

Kconfig 语言的形式语义学

Steven She¹ and Thorsten Berger²

¹shshe@gsd.uwaterloo.ca, 加拿大, 滑铁卢大学

²berger@informatik.uni-leipzig.de, 德国, 莱比锡城大学

2010 年 1 月

摘 要

Kconfig 语言定义了一组在配置中被赋值的符号。我们依据 xconfig 配置程序表现出的行为来描述 Kconfig 语言的语义。我们为 Kconfig 语言中的概念假设一个抽象的语法陈述, 并且将由具体到抽象的语法转化的细节总结成文档。

1. 抽象语法

标识符和表达式. 我们开始定义在 Kconfig 语言中可利用的预备概念。假设 Id 为一个识别符号名的有限集合——更确切地说, $\text{Id} \in \mathcal{P}(\text{String})$ 。假设 $\text{Const} = \text{Tri} \cup \text{String} \cup \text{Hex} \cup \text{Int}$ 为一个对各个特征和变量(如表达式中的常量)可赋值的值的集合。 $\text{Tri} = \{0t, 1t, 2t\}$, Tri 是有序的, $0t < 1t < 2t$ 。 Tri , String , Hex , 和 Int 的域是不连接的(即互斥的)。我们现在可以定义一个 Kconfig 语言的表达式。 $\text{KExpr}(\text{Id})$ 是 Id 依据下面的语法生成的表达式的一个集合, $e \in \text{KExpr}(\text{Id})$, $iv \in \text{Id} \cup \text{Const}$, $\otimes \in \{\text{or}, \text{and}\}$, $\ominus \in \{=, =6\}$:

$$e ::= e \otimes e \mid \text{not } e \mid iv \ominus iv \mid iv (1)$$

Evaluating a KExpr 返回一个三态值(即 $v \in \text{Tri}$)。我们会在 2.2 节定义估值函数的语义。

Kconfig 模式. Kconfig 在 Kconfig 语言中表示所有可能模式的集合。因此一个单 Kconfig 模型 $m \in \text{Kconfig}$ 是一个 configs 和 choices 集合构成的元组。

Kconfig 定义为:

$$\text{Kconfig} = \text{P}(\text{Configs}) \times \text{P}(\text{Choices}) \quad (2)$$

给定一个 Kconfig 模型 $m \in \text{Kconfig}$, 我们定义 $m\text{config}$ 为适用于 configs 集合的缩写。

本文定义的语义直接反映了 Linux 内核的 `make xconfig` 工具的行为, 在某些特定实例中, 可能表现得与 Kconfig 语言开发人员最初所想要的不尽相同。比如为防止反向依赖, 本文明确指出下面的声明: “应该小心地使用 `Select`。Select 会强制一个符号为值而不考虑相关性。如果滥用 `select`, 你可能会选择一个 `foo` 符号, 即使 `foo` 依赖一个没设置的符号 `bar`。”

Configs 是 Kconfig 模型中的第一个成分。一个 config 用类型、提示条件(定义 config 变得用户可变, 一个默认值列表, 一个表明自身反向依赖的表达式, 这个条件会通过 `select` 语句强制启用这个特性), 和一个范围的集合(configs 或 hex 类型)定义了一个唯一的标识符类型。我们定义 Configs 如下:

$$\text{Configs} = \text{Id} \times \text{Type} \times \text{KExpr}(\text{Id}) \times \text{Default} * \times \text{KExpr}(\text{Id}) \times \text{P}(\text{Range}) \quad (3)$$

