the value of $p(f(s_i))$ is constant, denoted by $p(f)$.

f	$p(\llbracket M, f \rrbracket_k)$	Size
$\mathbf{G} f_1$	$p(I) + t^k + t^{k+1} + (k+1)$	17
$\mathbf{F} f_1$	$p(I) + t^k + 1$	7
$\mathbf{X} f_1$	$p(I) + t^k + 1$	7
$f_1 \mathbf{U} f_2$	$p(I) + t^k + (k+1)!$	12
$f_1 \mathbf{R} f_2$	$p(I) + t^k + t^{k+1} (k+2)! + (k+1)(k+2)!$	270

Sizes with Restricted CTL (2)

Observations:

- The translation of $f_1 \mathbf{R} f_2$, corresponding to CTL $\mathbf{A}[f_1 \mathbf{U} f_2]$, is much larger than the others, even for a very simple example with extensive simplifications.
- We have already come a long way. Without the specific rearrangement at the bottom of page 5, we have

$$p(I) + t^k + ((k+1)\bar{t} + (k+2)!)(t^{k+1} + (k+1)(k+2)!)$$

which has size 2406.

- Without the observation that $l \notin \text{fv}(\llbracket f \rrbracket_k^0)$, we have the even worse situation of

$$p(I) + t^k + ((k+1)\bar{t} + (k+2)!)(t + (k+1)(k+2)!)^{k+1}$$

which has size 12 156 726.

- These observations indicate potential problems with non-restricted CTL, as the simplifications used so far will not be available.

Taming Until

CTL specifications of the form $A[f_1 \mathbf{U} f_2]$ appear to be the most problematic in translation, but are also likely to be widely used in real specifications. We propose using renaming as defined in [NRW98] to improve the encoding. Again, we present the definition for proposition rather than first order logic.

- The *renaming* of a subformula ψ in formula ϕ is given by $\phi[x/\psi] \wedge d(x, \psi)$. That is, the subformula is replaced by a new variable, which is *defined* by the conjunction of function $d(x, \psi)$.
- The definition of a renaming depends on the *polarity* with which ψ occurs. That is, if $\phi[x/\psi]$ were converted to negation normal form, whether x would appear negated (polarity -1), positive (polarity -1), or both (polarity 0).
- For polarity 1, $d(x, \psi) \hat{=} (x \Rightarrow \psi)$
- For polarity -1, $d(x, \psi) \hat{=} (\psi \Rightarrow x)$
- For polarity 0, $d(x, \psi) \hat{=} (x \Leftrightarrow \psi)$

To identify the subformulae to rename, we can examine the translation of $f_1 \mathbf{R} f_2$.

- Obviously, if there is a repeated subformula then renaming it can be beneficial, as only one definition is used for several renamings.
- We can rewrite the translation (substituting the operator and replacing the loop translation with its equivalent in terms of the non-loop translation) as

$$\llbracket M \rrbracket_k \wedge (\underline{\llbracket f_1 \mathbf{R} f_2 \rrbracket_k^0} \vee L_k) \wedge \left(\bigwedge_{j=0}^k f_2(s_j) \vee \underline{\llbracket f \mathbf{R} f_2 \rrbracket_k^0} \right)$$

- By renaming the repeated sections (underlined above) we obtain

$$\llbracket M \rrbracket_k \wedge (\rho \vee L_k) \wedge \left(\bigwedge_{j=0}^k f_2(s_j) \vee \rho \right) \wedge (\rho \Rightarrow \llbracket f \mathbf{R} f_2 \rrbracket_k^0)$$

Taming Until (2)

- Can the encoding be improved by a second renaming? In the abstract another is proposed.
- We need to examine the relationship between the encoding and its size so that the next biggest influence can be determined.
- From above:

$$\begin{aligned} \llbracket M \rrbracket_k \wedge (\llbracket f \rrbracket_k^0 \vee L_k) \wedge (\bigwedge_{j=0}^k f_2(s_j) \vee \llbracket f \rrbracket_k^0) \\ \downarrow \qquad \qquad \downarrow \qquad \qquad \downarrow \\ p(I) + t^k + (p(\llbracket f \rrbracket_k^0) \times t^{k+1}) + ((k+1)p(f_2) \times p(\llbracket f \rrbracket_k^0)) \end{aligned}$$

- The largest term here is the exponential t^{k+1} produced by L_k . However, the other term in this disjunction has already been renamed. By renaming this we replace a clause of the form $a \vee b$ with two clauses $(a \vee \lambda) \wedge (\neg \lambda \vee b)$
- Similarly, the conjunction term in the other disjunction is not a candidate for renaming.
- We deduce that one renaming is sufficient.
- The table of sizes for the example becomes

f	$p(\llbracket M, f \rrbracket_k)$	Size
$\mathbf{G} f_1$	$p(I) + t^k + t^{k+1} + (k+1)$	17
$\mathbf{F} f_1$	$p(I) + t^k + 1$	7
$\mathbf{X} f_1$	$p(I) + t^k + 1$	7
$f_1 \mathbf{U} f_2$	$p(I) + t^k + (k+1)!$	12
$f_1 \mathbf{R} f_2$	$p(I) + t^k + t^{k+1} + (k+1) + (k+2)!$	41

- This is better than the 270 above, and far better than the 12 156 726 in the original translation.

References

- [NRW98] Andreas Nonnengart, Georg Rock, and Christoph Weidenbach. On generating small clause normal forms. In Claude Kirchner and Hélène Kirchner, editors, *Automated Deduction — CADE-15 International Conference*, LNAI 1421, pages 397–411. Springer, 1998.