在这里,

- $\text{Type} = \{\text{boolean}, \text{tristate}, \text{int}, \text{hex}, \text{string}\}$ 表示一个类型, 就是 config 的可能值。
- $\text{Default} = \text{KExpr}(\text{Id}) \times \text{KExpr}(\text{Id})$ 表示默认值。第一个 KExpr 表示一个默认表达式(即赋值给符号的) 第二个 KExpr 表示默认值变得有效需要的条件。
- $\text{Range} = (\text{Int} \cup \text{Hex} \cup \text{Id}) \times (\text{Int} \cup \text{Hex} \cup \text{Id}) \times \text{KExpr}(\text{Id})$ 是上下界和条件的三倍。注意在范围的上下界少了 `Tri`; 这是因为如下段所述, 范围在 `int` 和 `hex` 类型 configs 上才有效。

我们进一步定义一个函数 $\text{Id}(m)$ 来表示在 model m 中的标识符:

$$\text{Id}(m) = \{n \mid (n, _, _, _, _) \in m\text{config}\} \quad (4)$$

Kconfig 第二个成分是 choice 结点的集合。一个 choice 是一个在配置中没定义符号的抽象概念, 然而, 它在其嵌套的元素上施加额外的约束。我们定义 choices 为一个类型的四倍组成, `boolean` 或者 `tristate` 是仅有的有效类型, 标记

表明这个 choice 是强制的，随着一个表明其成员的标识符之后的提示条件。集合 Choices 定义为：

$$\text{Choices} = \{\text{boolean}, \text{tristate}\} \times \text{Bool} \times \text{KExpr}(\text{Id}) \times \text{P}(\text{Id}(\text{m})) \quad (5)$$

结构完整性规则. 给定一个元素 $(_, t, _, _, \text{rev}, \text{rngs}) \in \text{Configs}$, config 如果满足下列条件即是结构完整的：

- int、hex 或者 string 类型 configs 的反向依赖必为 0t。换句话说，没有一个非 boolean 类型或者 tristate 类型的 config 会被选择。

$$(\text{rev} \neq 0t) \Rightarrow t = \text{boolean} \vee t = \text{tristate} \quad (6)$$

- 可以用数值类型(int 类型或 hex 类型)的 configs 唯一确定范围。因此，一个 config 若是结构完整的必有下列约束：

$$(|\text{rngs}| > 0) \Rightarrow t = \text{int} \vee t = \text{hex} \quad (7)$$

简述具体语法转化. Menuconfigs 和 menus 是 Kconfig 语言中具体语法的第一类概念，然而这些概念都没有出现在抽象语法中。首先，menuconfigs 与 configs 语义同源，只在配置程序方面有区别；因此，我们在抽象语法中模拟 menuconfigs 为 configs。

Menus 不定义符号，因此 menus 不在配置中。然而，menus 可以在其嵌套元素上施加约束。我们可以在所有嵌套符号上对提示、默认和范围条件进行语法的重写。语法重写细节将在下文提供。

2. 语义

2.1 语义域

一个 Kconfig 模型的配置是一个值 v 的赋值, 值 $v \in \text{Const}$ 到 config 元素。
因此, 所有可能的配置集合定义为:

$$\text{Confs} = \text{Id} \rightarrow [\text{Const}] \quad (8)$$

如果 $c \in \text{Confs}$ 且 $x \in \text{Id}$, 我们为了在配置 c 下参考标识符 x 的值而写作 $c(x)$,
现在我们在配置的集合方面定义了一个 Kconfig 模型的语义。

因此 $P(\text{Confs})$ 就是我们的语义域。我们定义 $[[\cdot]]_{\text{kconfig}}$ 为求一个 Kconfig 模型值, 并返回有效配置集合的函数:

$$[[\cdot]]_{\text{kconfig}} : \text{Kconfig} \rightarrow P(\text{Confs}) \quad (9)$$

2.2 全局函数

我们从一些完全符合语义的函数的定义开始。首先, 我们使用 `bool`, 用 `boolean` 逻辑定义了一个 `tristate` 值的解释 : $\text{Tri} \rightarrow \text{Bool}$, $\text{Bool} = \{T, F\}$:

$$\text{bool}(v) = \begin{cases} F & \text{iff } v = 0t \\ T & \text{iff } v = 1t \vee v = 2t \end{cases} \quad (10)$$

此外, 我们定义了一个函数 $\text{access} : (\text{Id} \cup \text{Const}) \times \text{Confs} \rightarrow \text{Const}$ 取回一个常量或符号的值。当一个标识符有 \perp 的值(将在方程式 14 中定义), 那么 `access` 函数以字符串形式返回标识符自身:

$$\text{access}(iv, c) = \begin{cases} iv & \text{iff } iv \in \text{Const} \vee (iv \in \text{Id} \wedge c(iv) = \perp) \\ c(iv) & \text{iff otherwise} \end{cases} \quad (11)$$

接下来, 我们定义函数 $\text{toStr} : \text{Const} \rightarrow \text{String}$ 表示常量值到一个字符串表示的转化。在 `toStr` 下列的定义中, 设 $i \in \text{Int}$, $h \in \text{Hex}$ 且 $s \in \text{String}$:

$$\begin{aligned} \text{toStr}(0_t) &= "n" \quad \text{toStr}(1_t) = "m" \quad \text{toStr}(2_t) = "y" \\ \text{toStr}(i) &= "" + i \quad \text{toStr}(h) = "0x" + h \quad \text{toStr}(s) = s \end{aligned} \quad (12)$$

`+o` 操作符是字符串连接符。

最后, 函数 $\text{eval} : \text{KExpr}(\text{Id}) \rightarrow \text{Tri}$ 描述在 Kconfig 语言中一个 KExpr 的求值。我们用 e_1, e_2 递归地定义 `eval`, $e_1, e_2 \in \text{KExpr}(\text{Id})$ 且 $iv, ivx, ivy \in \text{Id} \cup \text{Const}$:

$$\begin{aligned}
\text{eval}(iv_x = iv_y, c) &= \begin{cases} 2t \text{ iff } toStr(access(iv_x, c)) = toStr(access(iv_y, c)) \\ 0t \text{ iff otherwise} \end{cases} \\
\text{eval}(iv_x \neq iv_y, c) &= 2t - \text{eval}(iv_x = iv_y, c) \\
\text{eval}(\text{not } e_1, c) &= 2t - \text{eval}(e_1, c) \\
\text{eval}(e_1 \text{ and } e_2, c) &= \min(\text{eval}(e_1, c), \text{eval}(e_2, c)) \\
\text{eval}(e_1 \text{ or } e_2, c) &= \max(\text{eval}(e_1, c), \text{eval}(e_2, c)) \\
\text{eval}(iv, c) &= \begin{cases} viv \text{ iff } viv = access(iv, c) \wedge viv \in Tri \\ 0t \text{ off otherwise} \end{cases}
\end{aligned} \tag{13}$$

2.3 估值函数

Kconfig 模型. 我们开始定义 $\llbracket \cdot \rrbracket_{\text{kconfig}}$. Given 给定一个 Kconfig 模型 $m \in \text{Kconfig}$, 这个模型的语义就是是涵盖这个 model, configs 和 choices 的所有表示的交集。换句话说, 一个 Kconfig 模型的有效配置的集合就是满足所有表示的配置。

$\llbracket \cdot \rrbracket_{\text{kconfig}} : \text{Kconfig} \rightarrow \text{Confs}$ 定义为:

$$\begin{aligned}
\llbracket m \rrbracket_{\text{kconfig}} &= \left(\bigcap_{n \in m_{\text{config}}} \llbracket n \rrbracket_{\text{type}} \cap \llbracket n \rrbracket_{\text{bounds}} \cap \llbracket n \rrbracket_{\text{default}} \cap \llbracket n \rrbracket_{\text{range}} \right) \\
&\quad \cap \llbracket m \rrbracket_{\text{module}} \cap \llbracket m \rrbracket_{\text{undeclared}} \tag{14}
\end{aligned}$$

类型. 第一个表示约束属于 config 类型施加的约束。一个 config 类型对于各个域限定其有效值。 $\llbracket \cdot \rrbracket_{\text{type}} : \text{Configs} \rightarrow \text{Confs}$ 定义为:

$$\llbracket n, t, _, _, _, _ \rrbracket_{\text{type}} = \begin{cases} \{c \in \text{Confs} \mid c(n) \in \text{Tri} \setminus \{1_t\}\} & \text{iff } t = \text{boolean} \\ \{c \in \text{Confs} \mid c(n) \in \text{Tri}\} & \text{iff } t = \text{tristate} \\ \{c \in \text{Confs} \mid c(n) \in \text{String}\} & \text{iff } t = \text{string} \\ \{c \in \text{Confs} \mid c(n) \in \text{Hex} \cup \{""\}\} & \text{iff } t = \text{hex} \\ \{c \in \text{Confs} \mid c(n) \in \text{Int} \cup \{""\}\} & \text{iff } t = \text{int} \end{cases} \tag{15}$$

上下边界. 接下来, 边界表示一个 config 模拟的上下边界。下边界由一个 config 的反向依赖求值决定。联系到反向依赖用具体的语法模拟 select 语句的行为。

上边界被一个 config 的提示条件定义。这个表示对类型为 int, hex, 或 string 的 configs 无效, 因为我们的结构完整性规则, 反向依赖决定下边界的是 0t, eval 函数当评估一个值不在 Tri 中时返回 0t。

$[[\cdot]]_{\text{bounds}} : \text{Configs} \rightarrow \text{Confs}$ 定义为:

$$[[(n, _, \text{pro}, _, \text{rev})]]_{\text{bounds}} = \{c \in \text{Confs} \mid \text{eval}(c(n), c) \geq \text{Lower}(c) \wedge (\text{Upper}(c) < \text{Lower}(c) \vee \text{eval}(c(n), c) \leq \text{Upper}(c))\} \quad (16)$$

where $\text{Lower}(c) = \text{eval}(\text{rev}, c)$ and $\text{Upper}(c) = \text{eval}(\text{pro}, c)$.

Defaults. Kconfig 支持为一个 config 设置一个默认表达式. 这个默认表达式与决定 config 什么时候用户的可变提示条件相互作用。当提示条件满足时, 用户可以设置一个值。然而, 当它不满足时, 默认值就是这个 config 的值。 $[[\cdot]]_{\text{default}} : \text{Configs} \rightarrow \text{Confs}$ 定义为:

$$[[(n, _, _, \text{defs}, \text{rev}, _)]]_{\text{default}} = \{c \in \text{Confs} \mid \text{bool}(\text{eval}(\text{pro}, c)) \vee c(n) = \max(\text{eval}(\text{default}(\text{defs}, c)), \text{eval}(\text{rev}, c))\} \quad (17)$$

$\text{default} : \text{P}(\text{Default}) \times \text{Type} \times \text{Confs} \rightarrow \text{Const}$ 是一个默认模仿检索的函数. 联系到 defs 是一个默认值的列表 (且已排序). 一个默认值的效果依赖于它定义的 config 的类型. 如果 config 是 boolean 或 tristate 类型, 那么默认值就被评估为 Tri. 否则, 默认值必为 Const 或 Id 中的值. 设 Nil 为空列表并且 :: 为列表 cons 操作符. 设 $t_{\text{Tri}} \in \{\text{boolean}, \text{tristate}\}$ 并且 $t_{\text{Entry}} \in \{\text{int}, \text{hex}, \text{string}\}$. The default 函数被递归地定义, 所以我们开始定义它的基础情况

$$\begin{aligned} \text{default}(\text{Nil}, t_{\text{Tri}}, c) &= 0t \\ \text{default}(\text{Nil}, t_{\text{Entry}}, c) &= "" \end{aligned} \quad (18)$$

方程式 18 声明给定一个默认的空列表, 如果类型是 boolean, 或者 tristate, 或者是 (int, hex 或 string 类型的)空串之一, 我们返回 0t。接下来, 我们定义递归规则。在下面的方程式中, 我们将 list 分解为首尾两部分。首先, 我们描述 boolean 和 tristate 类型的函数:

$$\text{default}(e, \text{cond}) :: \text{rest}, t\text{Tri}, c = (\text{eval default}(e, \text{crest}, t) \text{Tri}, c) \text{ if otherwise} \\ \text{bool}(\text{eval}(\text{cond}, c)) \quad (19)$$

对于剩余类型:

$$\text{default}(e, \text{cond}) :: \text{rest}, t\text{Entry}, c = (\text{access}(\text{default}(e, c \text{rest}, t) \text{Entry}, c) \text{ if otherwise} \\ \text{bool}(\text{eval}(\text{cond}, c)) \quad (20)$$

范围. 范围利用 int 或 hex 的 configs 值的上下界。 $[[\bullet]]\text{range} : \text{Configs} \rightarrow \text{Confs}$ 定义为:

$$[[n, _ _ _ _, \text{rngs}]]\text{range} = \{c \in \text{Confs} \mid \forall(l, u, \text{cond}) \in \text{rngs}.$$

$$\text{bool}(\text{eval}(\text{cond}, c)) \rightarrow c(n) \geq \text{access}(l, c) \wedge c(n) \leq \text{access}(u, c)\} \quad (21)$$

选择. 选择限定了可被选择的成员的数量 (即有一个值大于 0t). 选择表示, $[[\bullet]]\text{choice} : \text{Choices} \rightarrow$

Confs 定义为:

$$[[(\text{boolOrT ri}, \text{isMand}, \text{prompt}, \text{mems})]]\text{choice} =$$

$$\{c \in \text{Confs} \mid \text{bool}(\text{eval}(\text{prompt}, c)) \rightarrow \text{Xor} \wedge \text{BChoice} \wedge \text{Mandatory}\} \quad (22)$$

Xor 定义有且只有一个成员可被设置为 2t 的条件:

$$\text{Xor} = \exists m1 \in \text{mems}. (m1 = 2t) \rightarrow (\forall m2 \in \text{mems} \setminus \{m1\}. m2 = 0t) \quad (23)$$

如果是一个 boolean 选择, 那么其成员唯一的有效值为 2t. 与 Xor 结合, 定义了一个 boolean 选择可能至多有一个成员的值不等于 0t 且必为 2t:

$$\text{BChoice} = (\text{boolOrT ri} = \text{boolean}) \rightarrow \exists m \in \text{mems}. c(m) = 2t \quad (24)$$

最后, 若这个选择是强制的, 则至少选择一个成员:

$$\text{Mandatory} = \text{isMand} \rightarrow \exists m \in \text{mems}. c(m) > 0t \quad (25)$$

模块, 一个专用的模块用于支持内核中的模块。Disabling 模块不接受 1t 声明 configs 并有效的将所有 tristate configs 转变为 boolean configs。 一个专用的

符号 m 用于在表达式中用具体的语法标识一个模块特性的依赖。有 m 依赖的 Confs 不能被选择(即必为 0t)

如果没有选择模块, 我们假定专用的 m 标识符在抽象语法中被扩展为模块了。

$$[[m]]\text{module} = \{c \in \text{Confs} \mid c(\text{modules}) = n \rightarrow \forall i \in \text{Id}. c(i) \neq 1t\} \quad (26)$$

	type	interpretation in tristate logic
X	tristate	$X = y$ or $X = m$
!X	tristate	$X = n$
X	boolean	$X = y$
!X	boolean	$X = n$
X	string	$X = \dots$ (some non-empty string)
!X	string	$X = \dots$
X	int	$X = i$ (some integer, including zero)
!X	int	$X = \dots$
X	hex	$X = i$ (some hex)
!X	hex	$X = \dots$

Table 1: Interpretation of propositional variables

Undeclared symbols. 我们也定义 undeclared symbols 的行为。Kconfig 语言支持引用我们在约束中没声明的符号。这些未声明的符号在我们的语义中被赋值为特殊符号 \perp 。在 2.2 节中的 eval 函数中，这个符号的用处将显而易见。The

$[[\bullet]]_{\text{undeclared}} : \text{Kconfig} \rightarrow \text{P}(\text{Confs})$ 表示定义为:

$$[[m]]_{\text{undeclared}} = \{c \in \text{Confs} \mid \forall x \in \text{Id} \setminus \text{Id}(m). c(x) = \perp\} \quad (27)$$

3.1-Var 命题语义

命题语义的目标是弱化完整语义的约束。

重写表达式规则. rewrite 是一个在表达式实现重写规则的部分函数. 函数 $\text{rewrite} : \text{KExpr}(\text{Id}) \rightarrow \text{KExpr}(\text{Id})$ 定义为:

$$\text{rewrite}(e) = \begin{cases} 0 & \text{if } (e \text{ is an variable} \wedge \text{typeof}(e) \in \{\text{int}, \text{hex}, \text{string}\}) \vee e = 0_t \\ 1 & \text{if } e = 1_t \vee e = 2_t \\ X \leftrightarrow Y & \text{if } e \text{ is } X = Y \text{ where } X \text{ and } Y \text{ are variables} \\ \text{Use Table 1} & \text{if } e \text{ is } X = \text{lit} \vee X \neq \text{lit} \end{cases}$$

(28)

我们进一步设函数 $\text{relax} : \text{KExpr}(\text{Id}) \rightarrow \text{KExpr}(\text{Id})$ 将表达式转化为 CNF，且移除子句相当于等式检查。T 函数用于降低隐式先行词(LHS)的约束。

语义. 在命题语义中, 设命题配置的集合为:

$$\text{Confs } p = \text{Id} \rightarrow \text{Bool} \quad (29)$$

函数 `default` 定义为 $\text{Id} \times \text{Default} * \times \text{Confsp} \rightarrow \text{Bool}$. 命题语义需要一个额外的参数提供标识符声明。

$$\text{default}(n, \text{defs}, c) = \begin{cases} \neg n & \text{if no default conditions are satisfied} \\ \text{rewrite}(n = \text{eval}(iv_i, c)) & \text{otherwise if } t \in \{\text{boolean, tristate}\}, \text{ where } iv_i \text{ is 1st matching default value} \\ n & \text{otherwise if } t \in \{\text{int, hex, string}\} \end{cases} \quad (30)$$

默认表示定义为:

$$[[[n, t, \text{vis}, \text{pro}, \text{defs}, \text{rev}, \text{rngs}]]] \text{default} = \{c \in \text{Confsp} \mid \text{eval}(\text{pro}, c) \vee \text{default}(n, \text{defs}, c)\} \quad (31)$$

反向依赖和可见性条件施加的模型约束定义为:

$$[[[n, t, \text{vis}, \text{pro}, \text{defs}, \text{rev}, \text{rngs}]]] \text{bounds} = \{c \in \text{Confsp} \mid (\text{eval}(\text{relax}(\text{rev}), c) \rightarrow c(n)) \wedge (c(n) \rightarrow \text{eval}(\text{vis}, c))\} \quad (32)$$

由于我们抽象各个 `config` 的值, 故我们忽略范围. 我们也假定模块 `config` 被启用, 因此在 `tristate configs` 中允许 `1t`.

$$[[[(\text{boolOrT } ri, \text{isMand}, \text{vis}, \text{mems})]]] \text{choice} = \\ (c \in \text{Confs} \mid \text{eval}(\text{vis}, c) \rightarrow \text{choose}(1, \text{ids}(\text{mems})) \wedge \text{isMand} \rightarrow m \in _ \text{mems } m!) \quad (33)$$